Fake COVID-19 survey hides ransomware in Canadian university attack

Published: 2020-10-27 · Archived: 2026-04-10 02:19:11 UTC

- Nebula support
- OneView support

- Nebula sign in
- OneView sign in
- Partner Portal sign in

THREATDOWN™

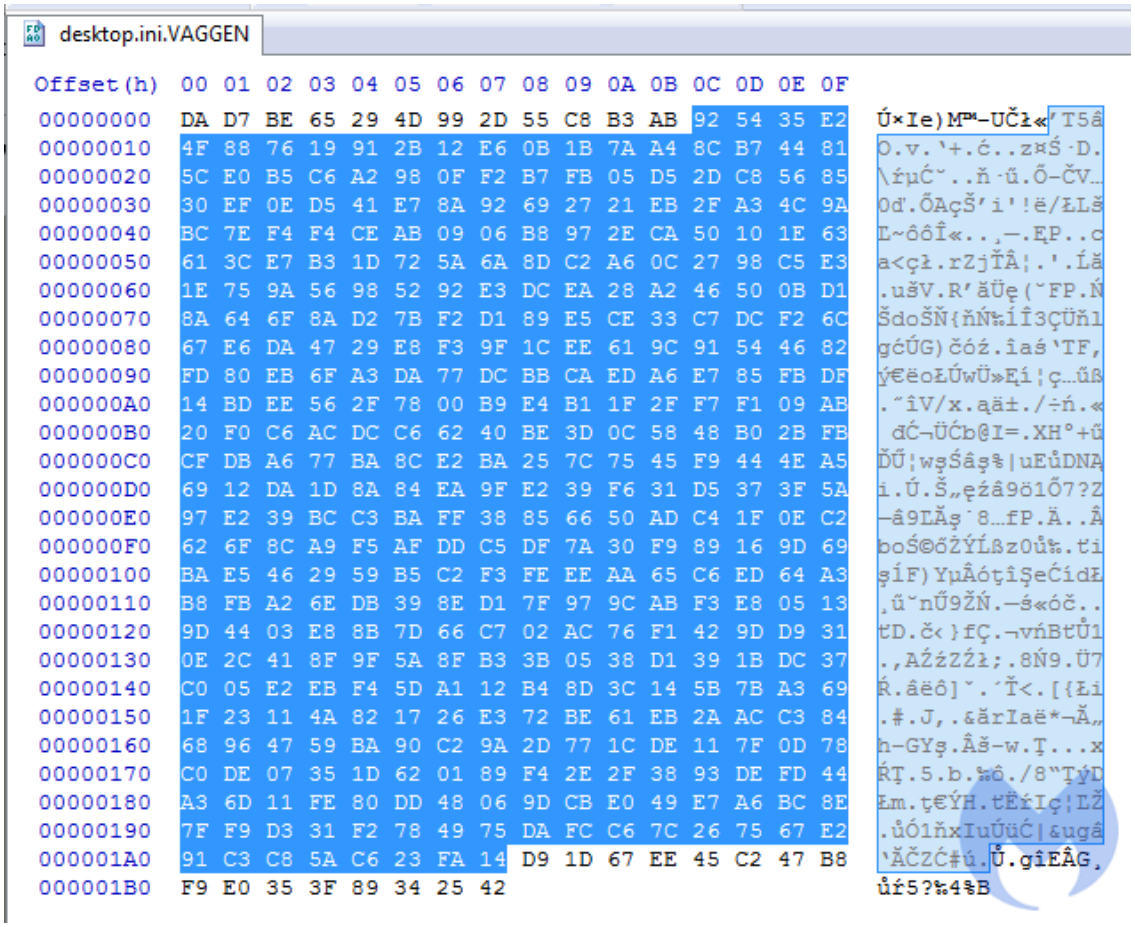
Powered by Malwarebytes™

Products

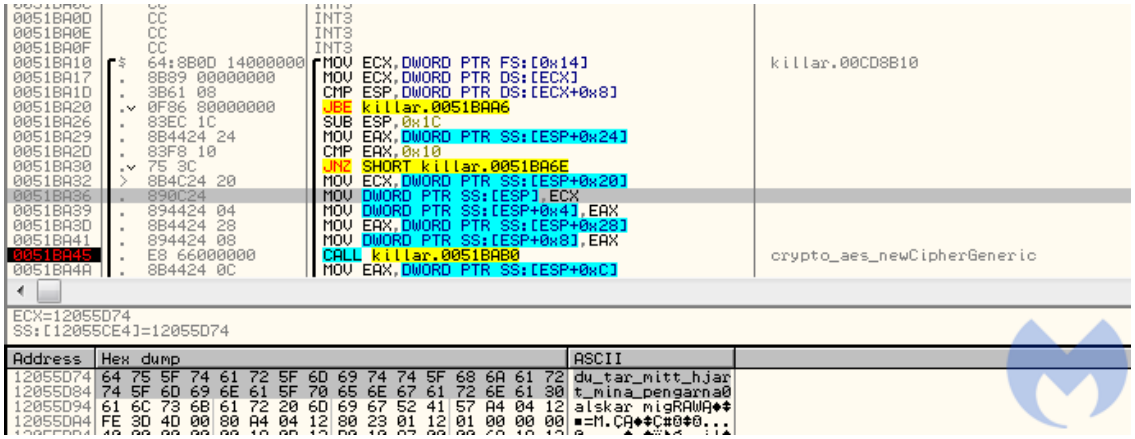
Partners

Resources

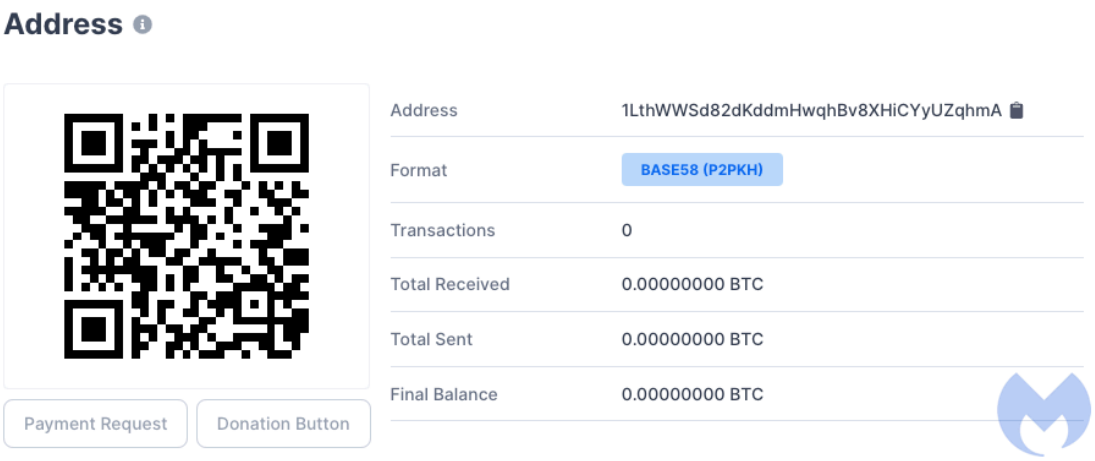
Why ThreatDown



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

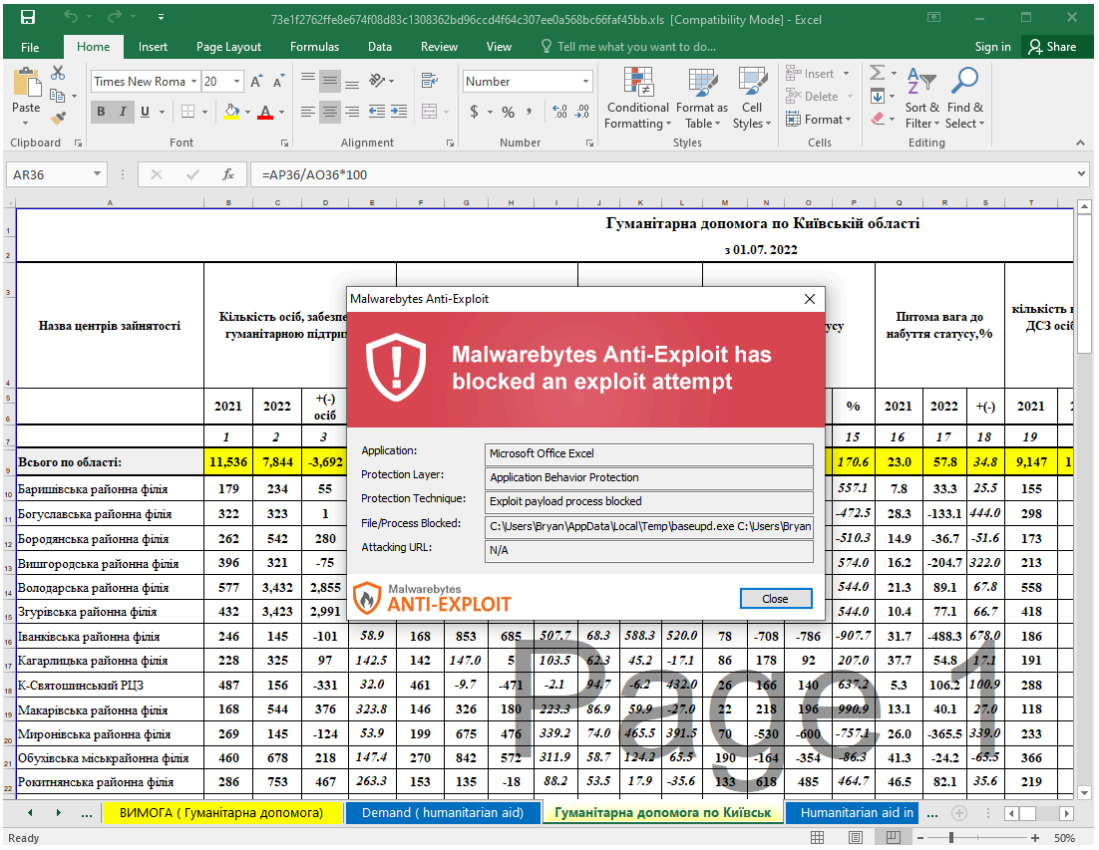
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

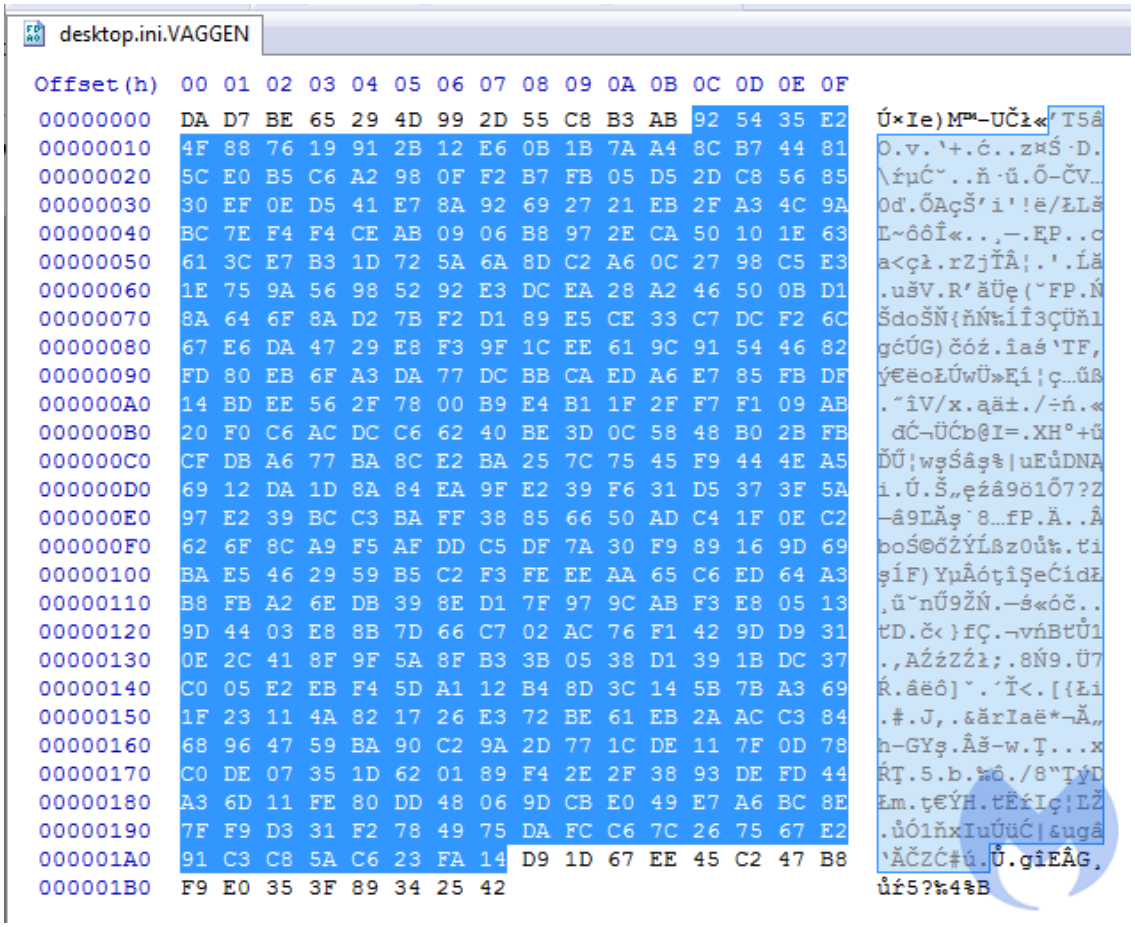
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

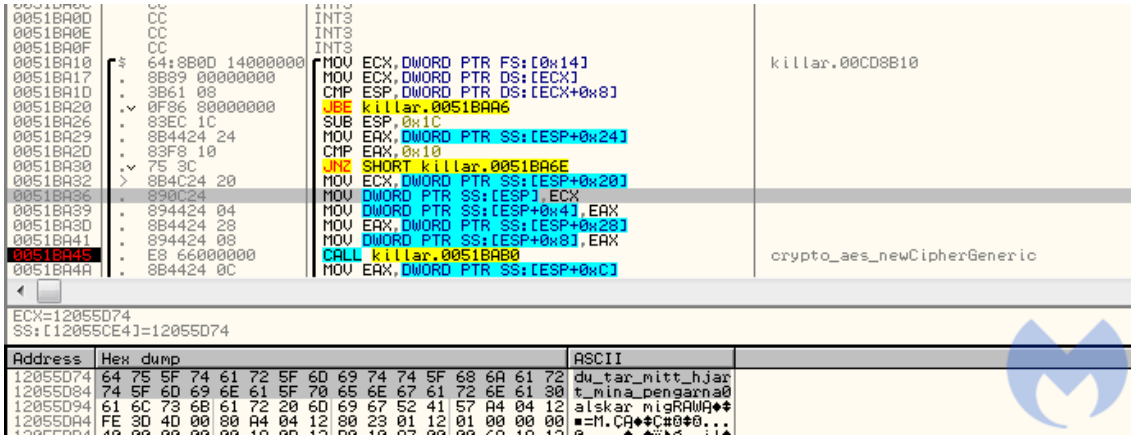


The content of the output file (with .VAGGEN extension) contains:

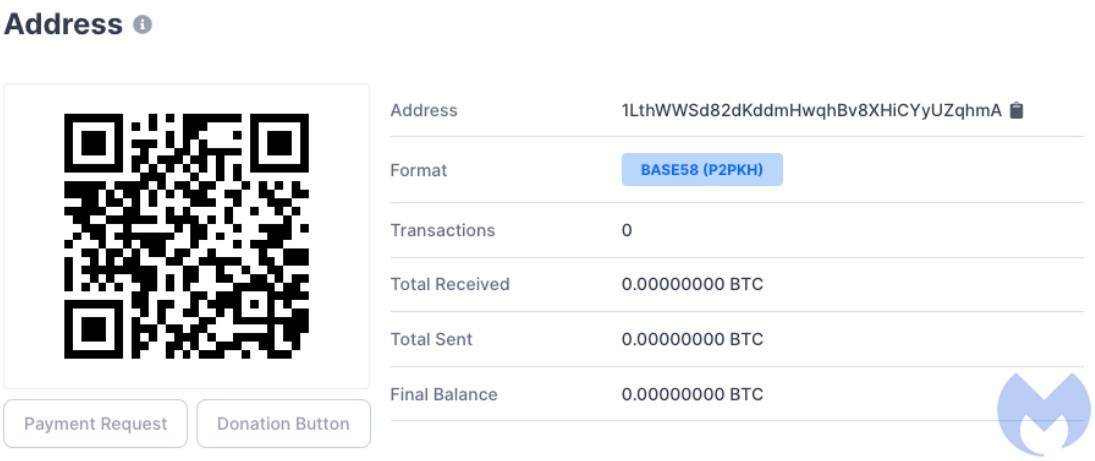
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

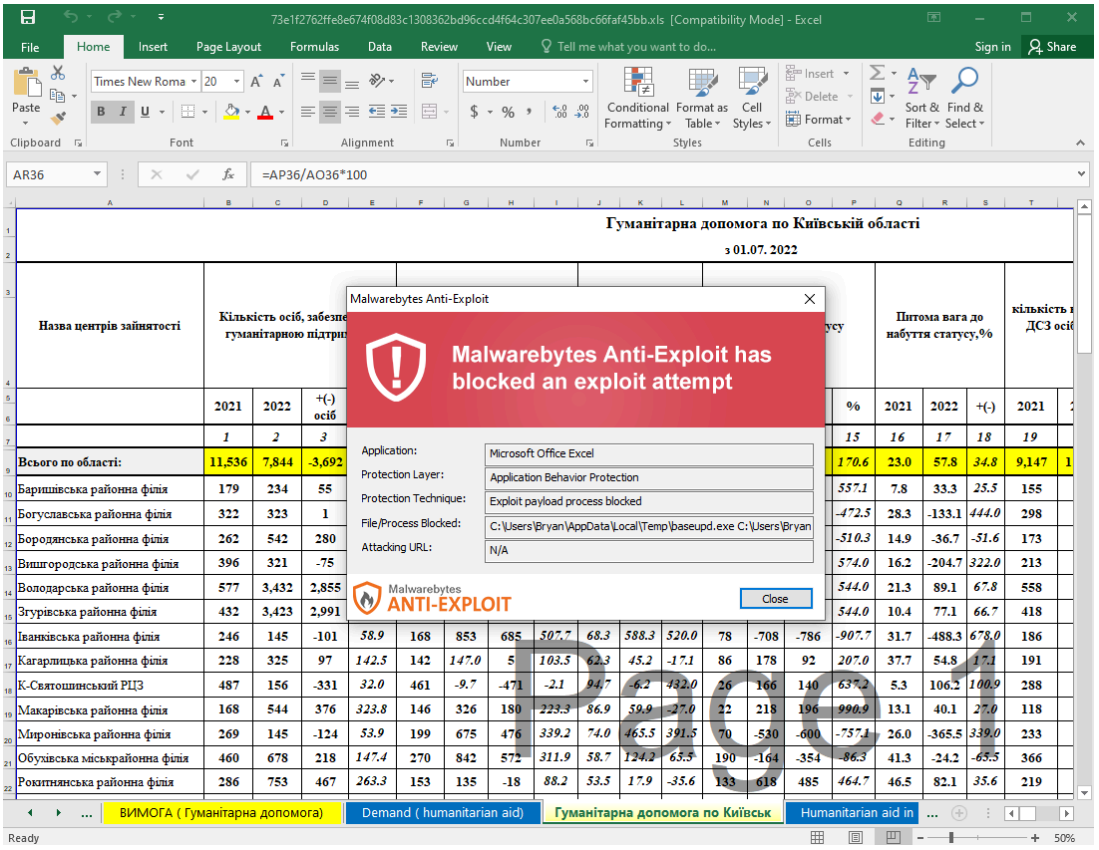
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
46  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
47    runtime_morestack_noctxt();
48  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
49  ((void (*)(void))loc_45CDA0)();
50  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
51  v38 = v4;
52  v30 = v5;
53  v31 = v6;
54  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
55  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
56  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
57  v32 = v10;
58  v11 = *(void (__cdecl **)(int))(v10 + 16);
59  v39 = v12;
60  v11(v12);
61  v29 = v13;
62  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
63  v40 = v15;
64  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
65  v17 = v27;
66  if ( v27 )
67  {
68    if ( v27 )
69      v17 = *(_DWORD *)(v27 + 4);
70    v43 = v17;
71    v44 = v28;
72    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
73  }
```

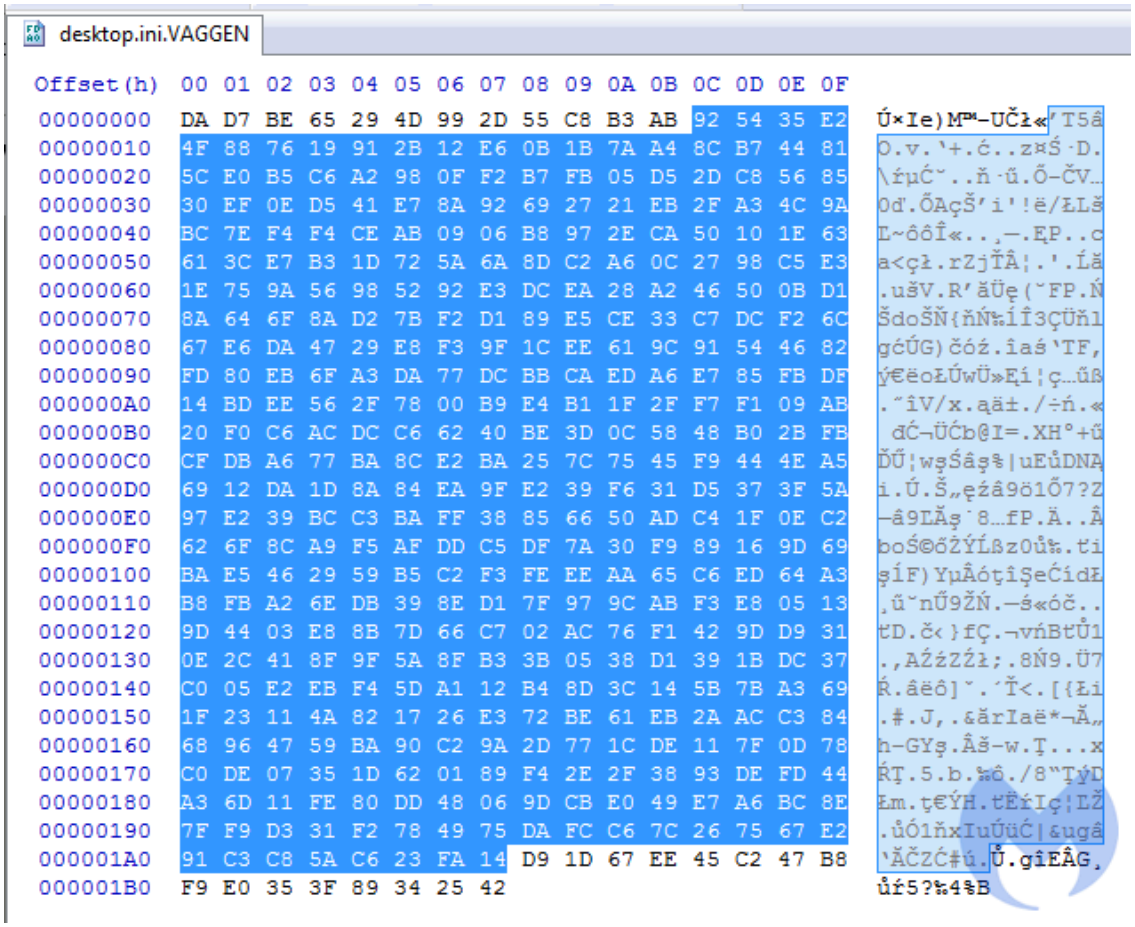


The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

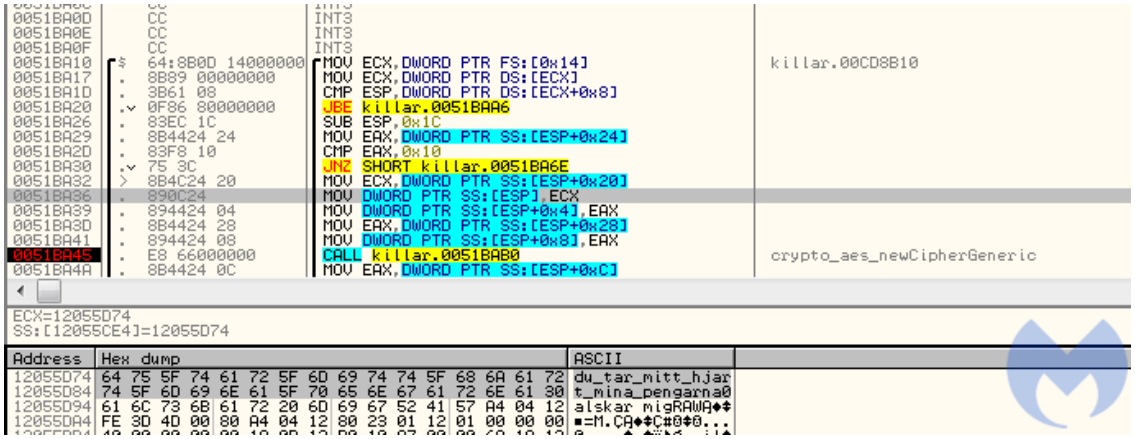


The content of the output file (with .VAGGEN extension) contains:

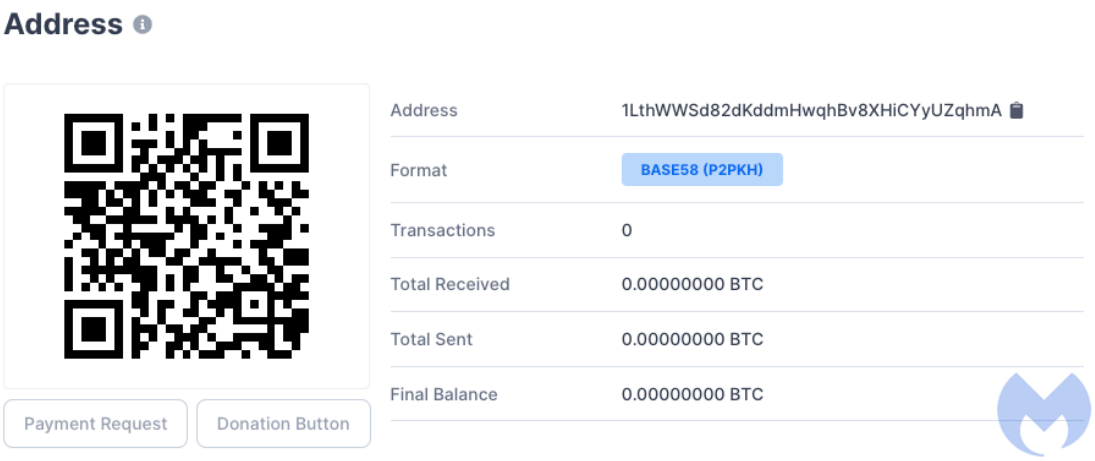
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

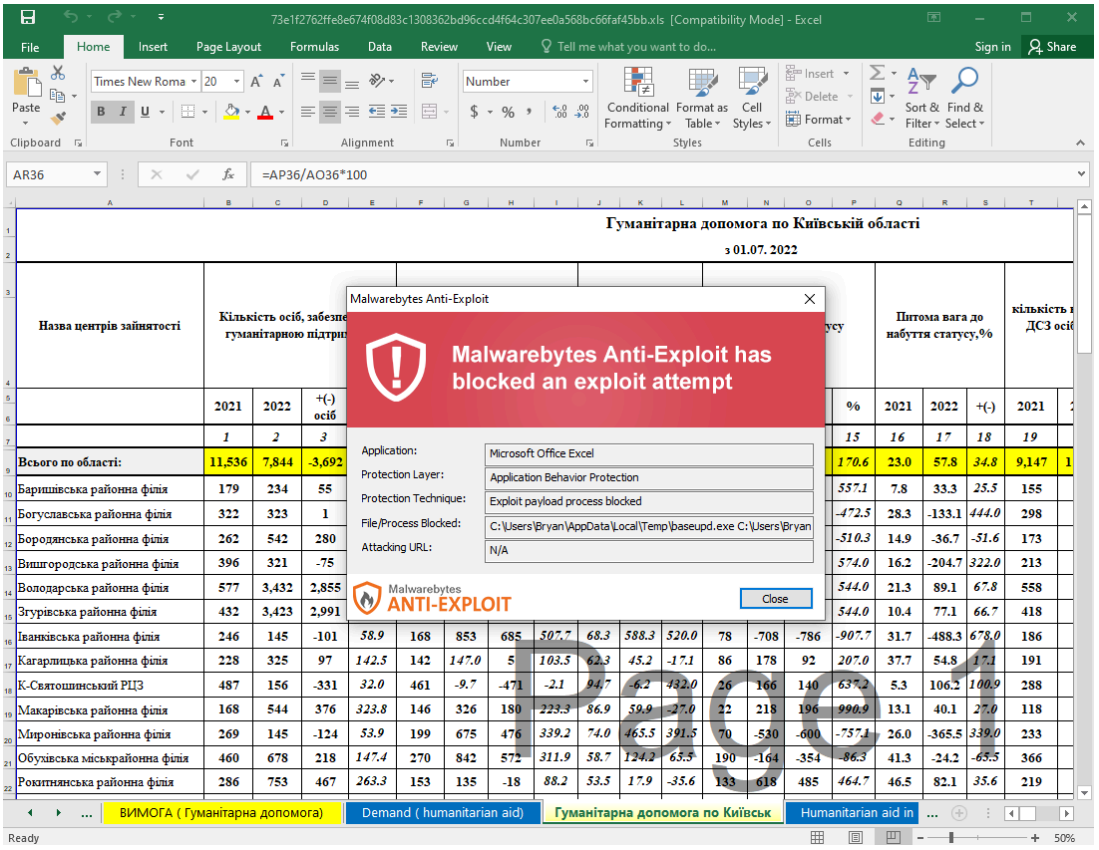
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

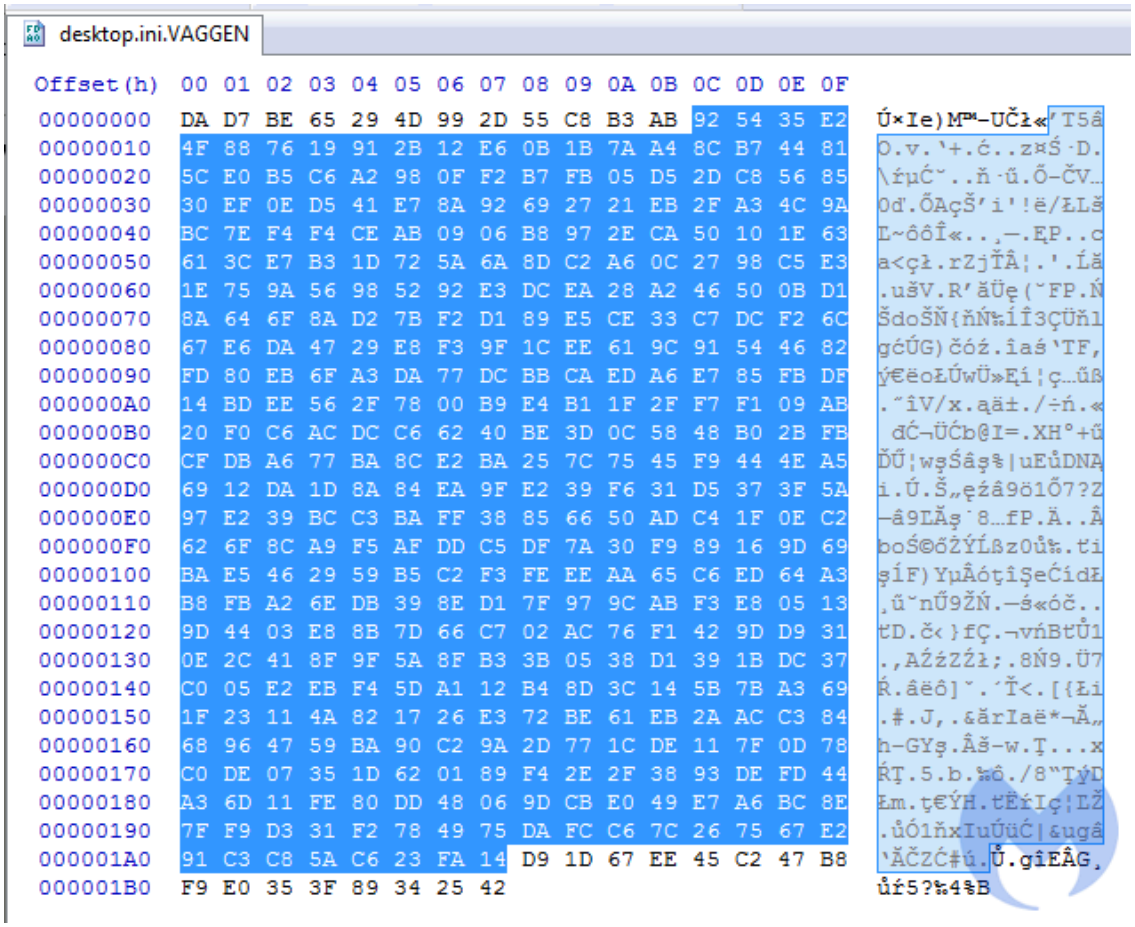
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

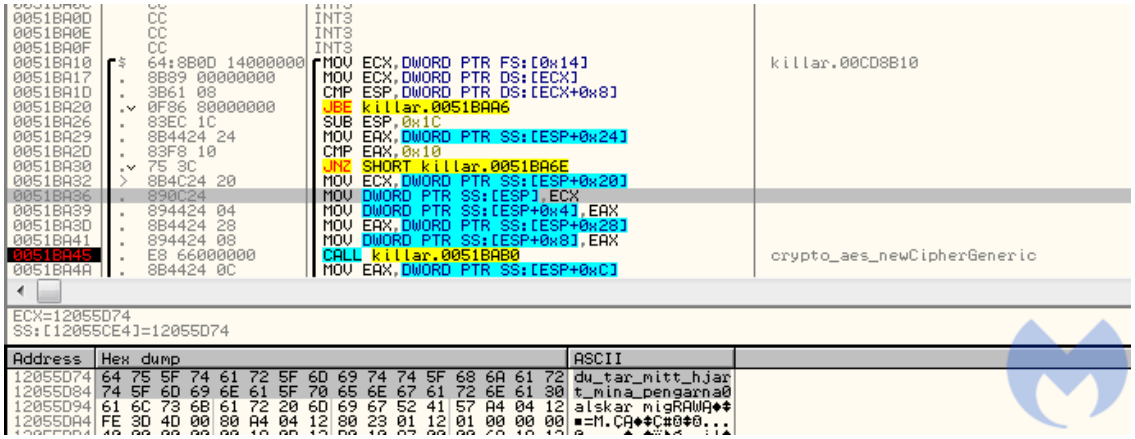


The content of the output file (with .VAGGEN extension) contains:

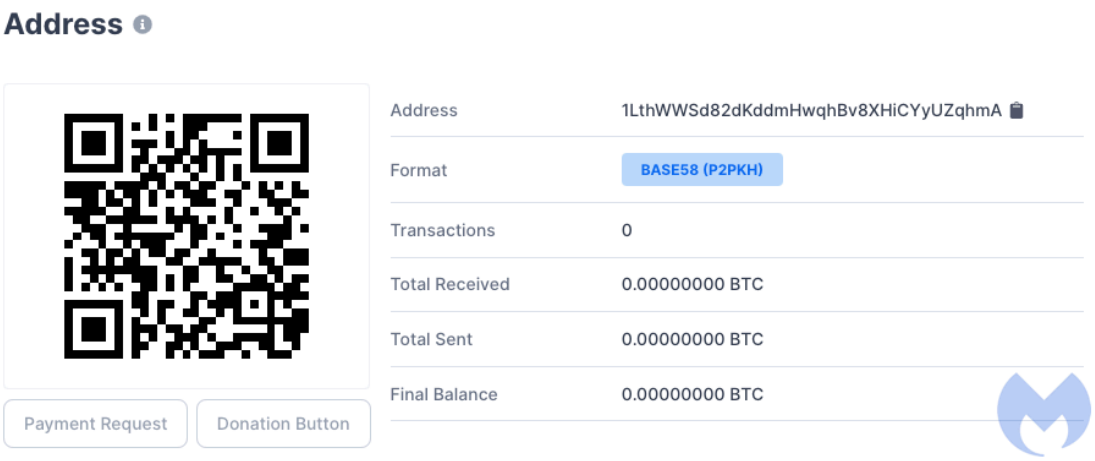
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

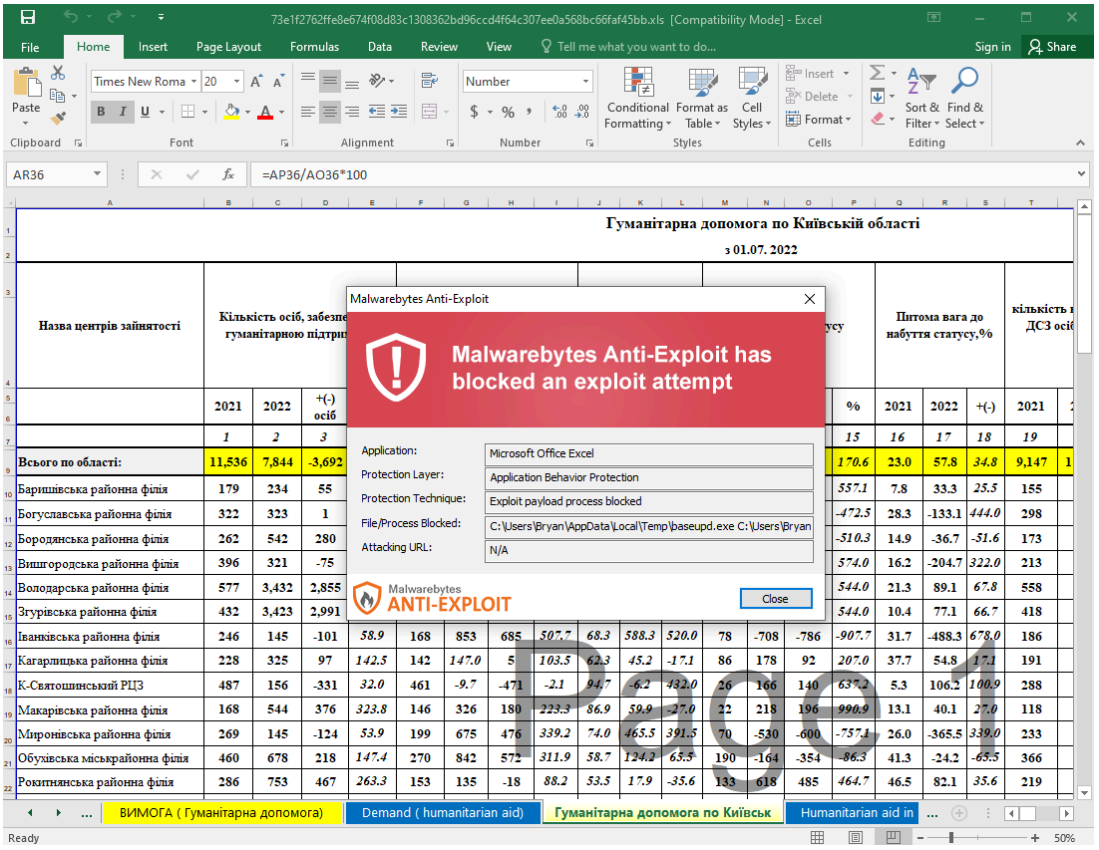
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

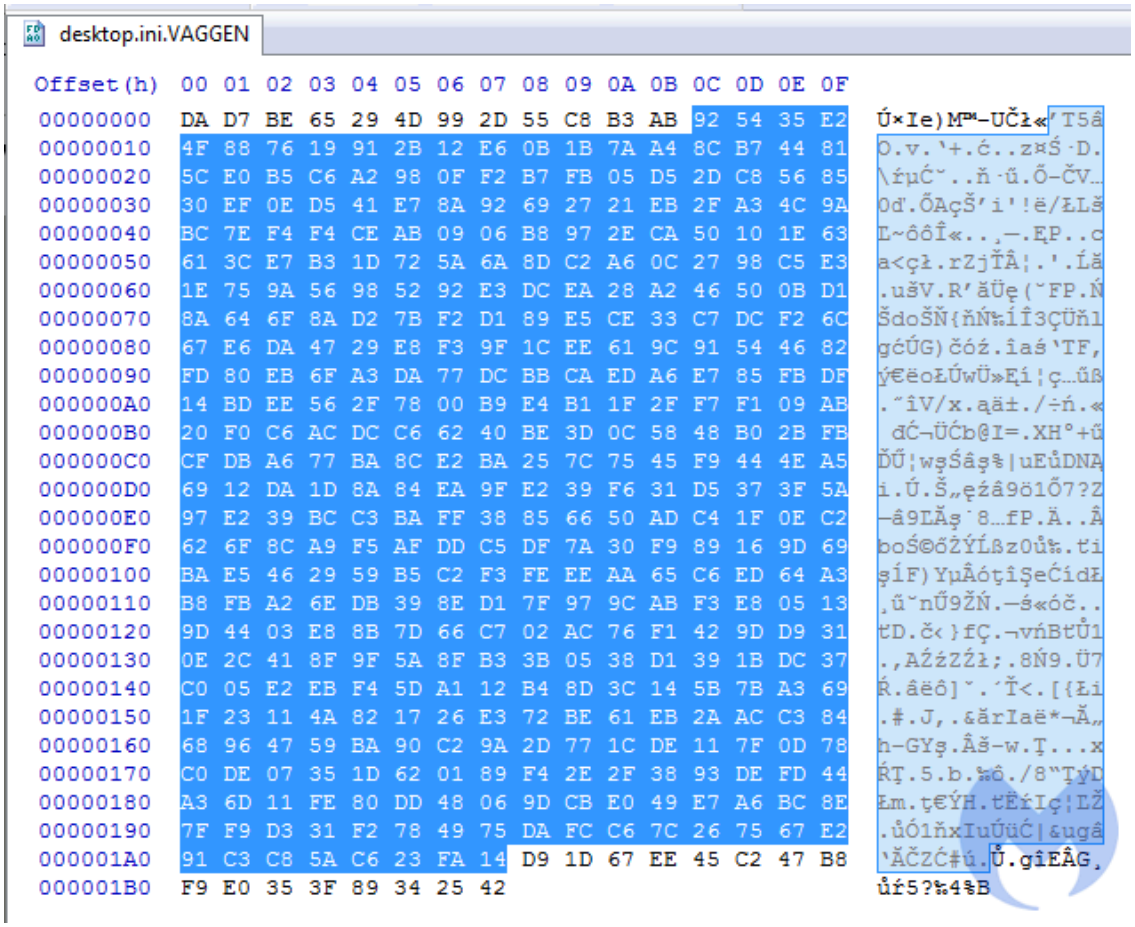
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

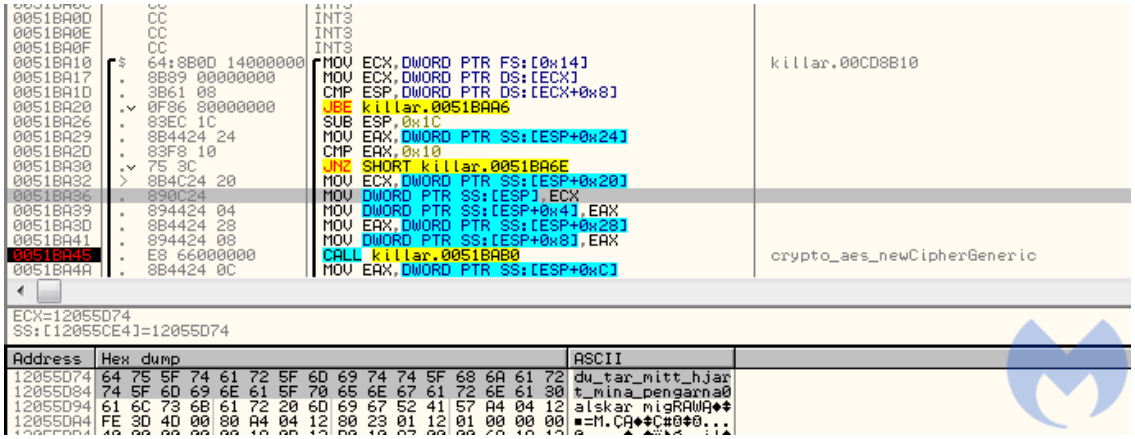


The content of the output file (with .VAGGEN extension) contains:

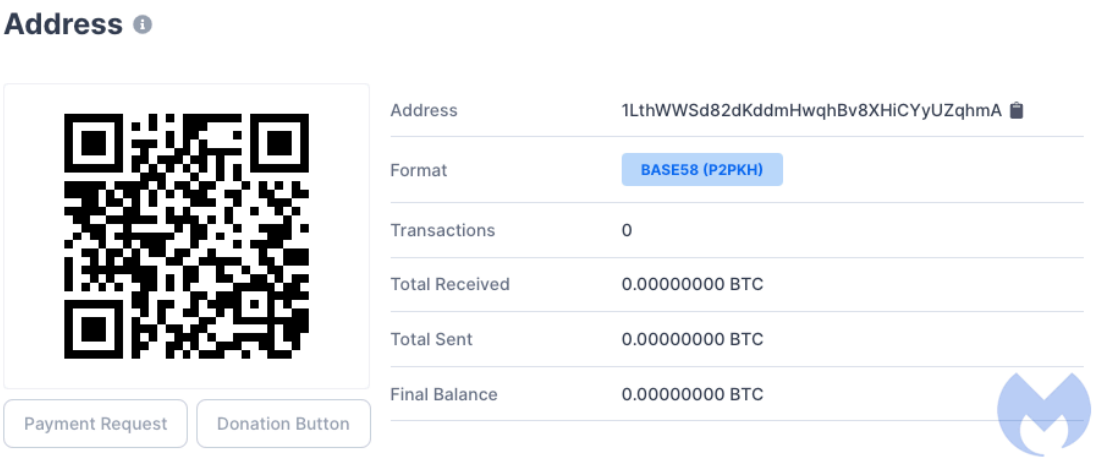
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

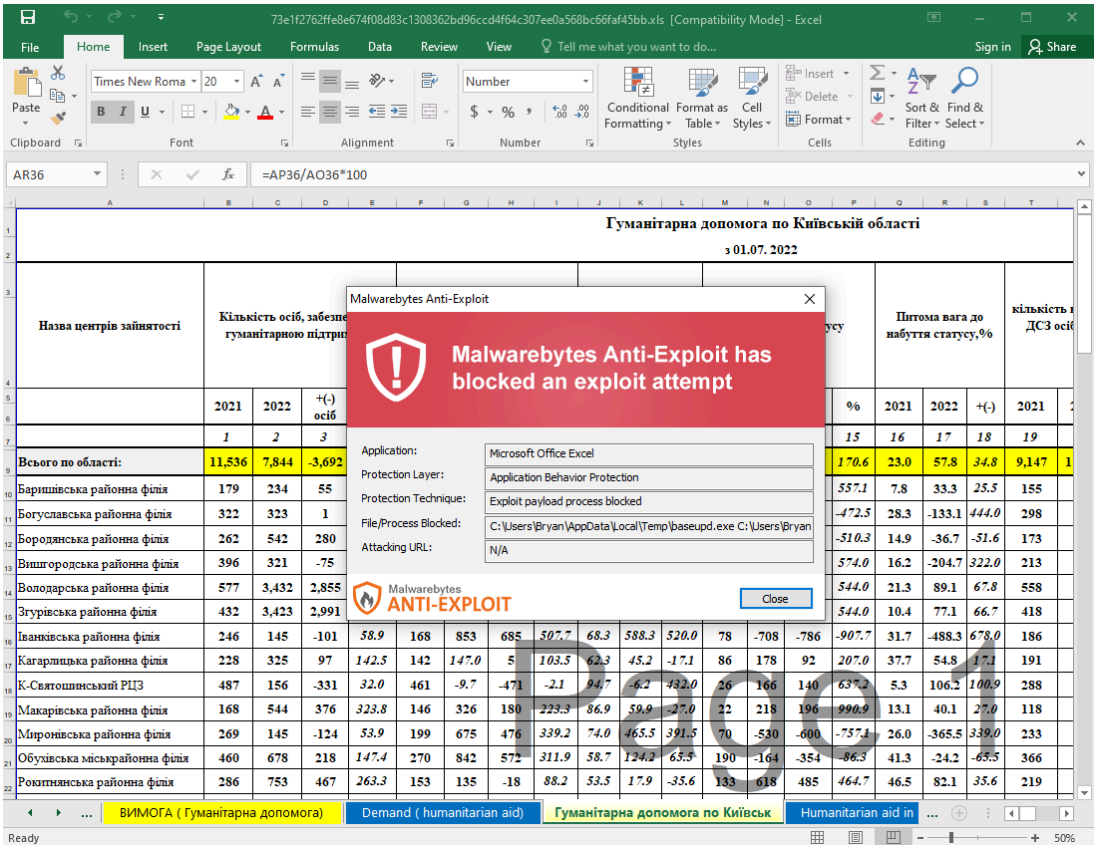
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

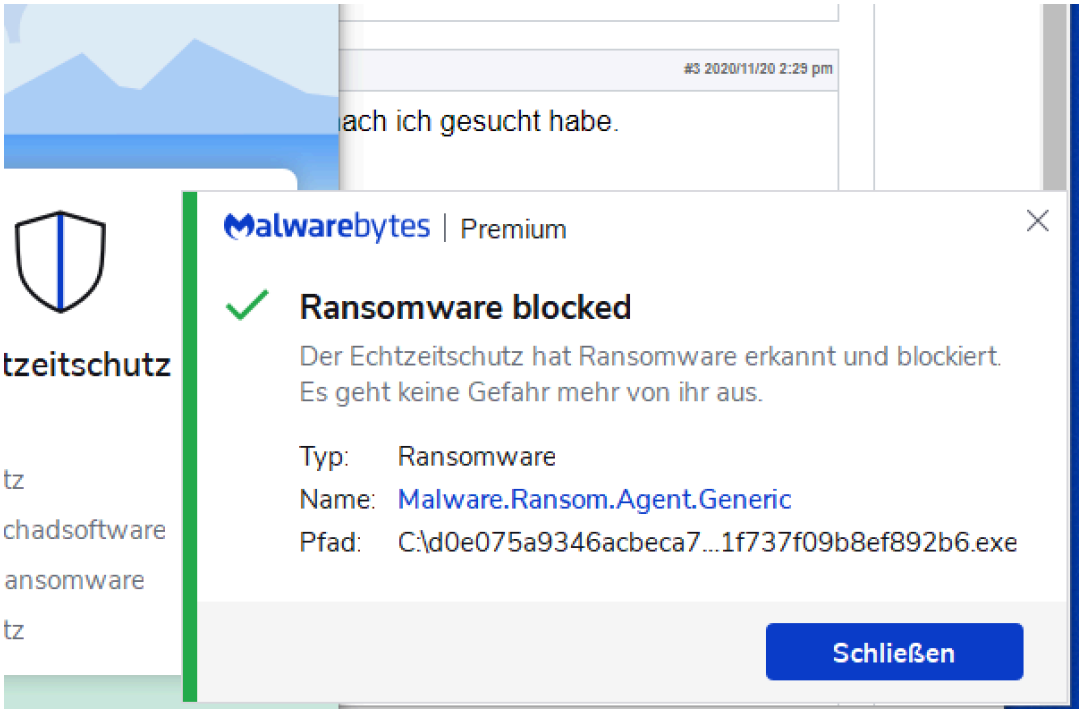
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

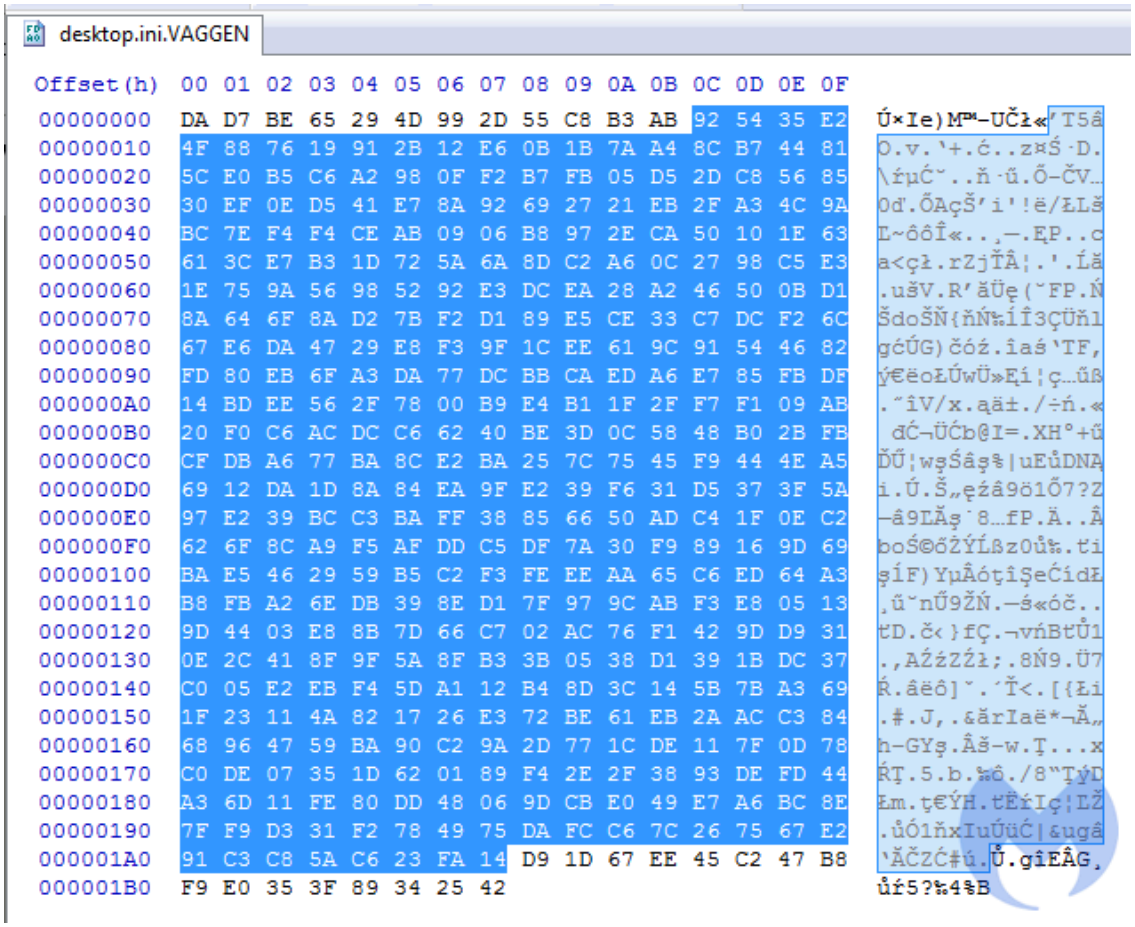
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

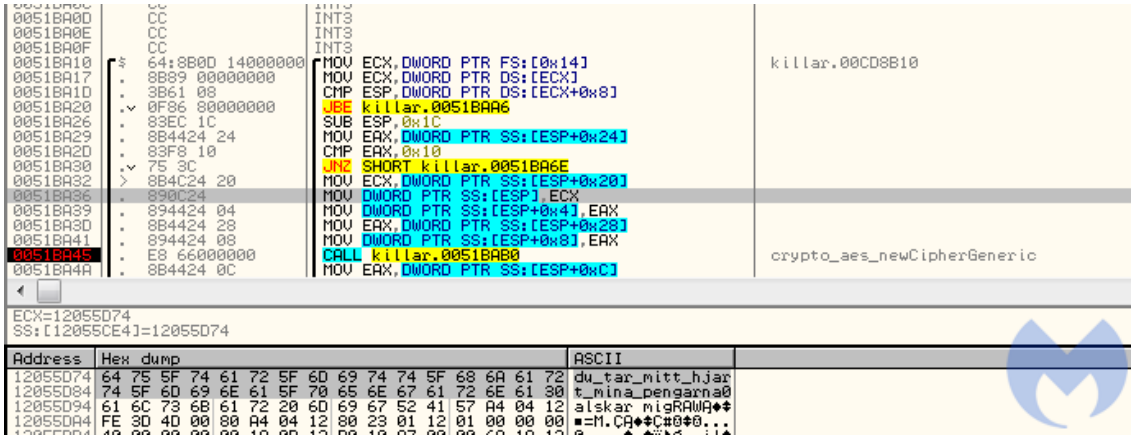


The content of the output file (with .VAGGEN extension) contains:

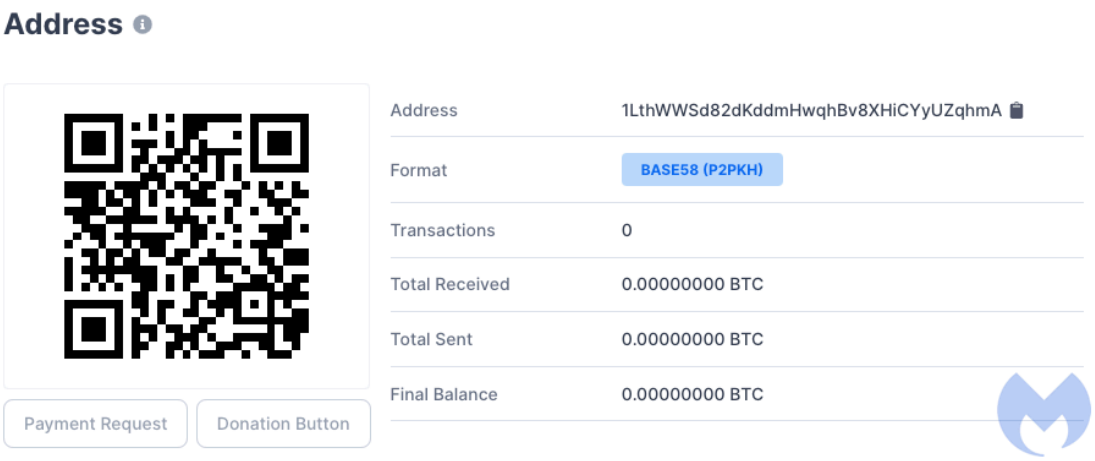
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

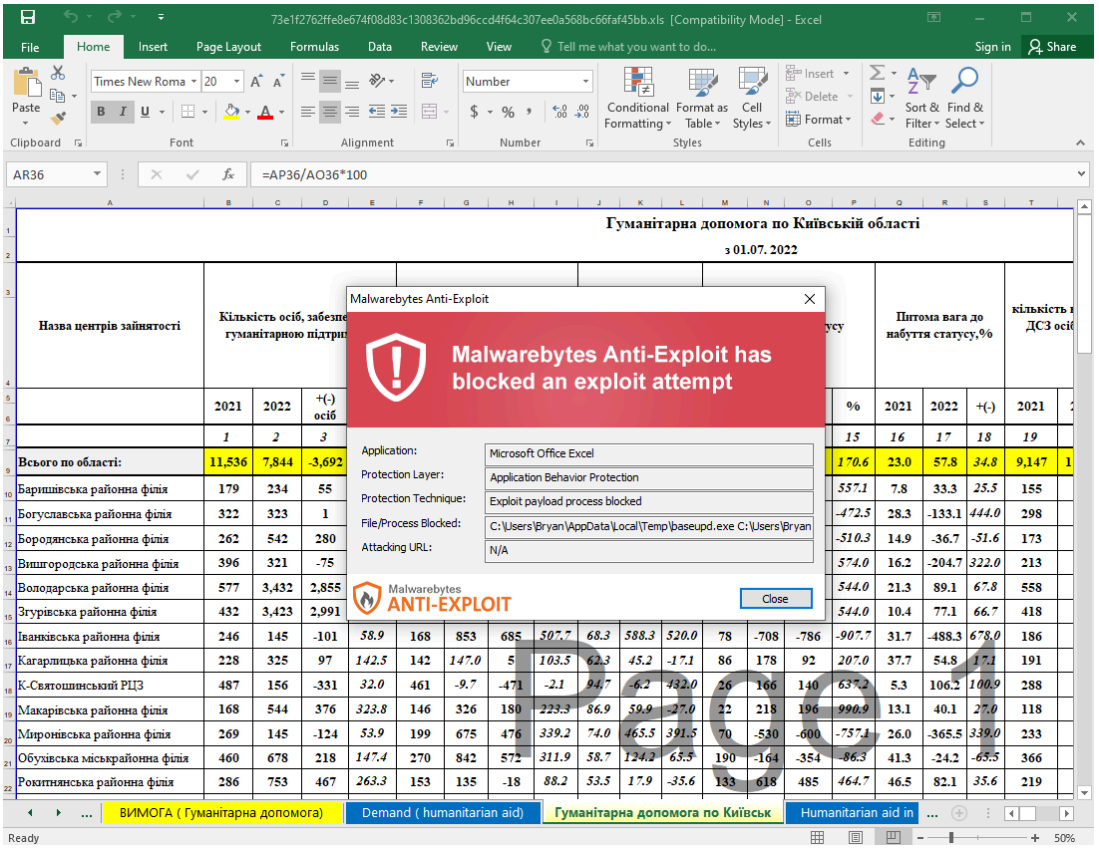
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

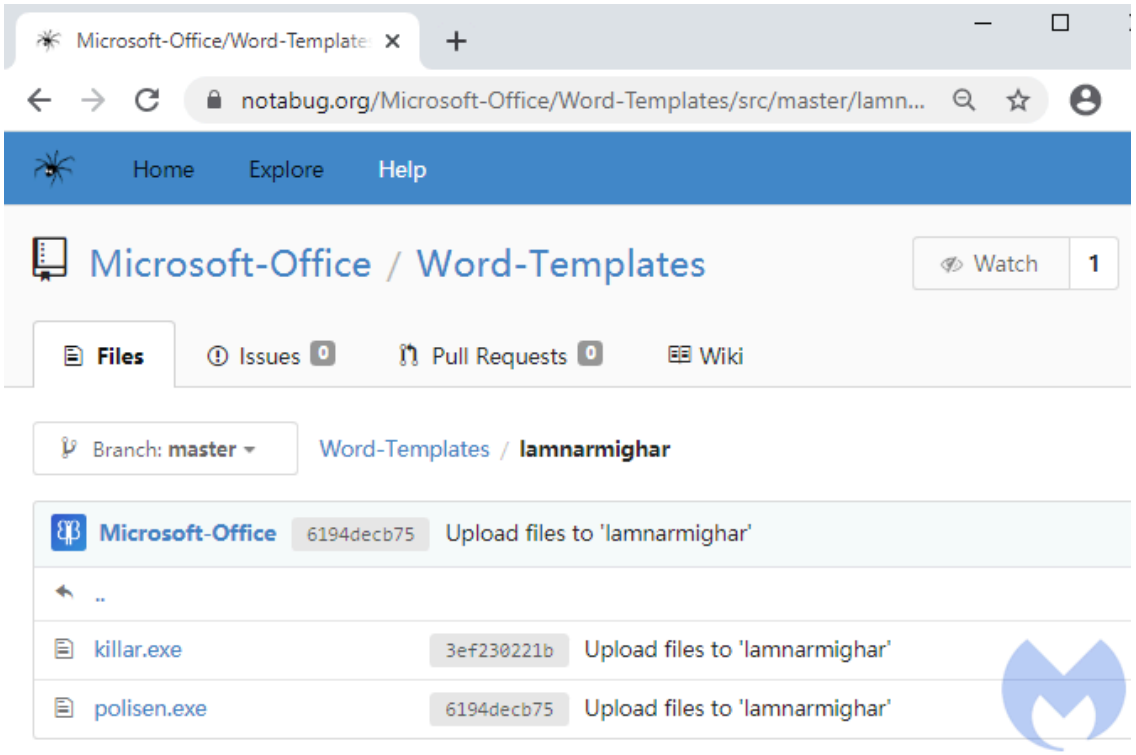
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



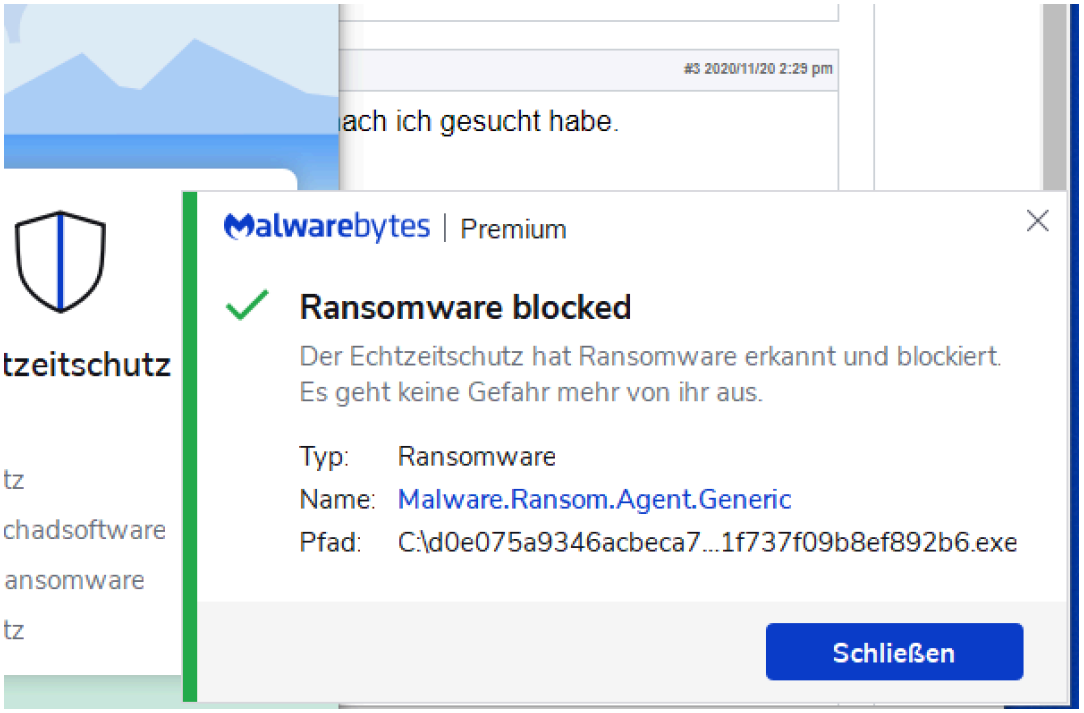
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

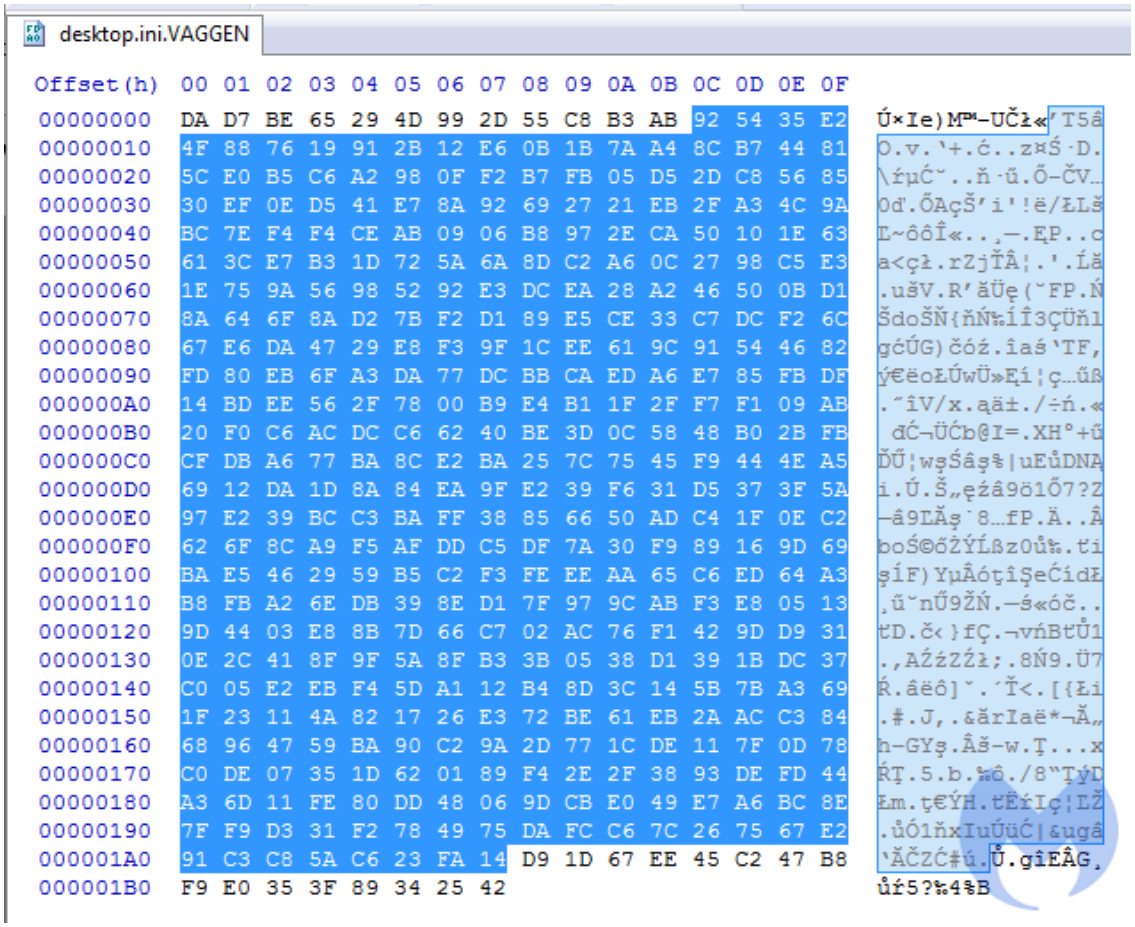
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



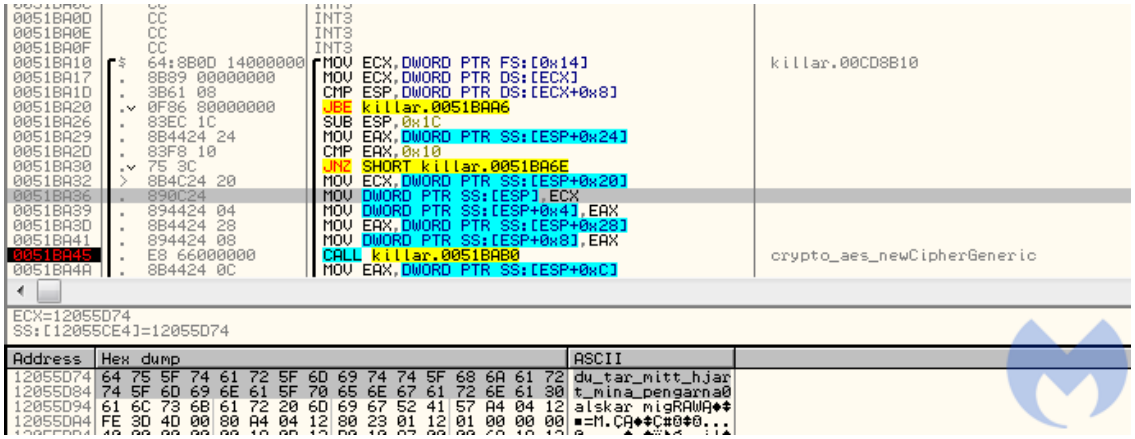
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

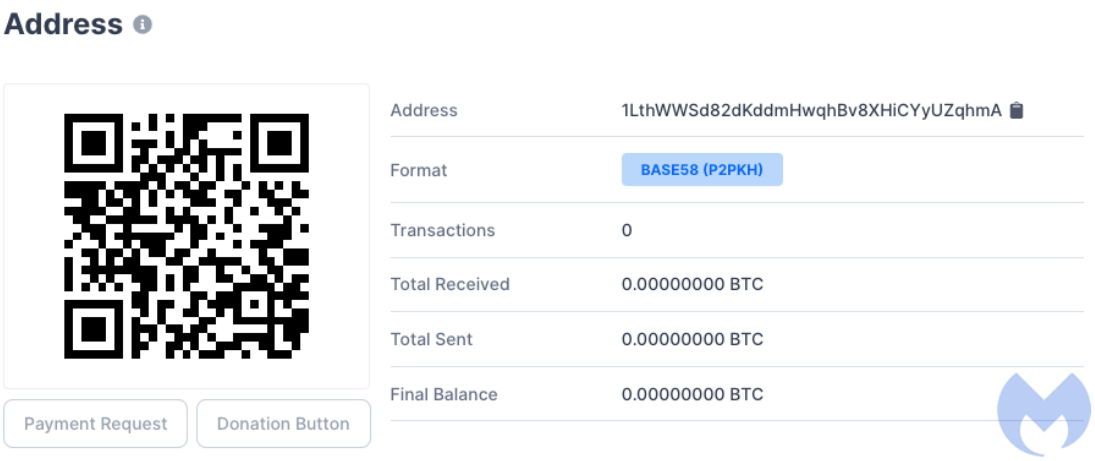
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

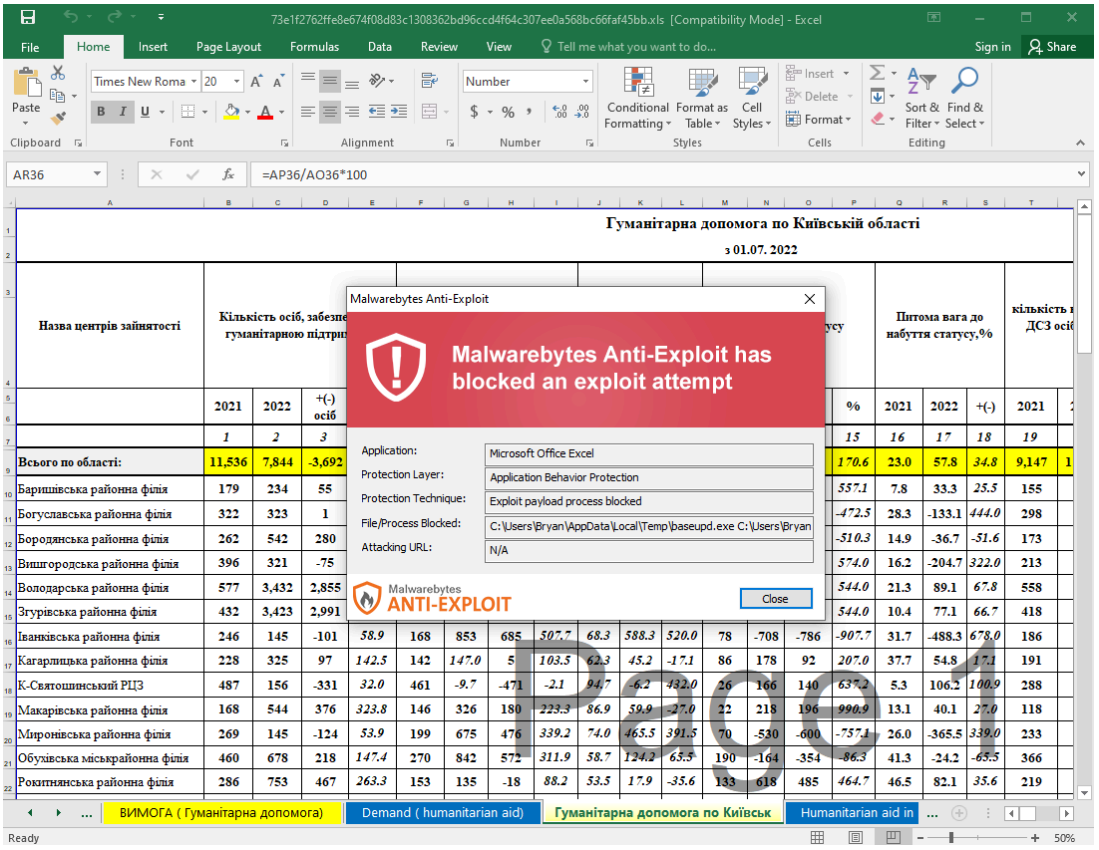
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

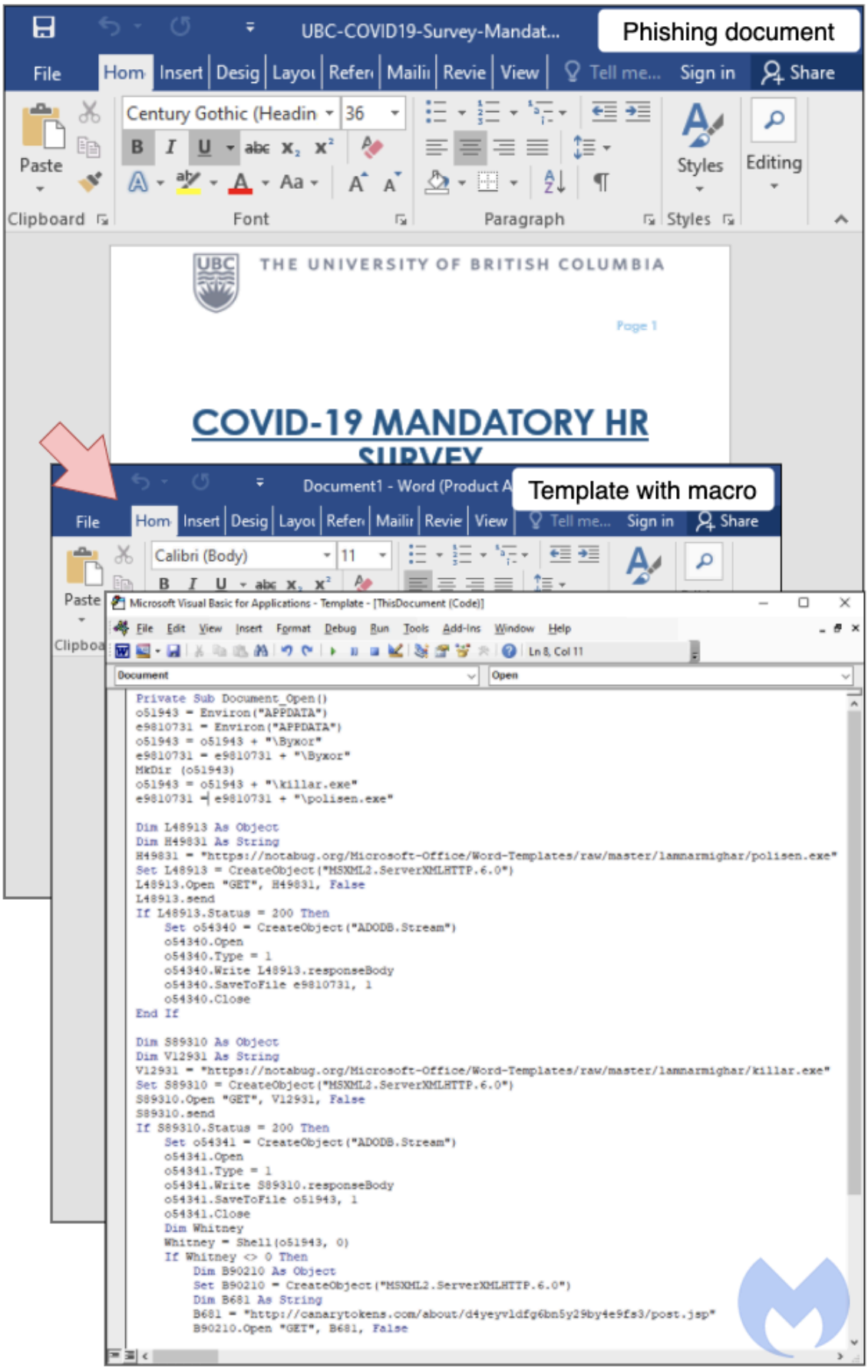
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

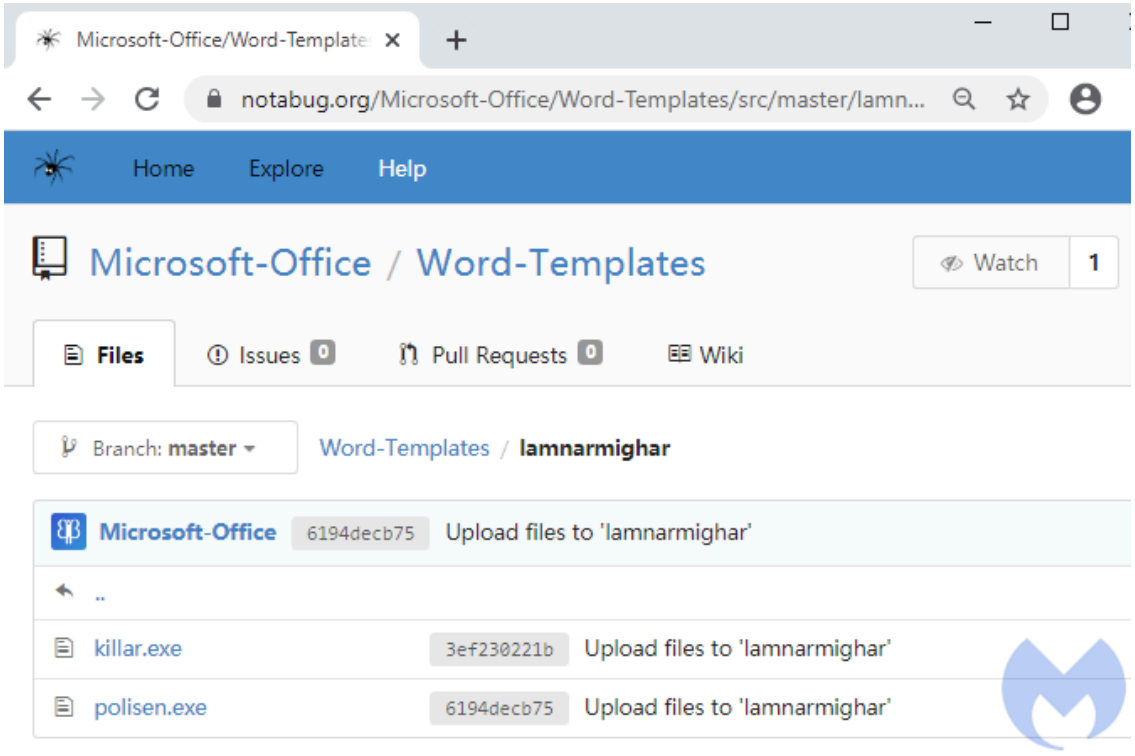
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



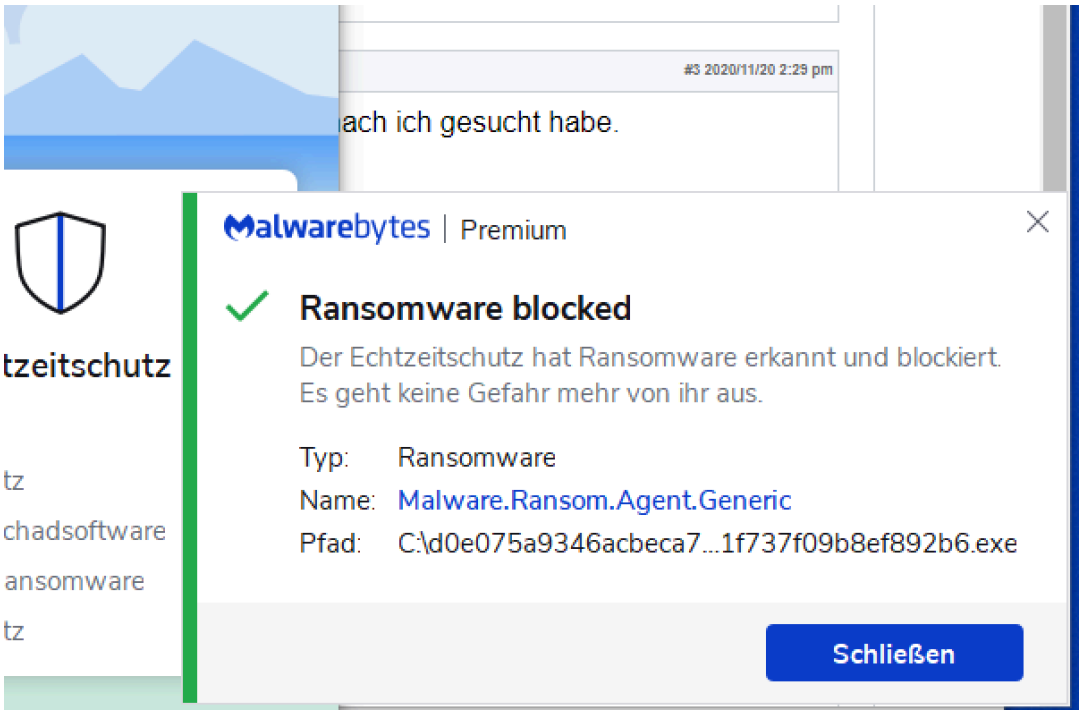
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

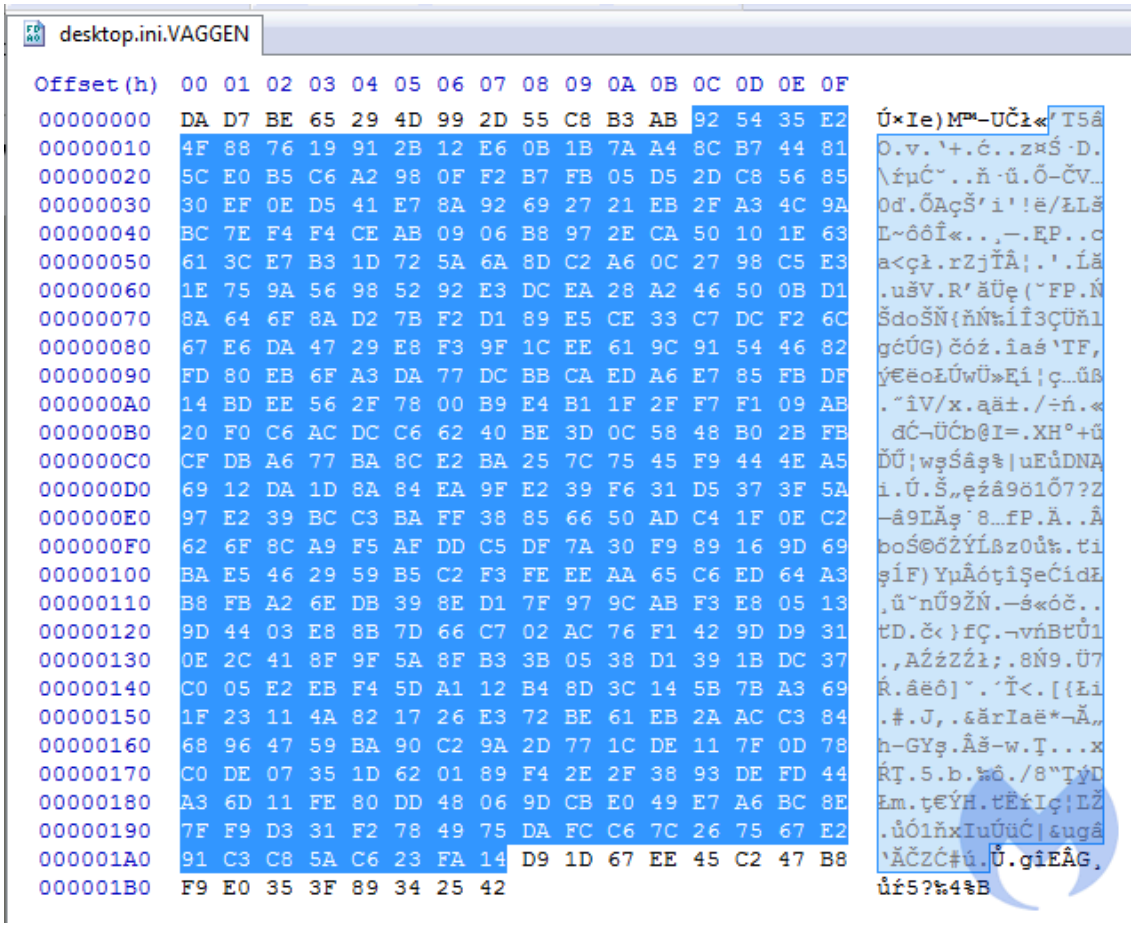
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

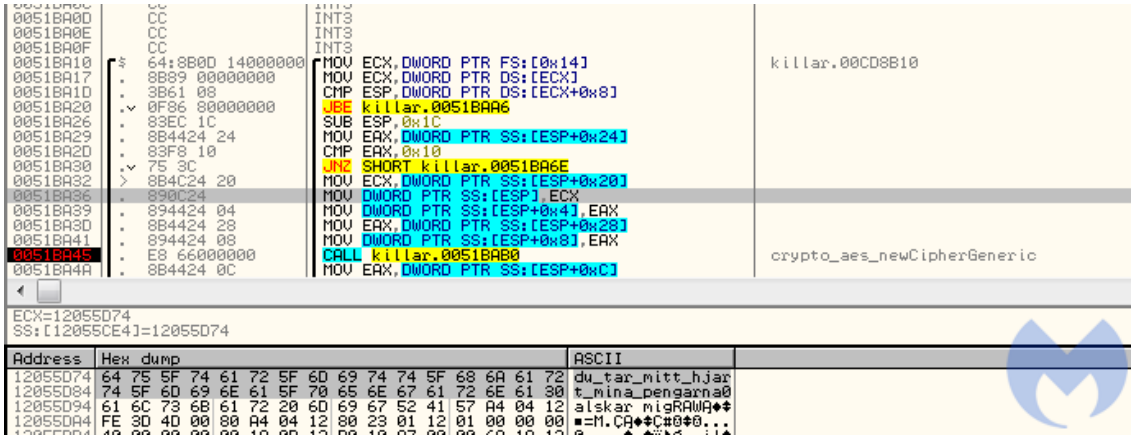


The content of the output file (with .VAGGEN extension) contains:

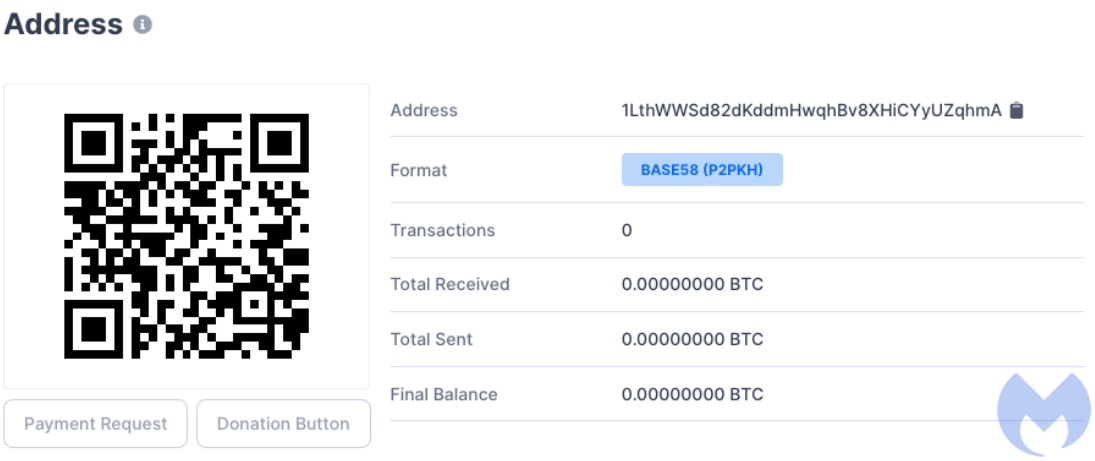
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

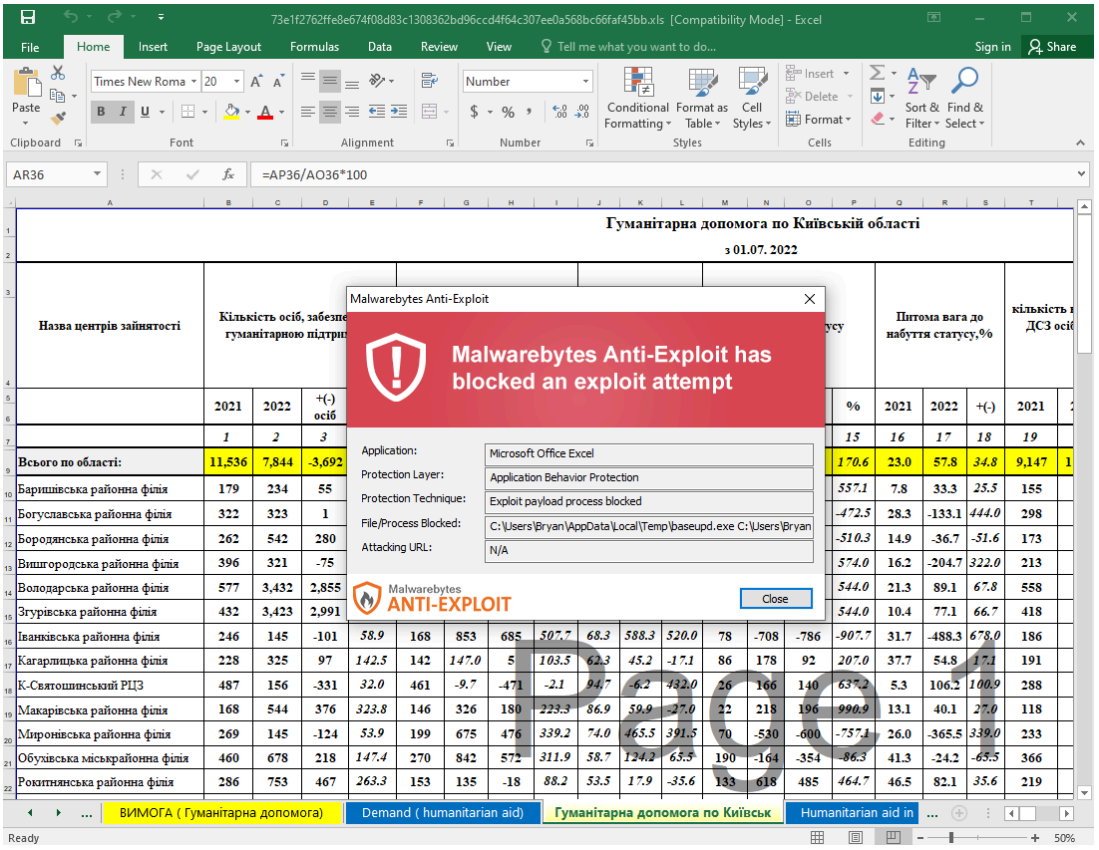
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

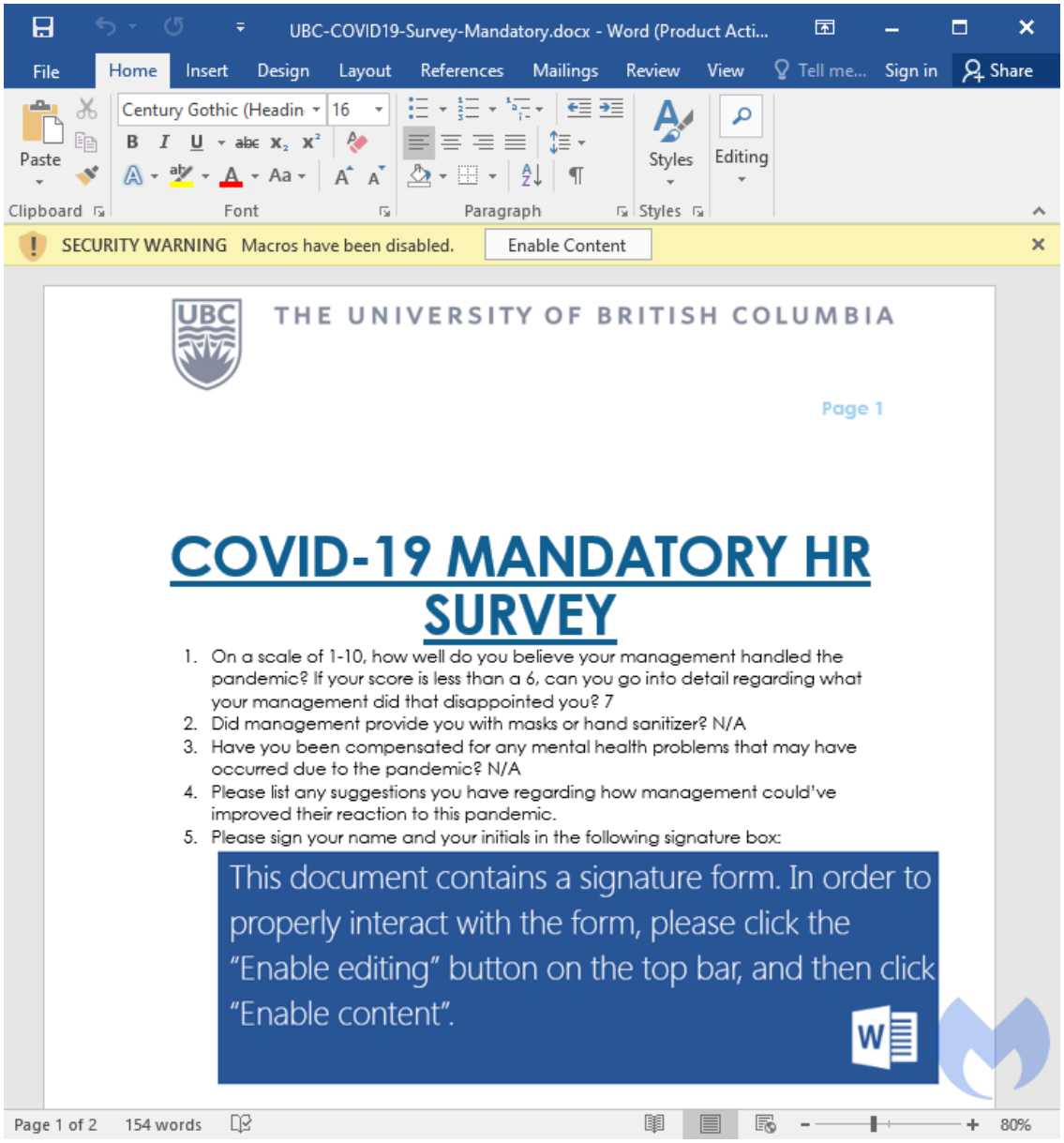
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

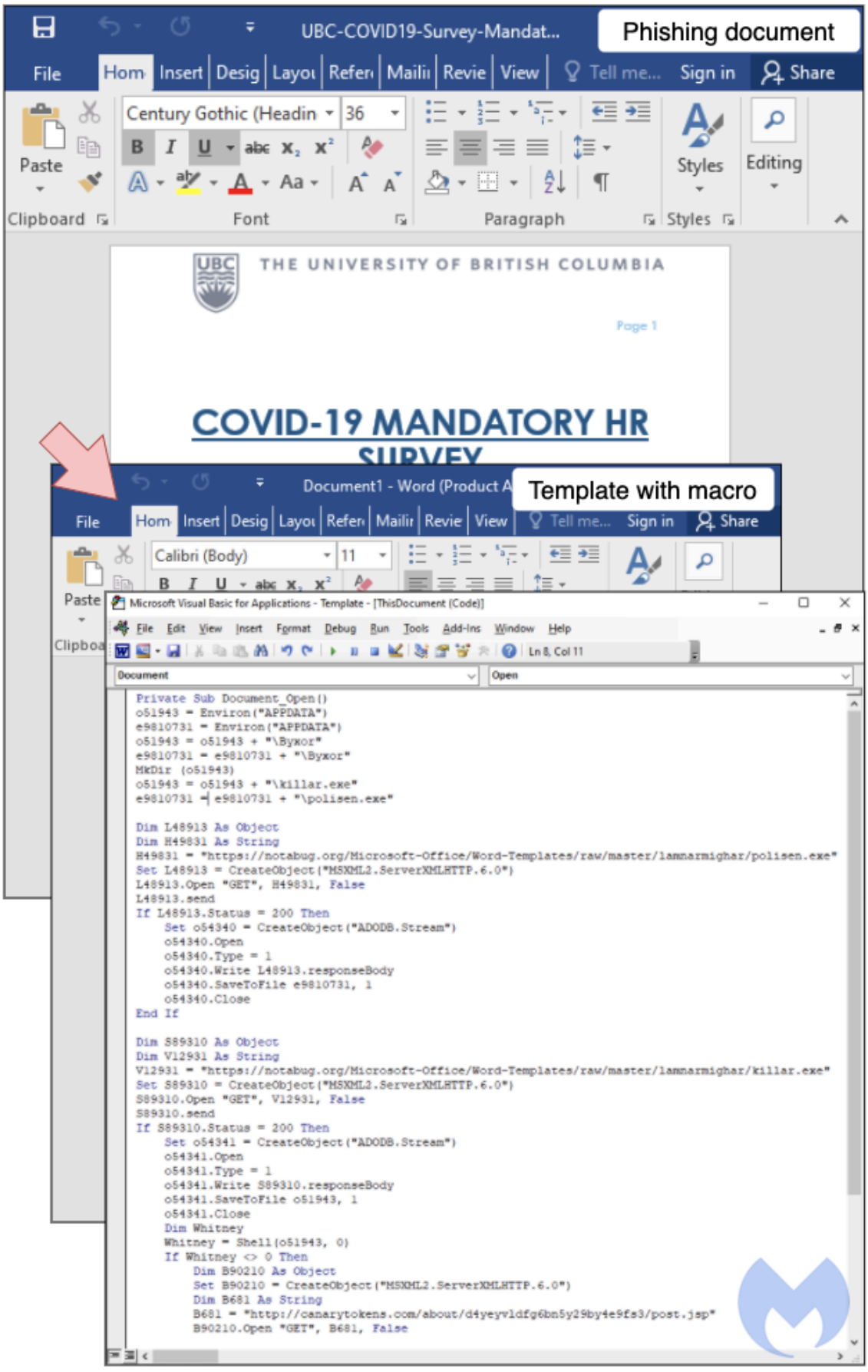
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

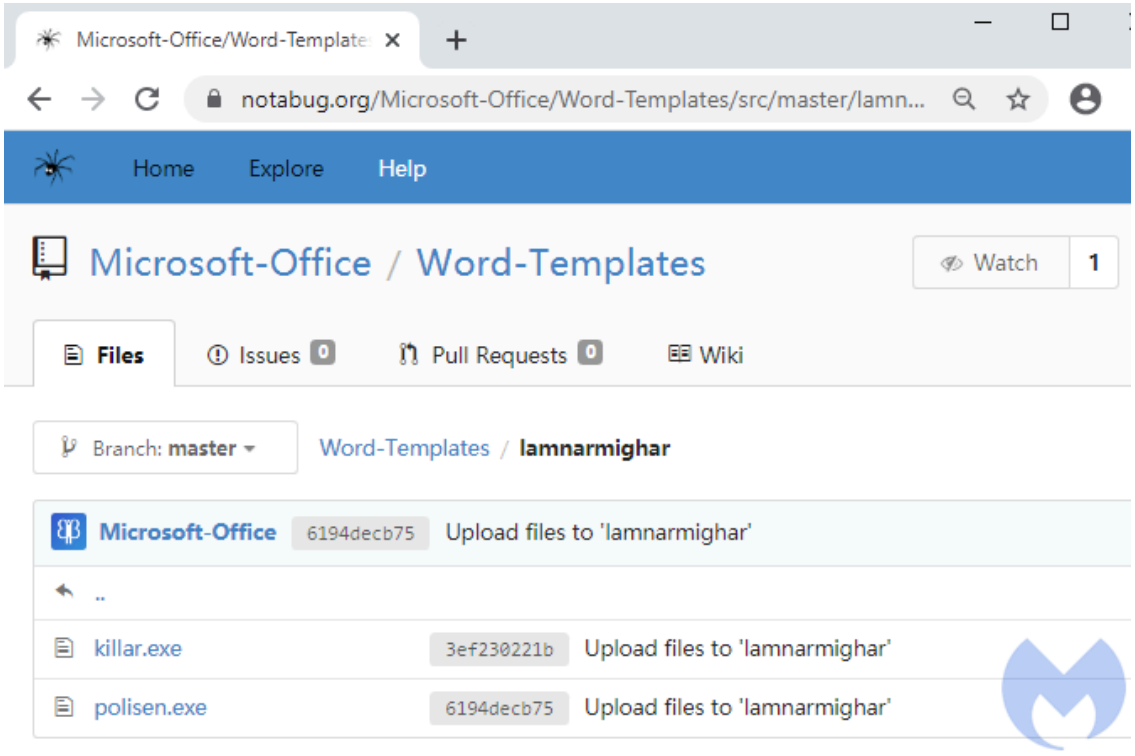
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



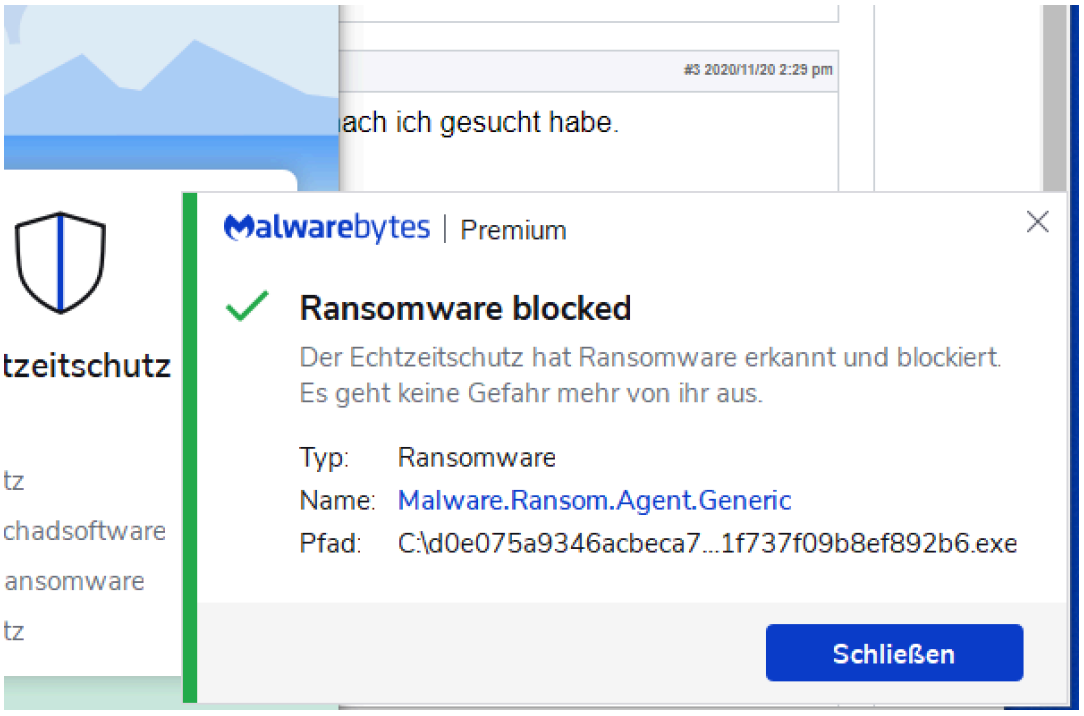
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

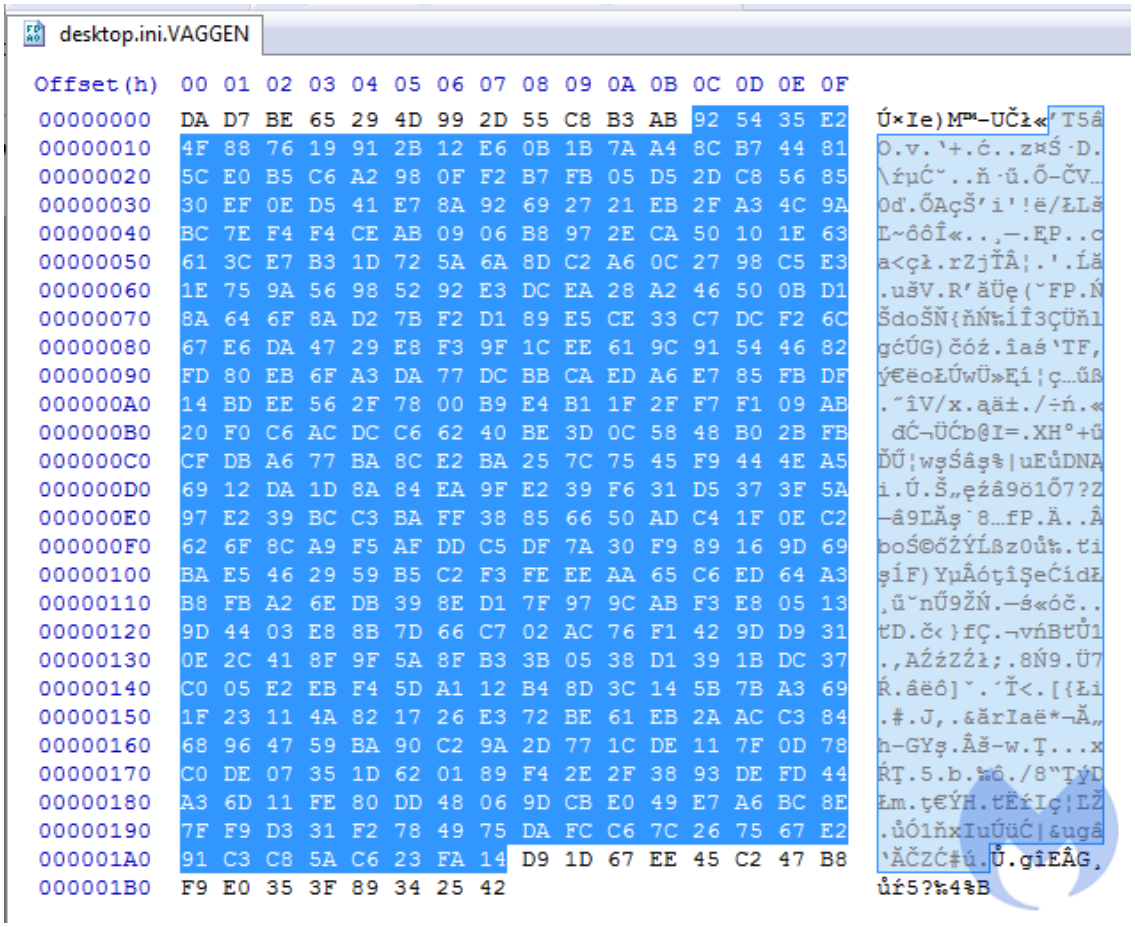
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

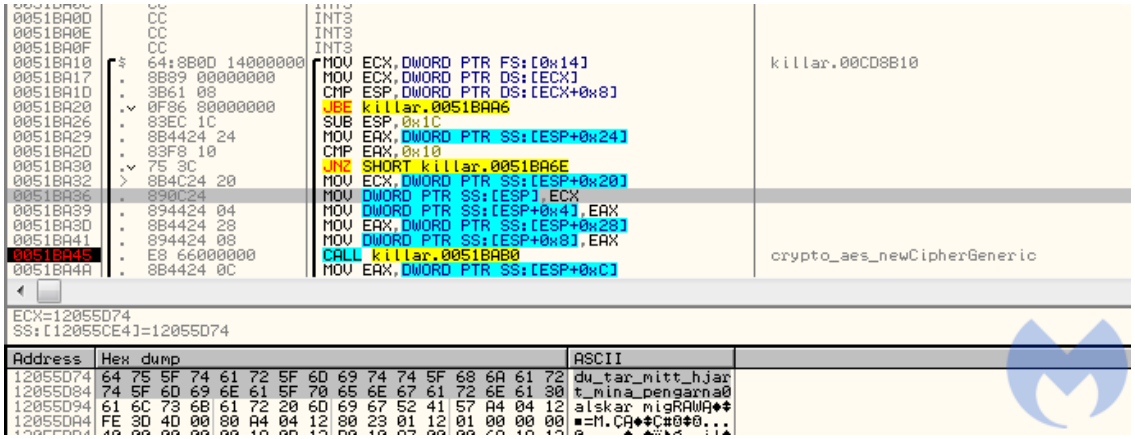


The content of the output file (with .VAGGEN extension) contains:

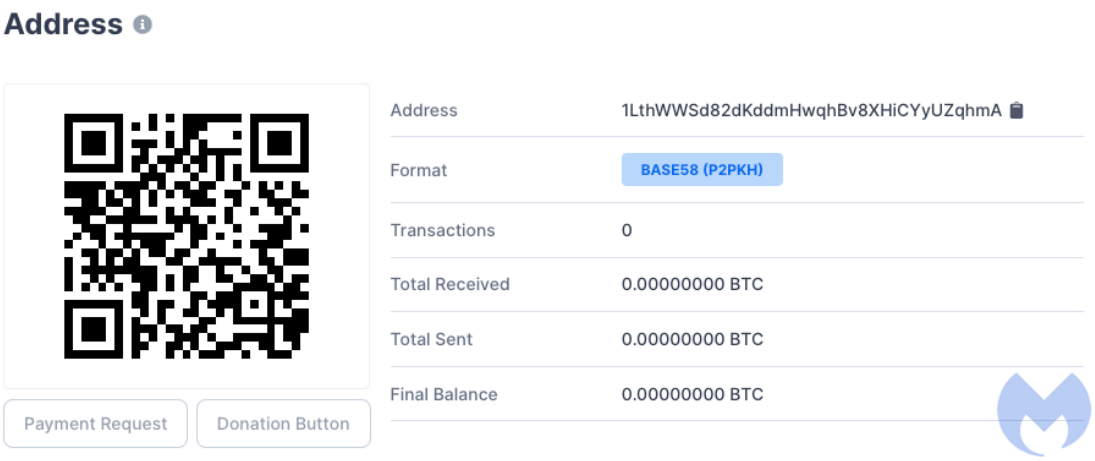
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

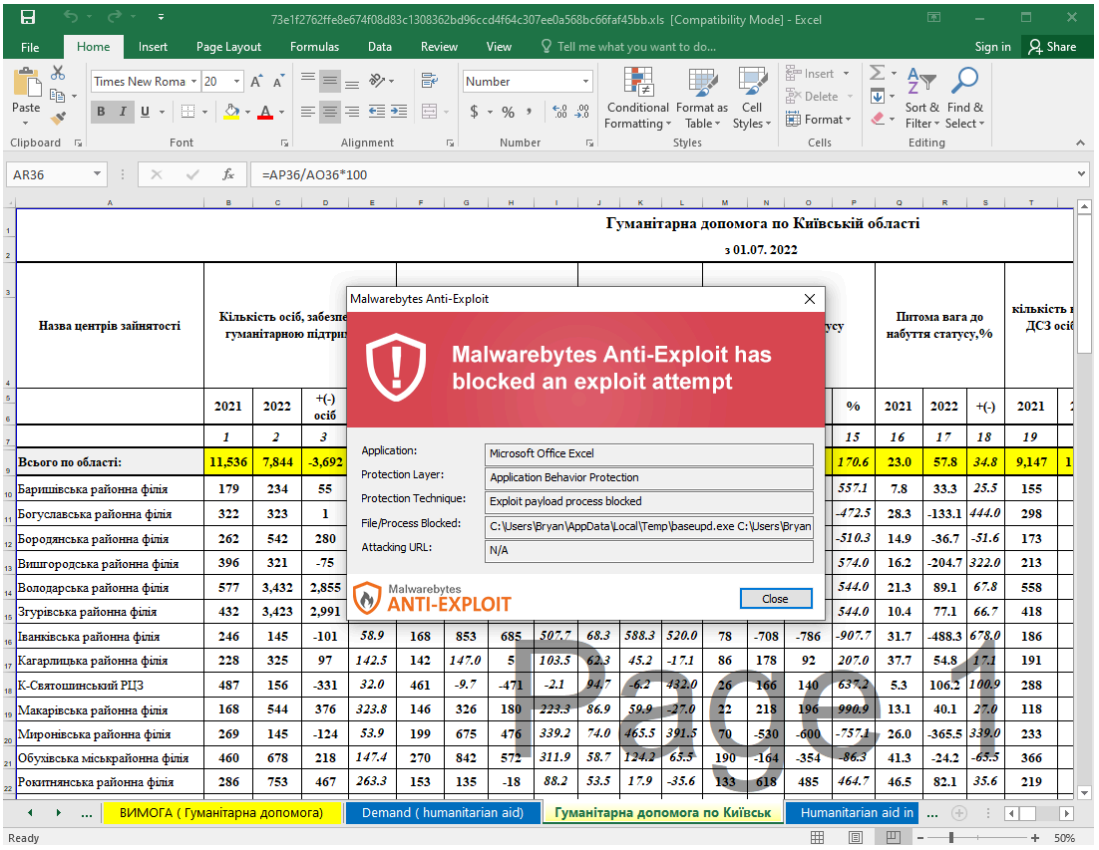
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

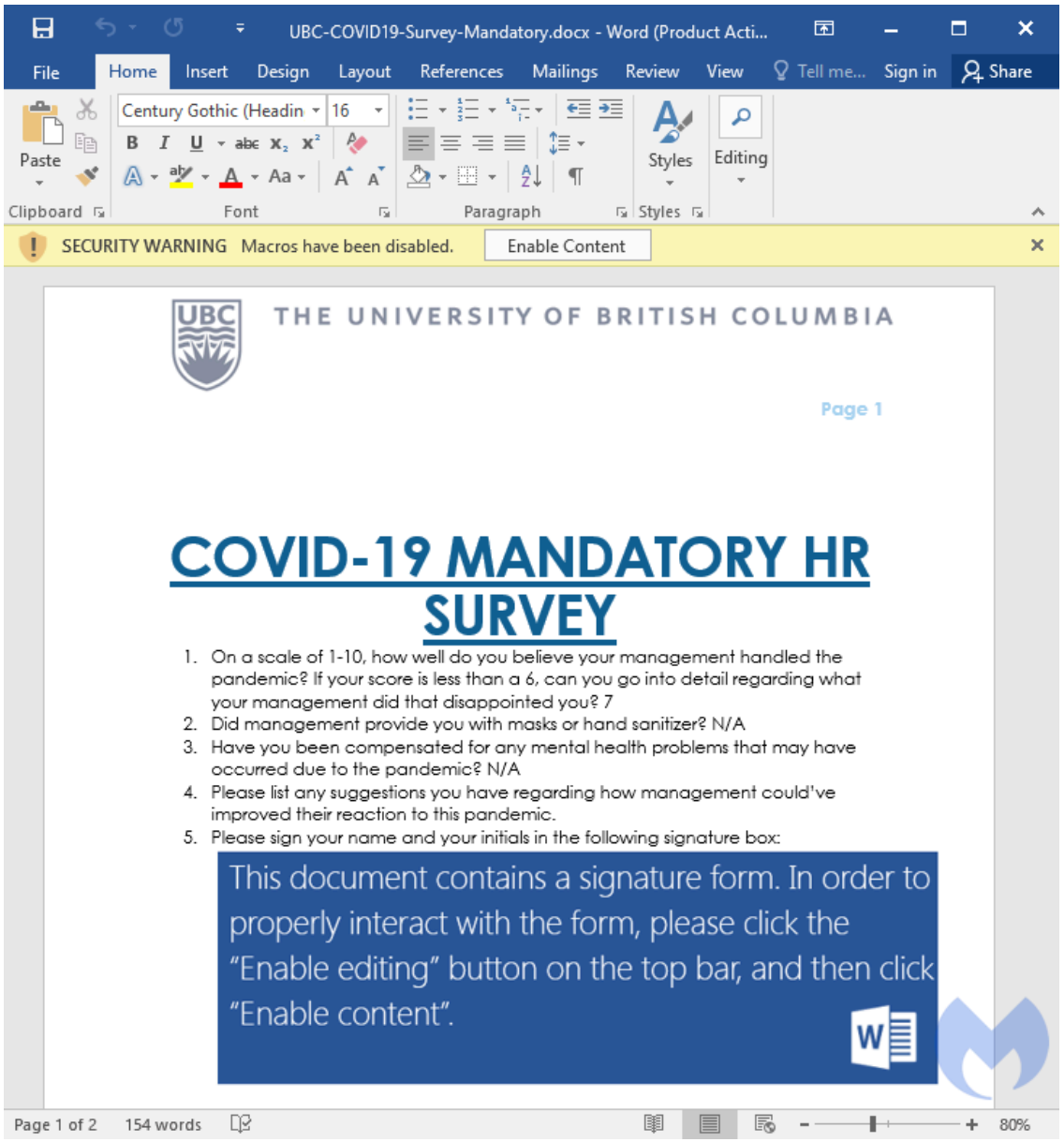
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

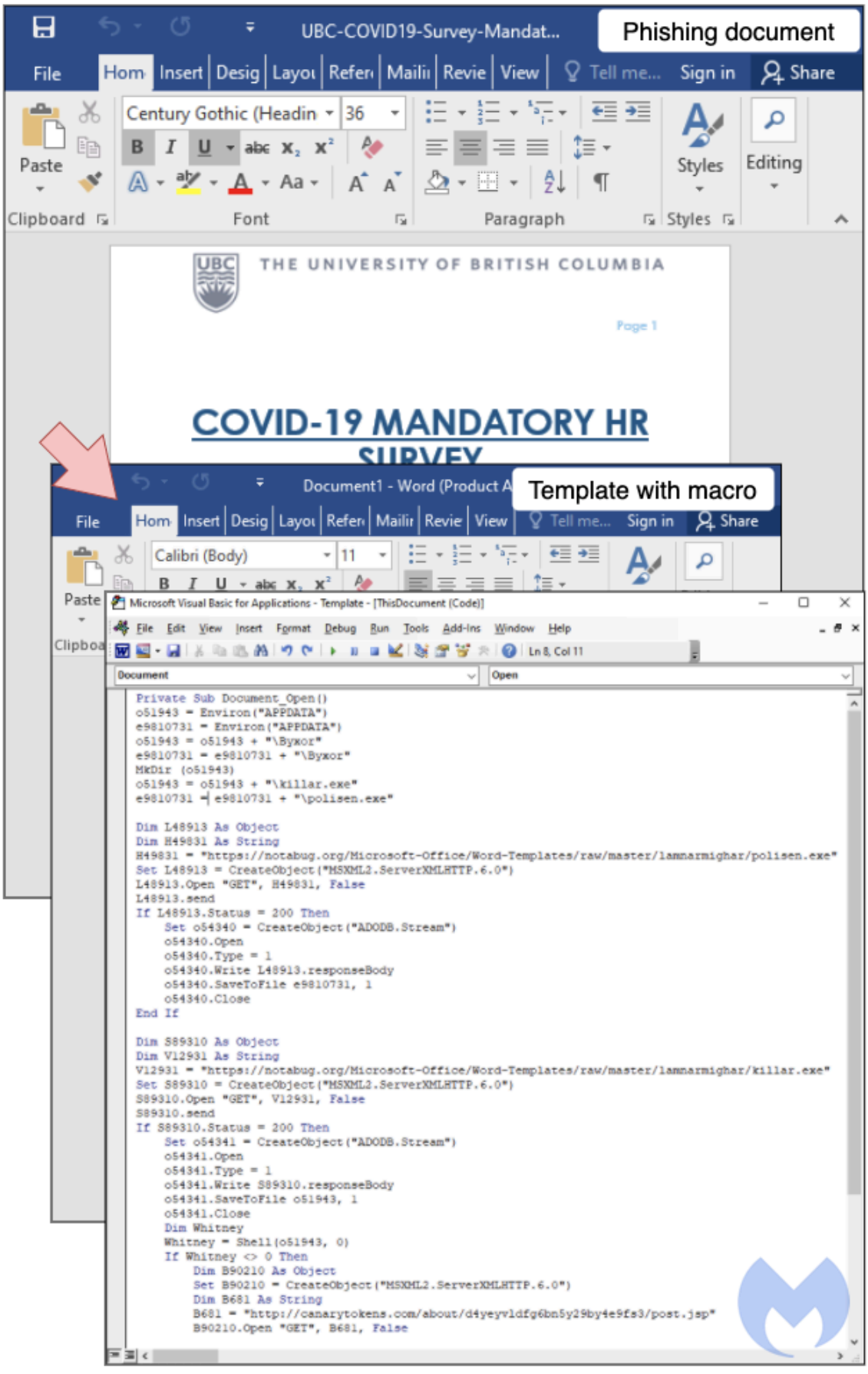
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

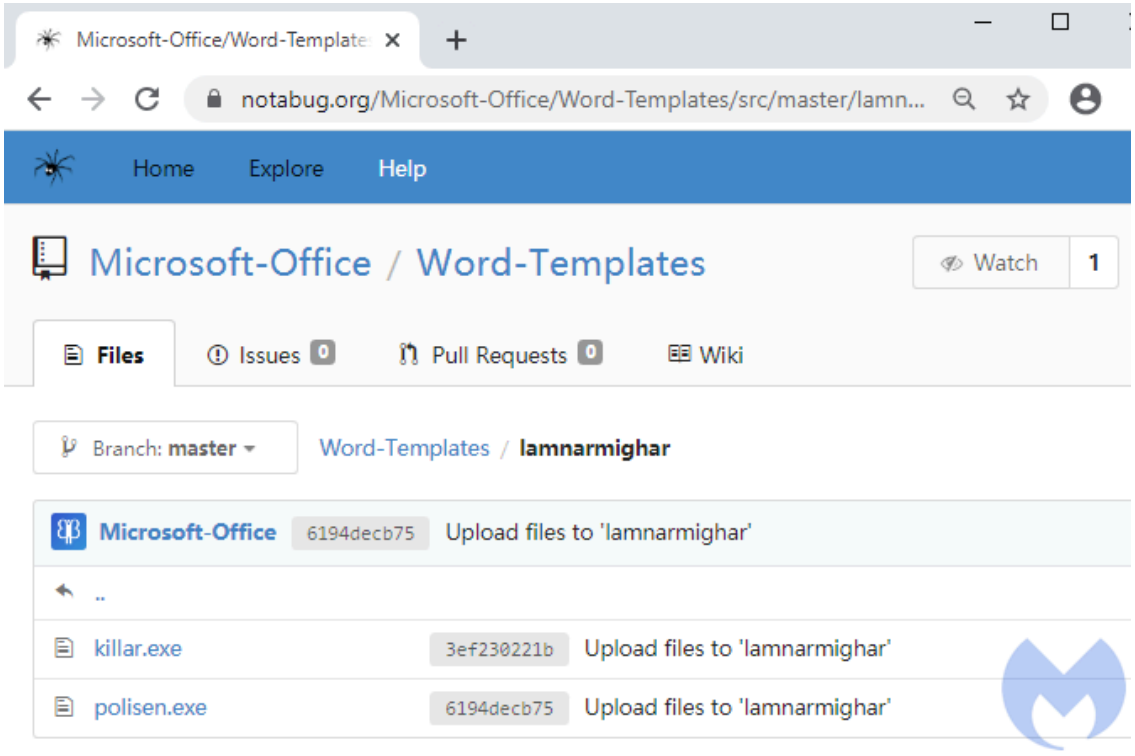
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



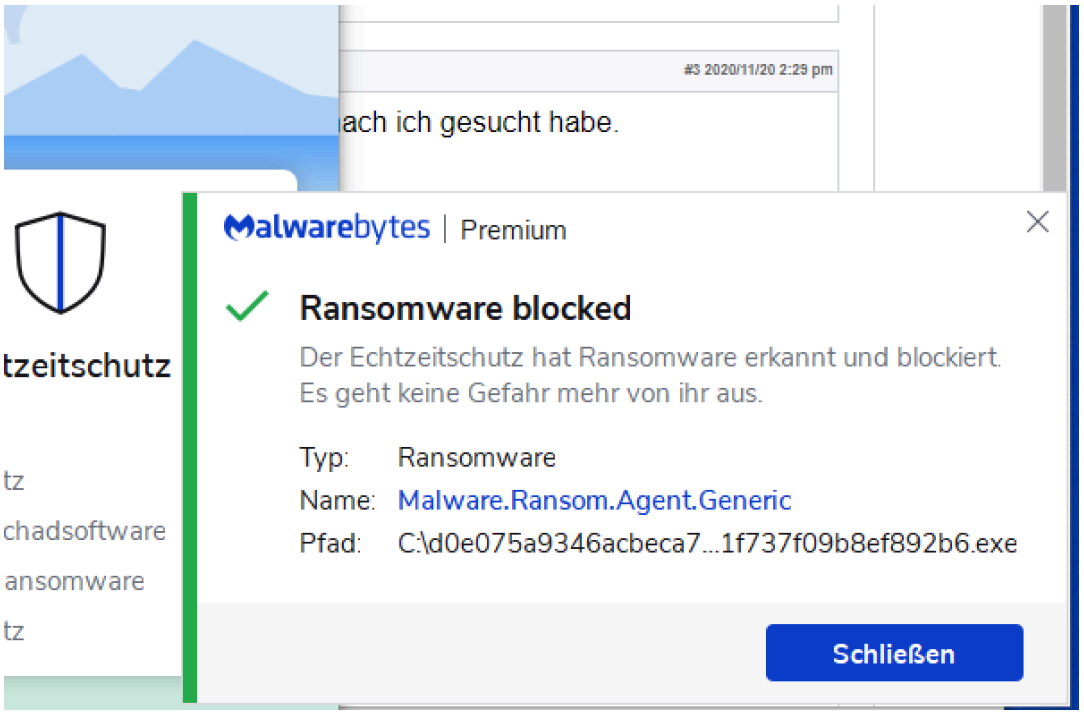
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

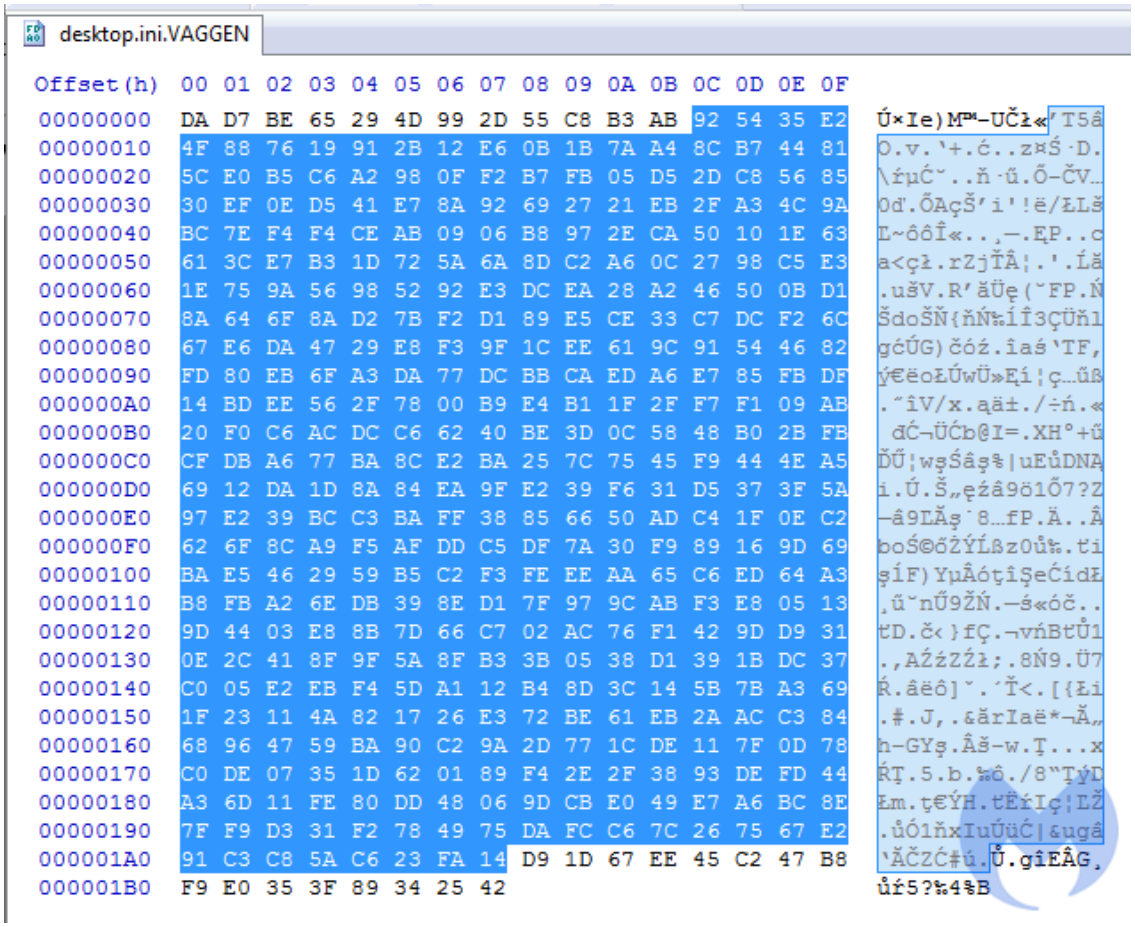
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



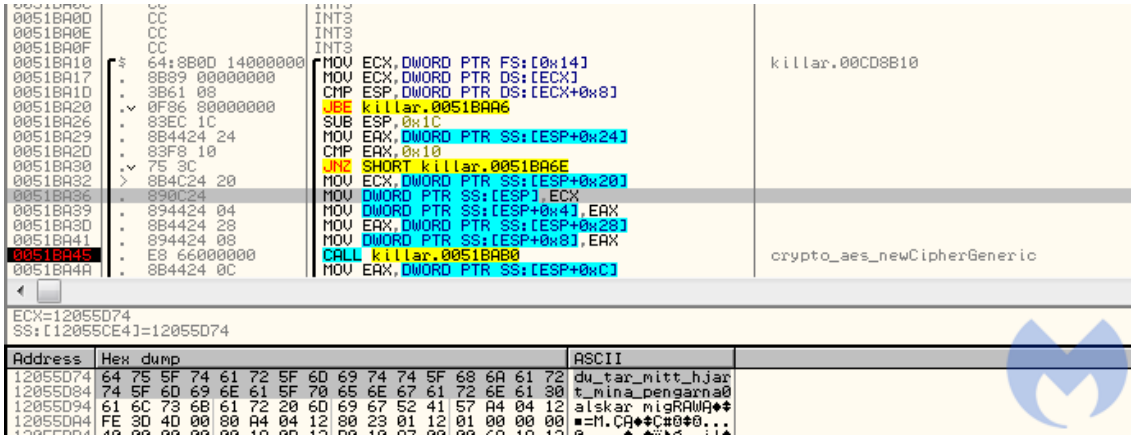
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

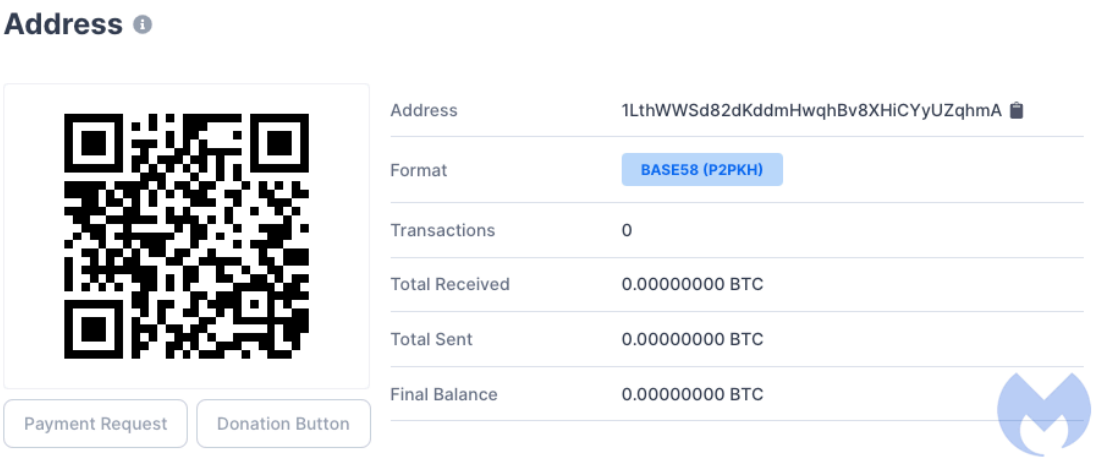
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

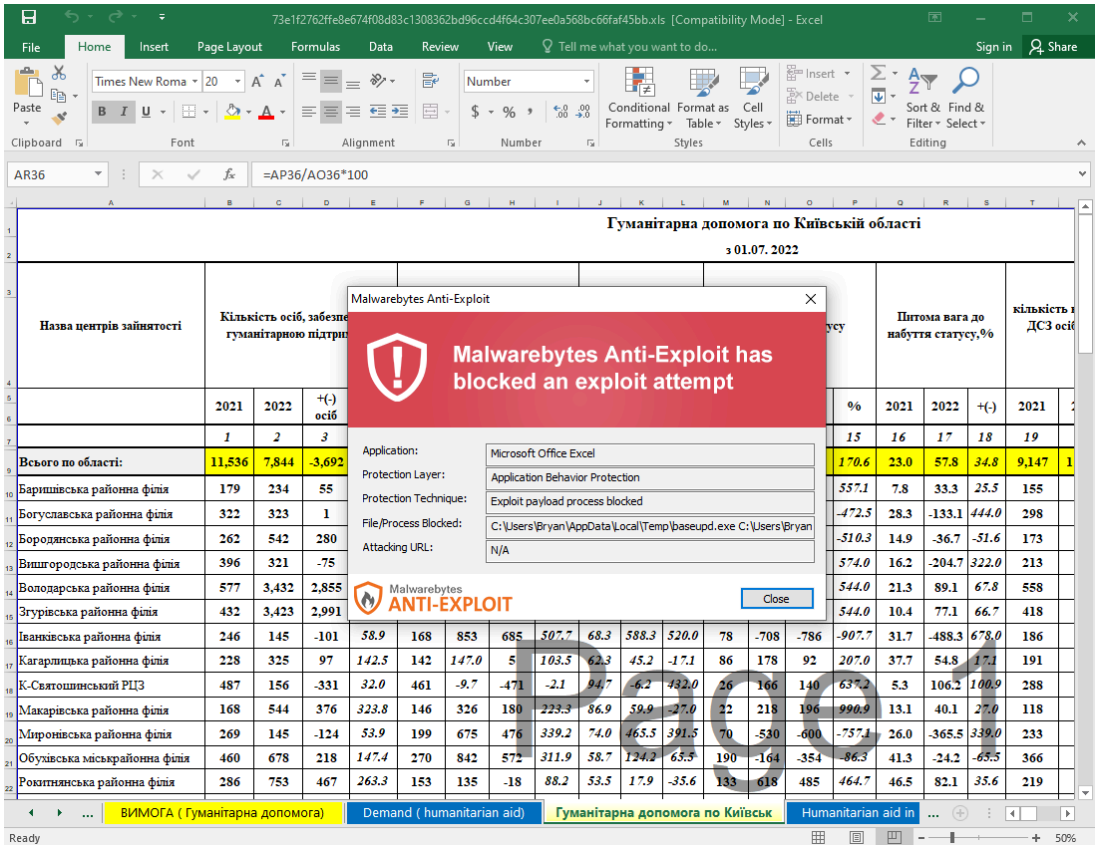
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

Mandatory COVID-19 survey distributed to targeted recipients

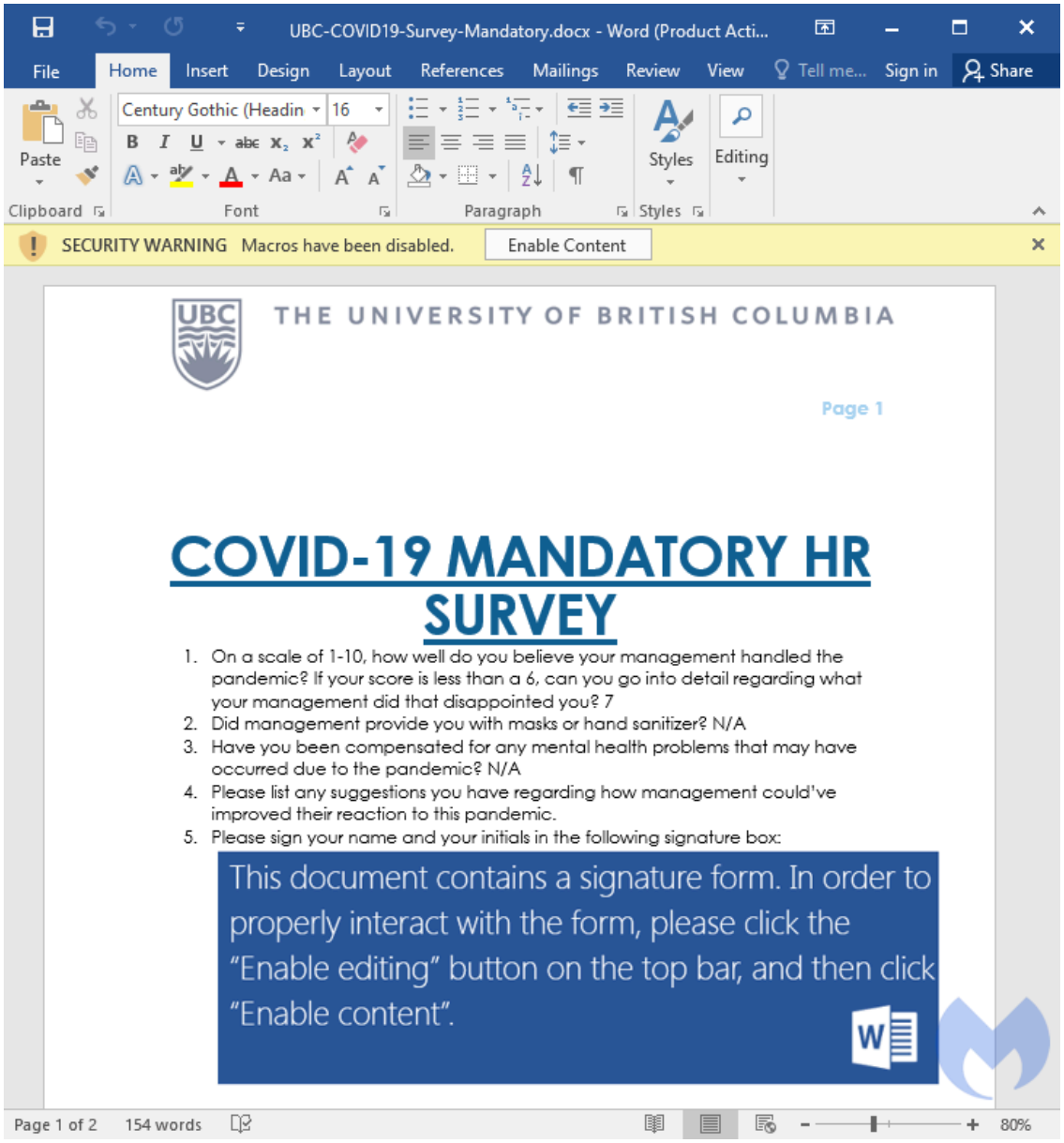
The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

Article continues below this ad.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

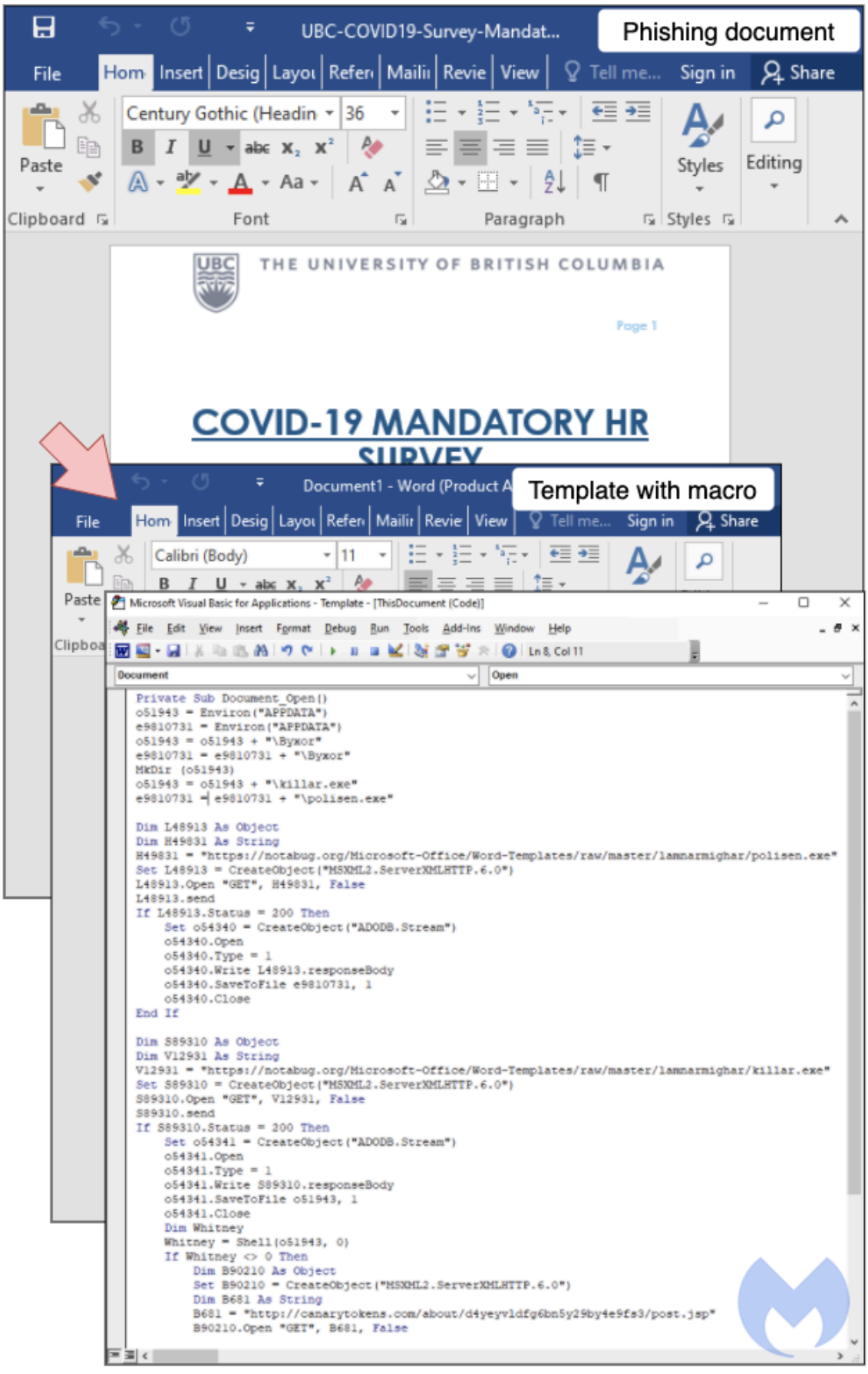
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

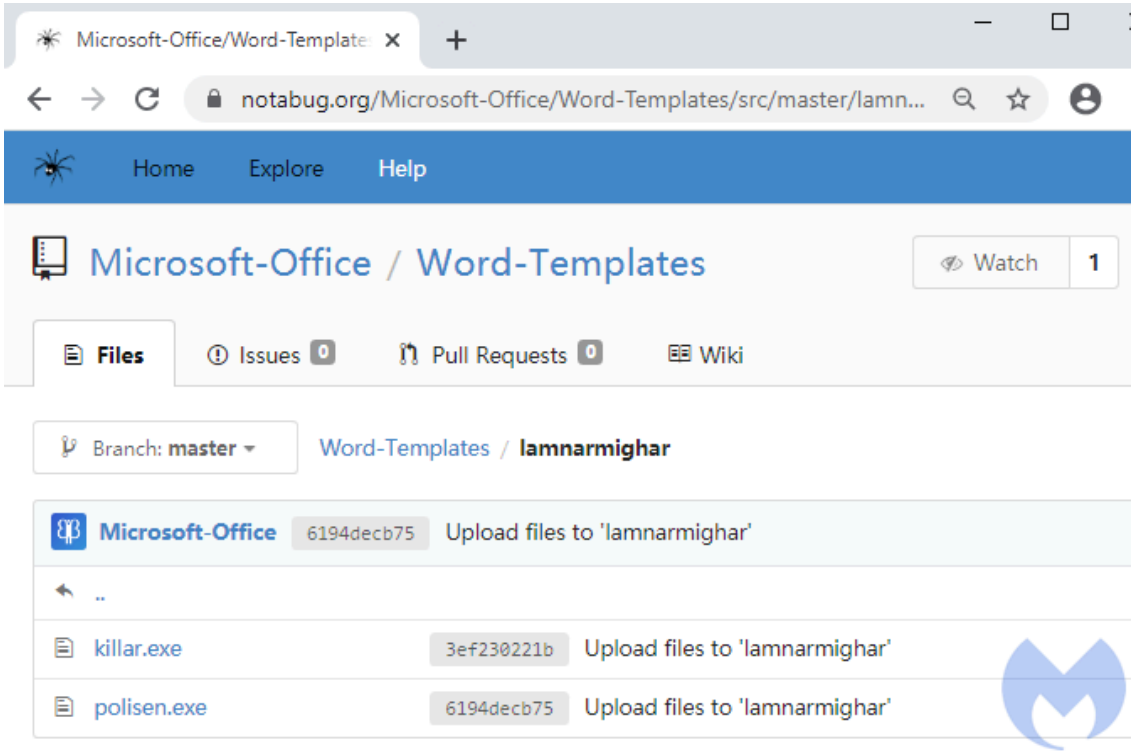
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



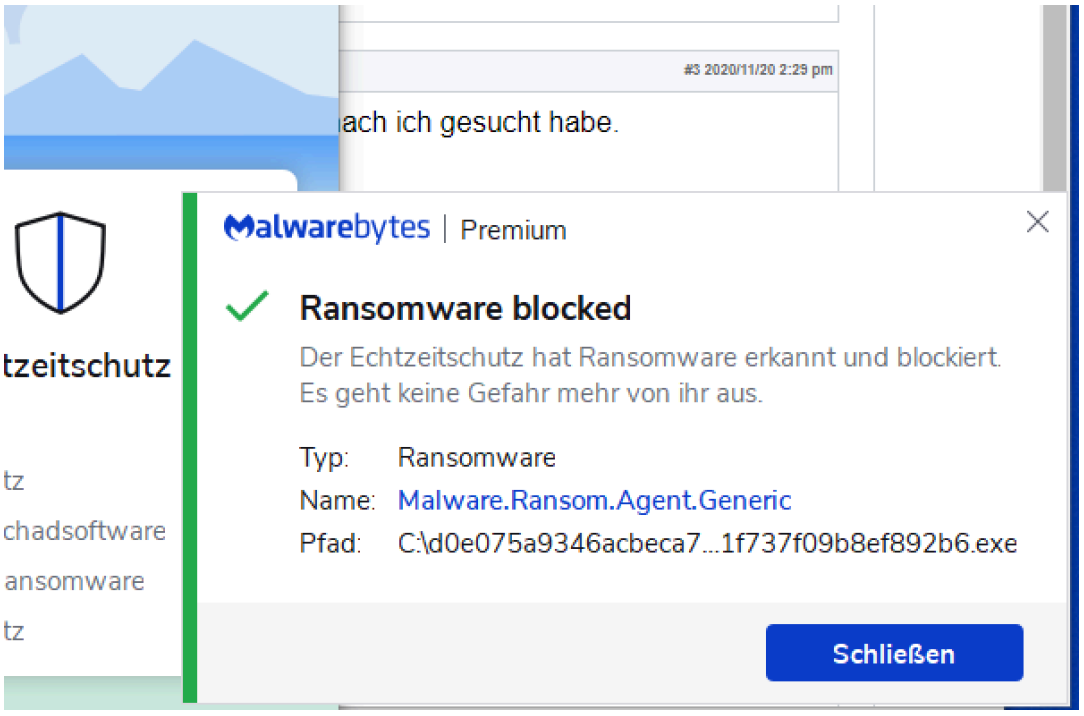
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

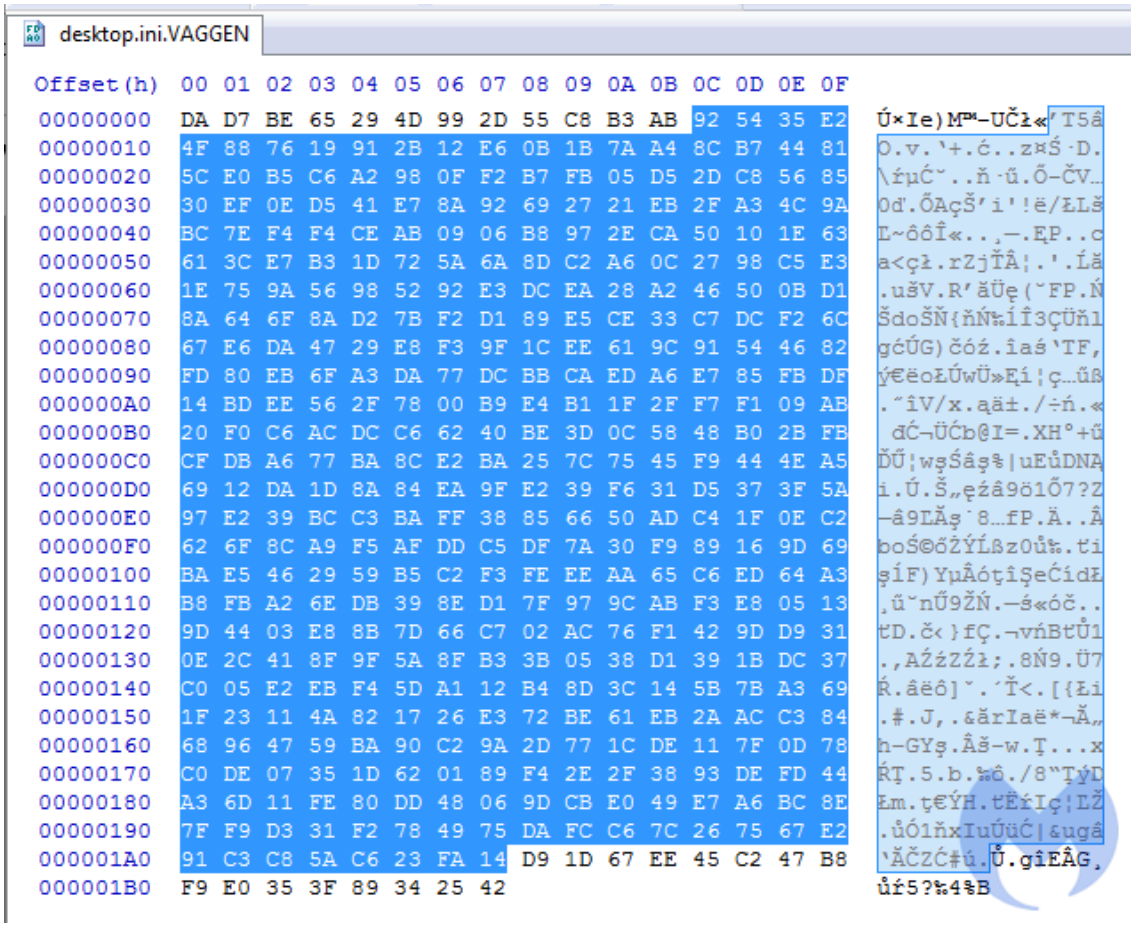
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

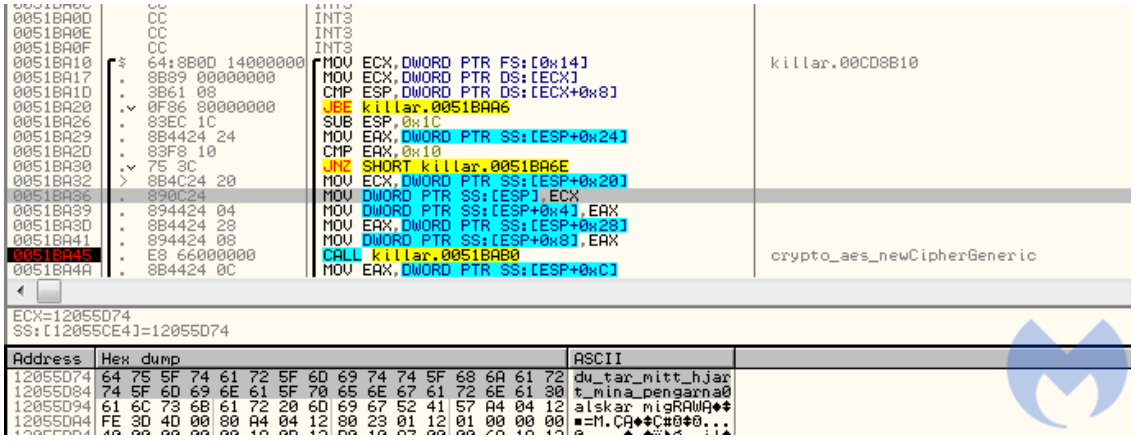


The content of the output file (with .VAGGEN extension) contains:

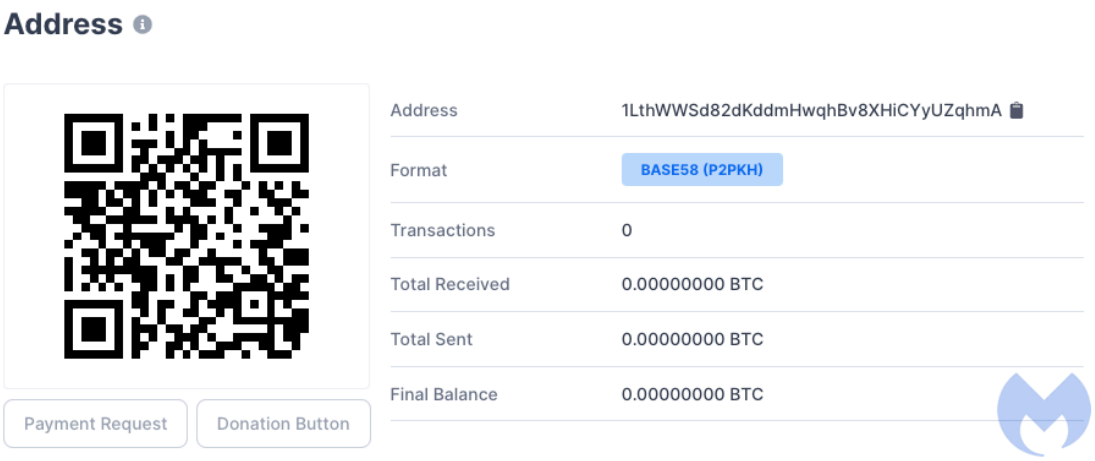
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

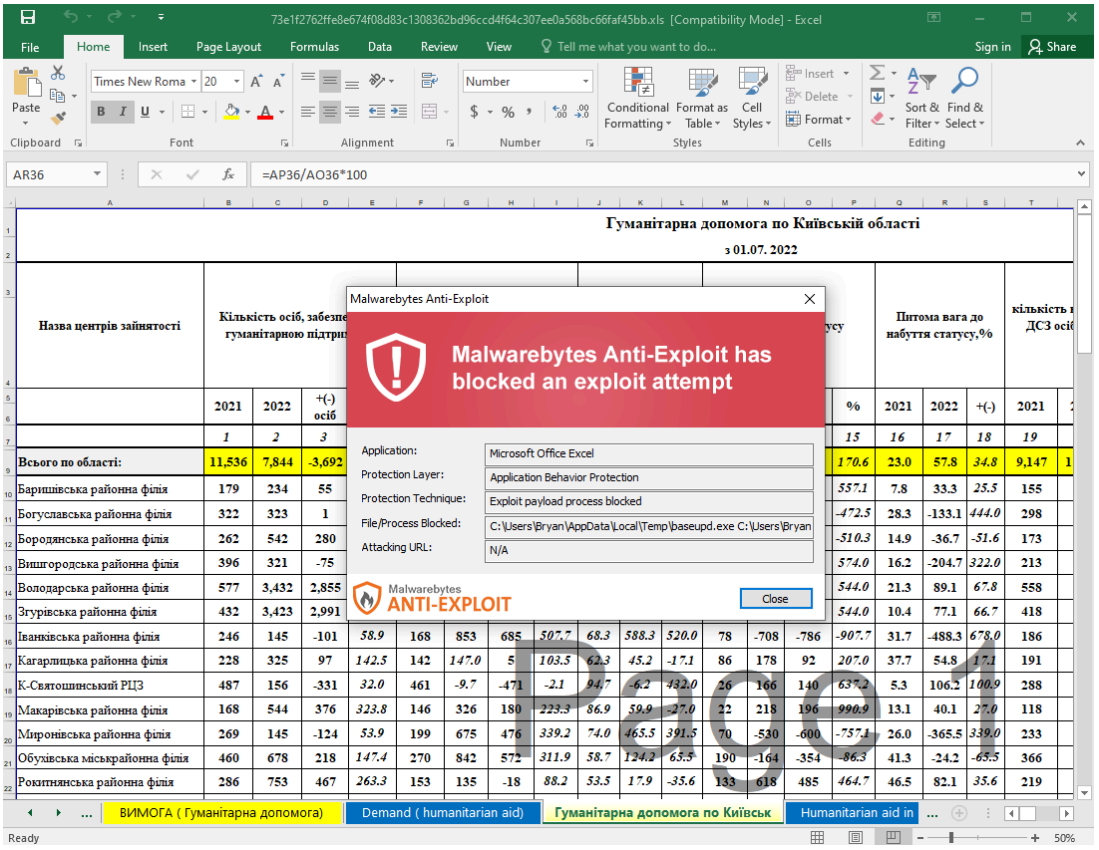
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

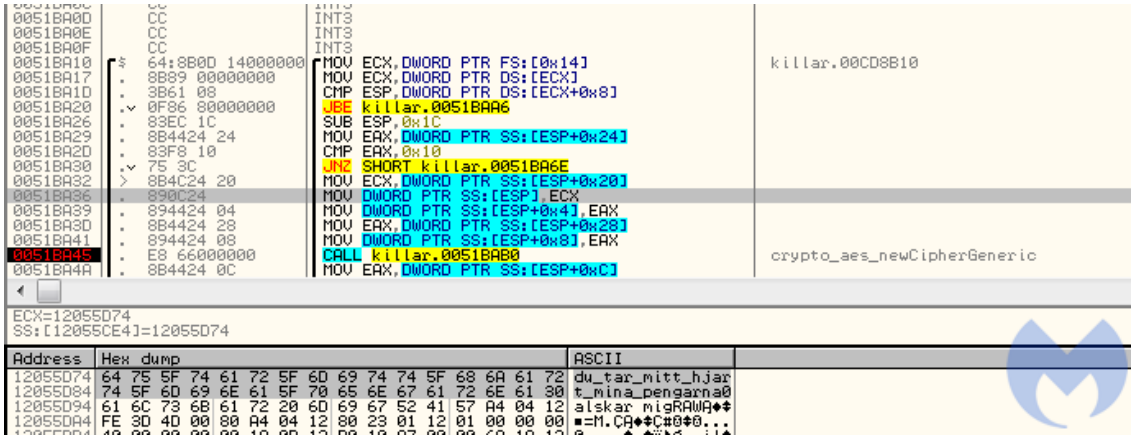
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

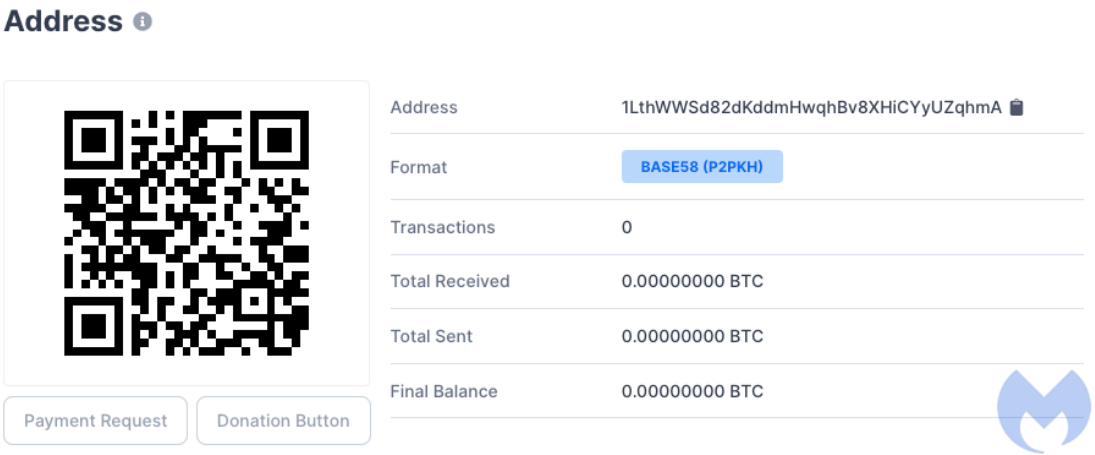
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

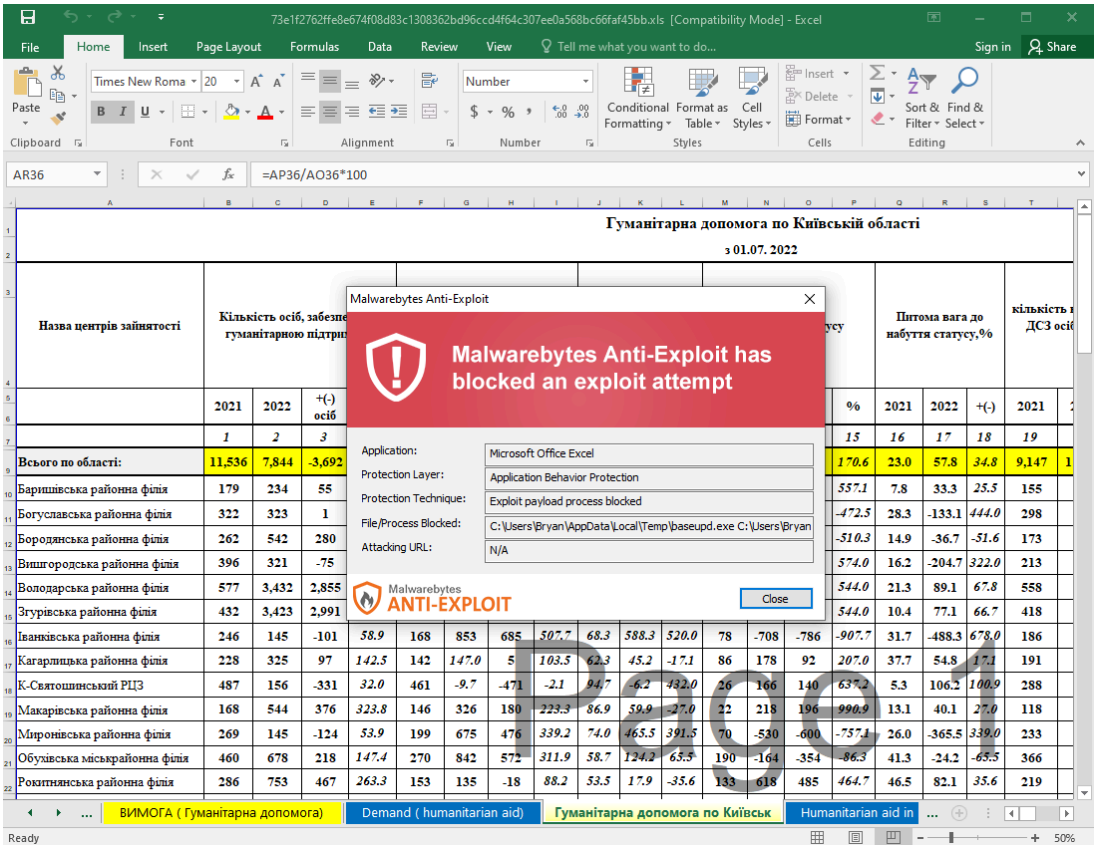
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

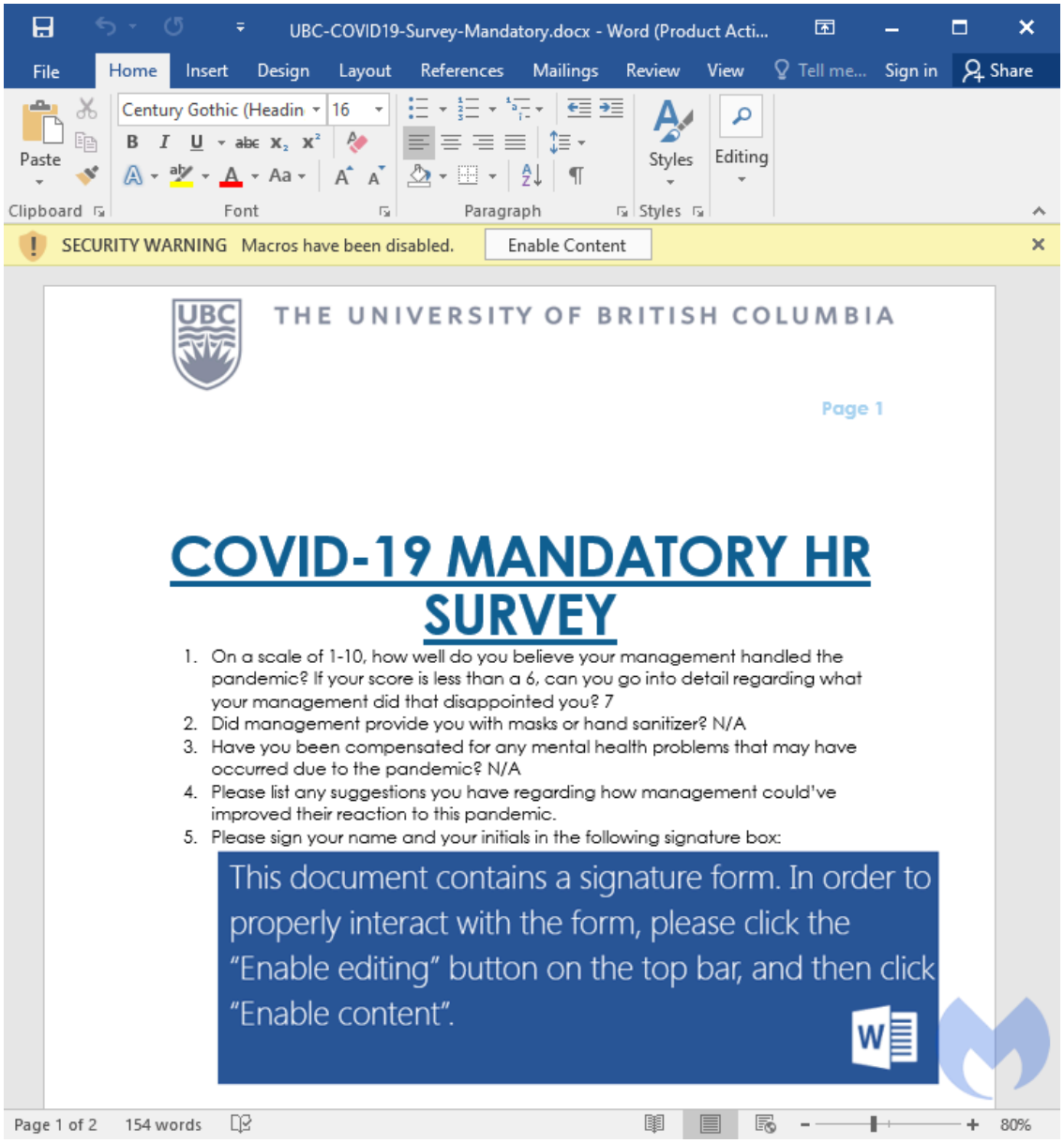
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

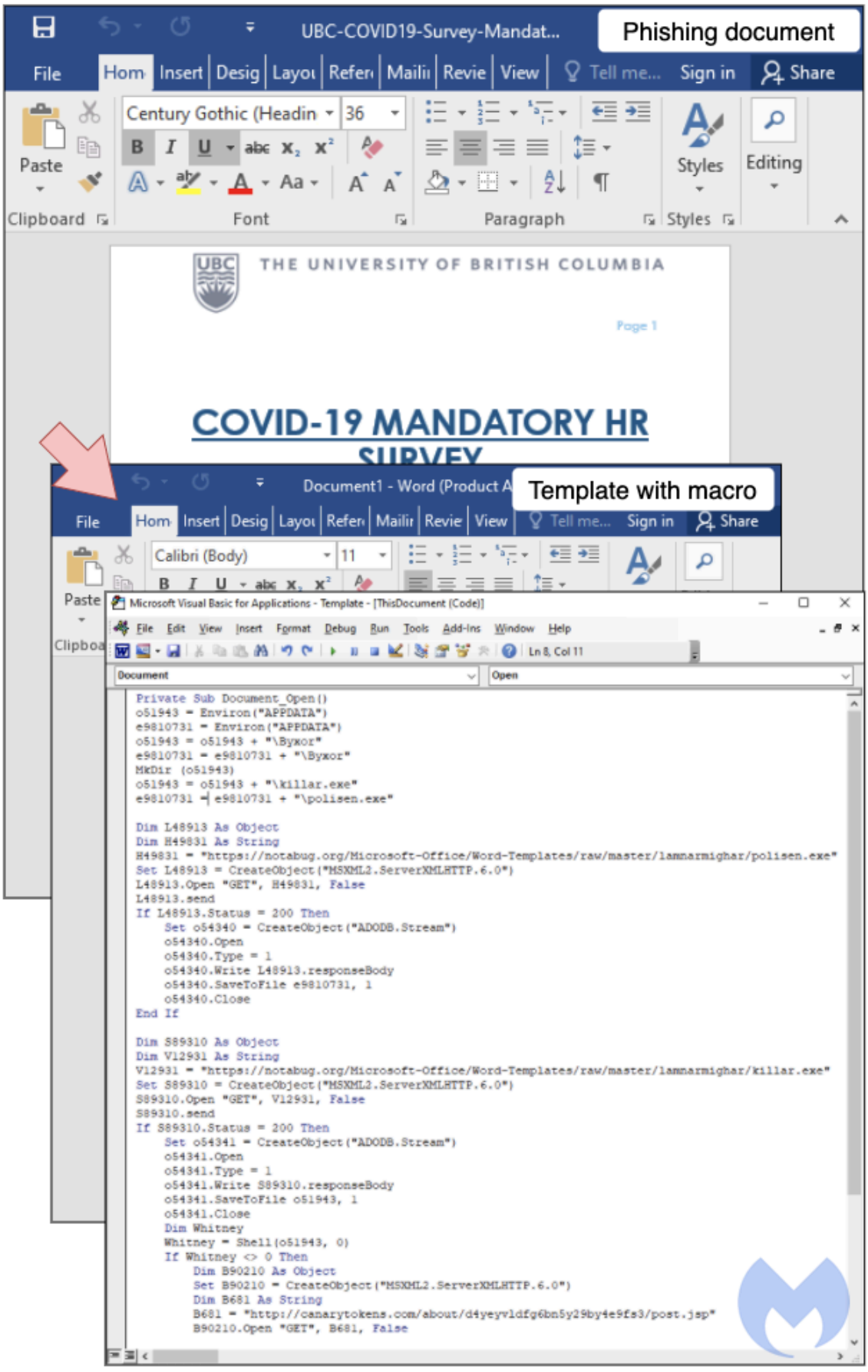
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

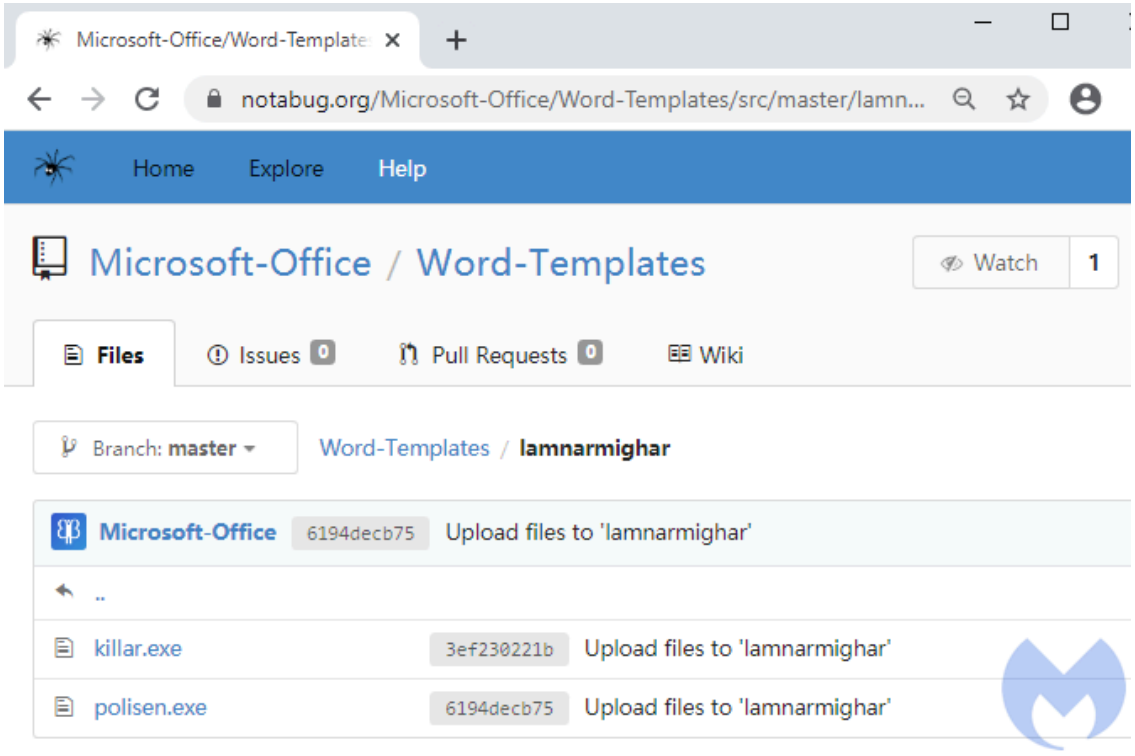
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



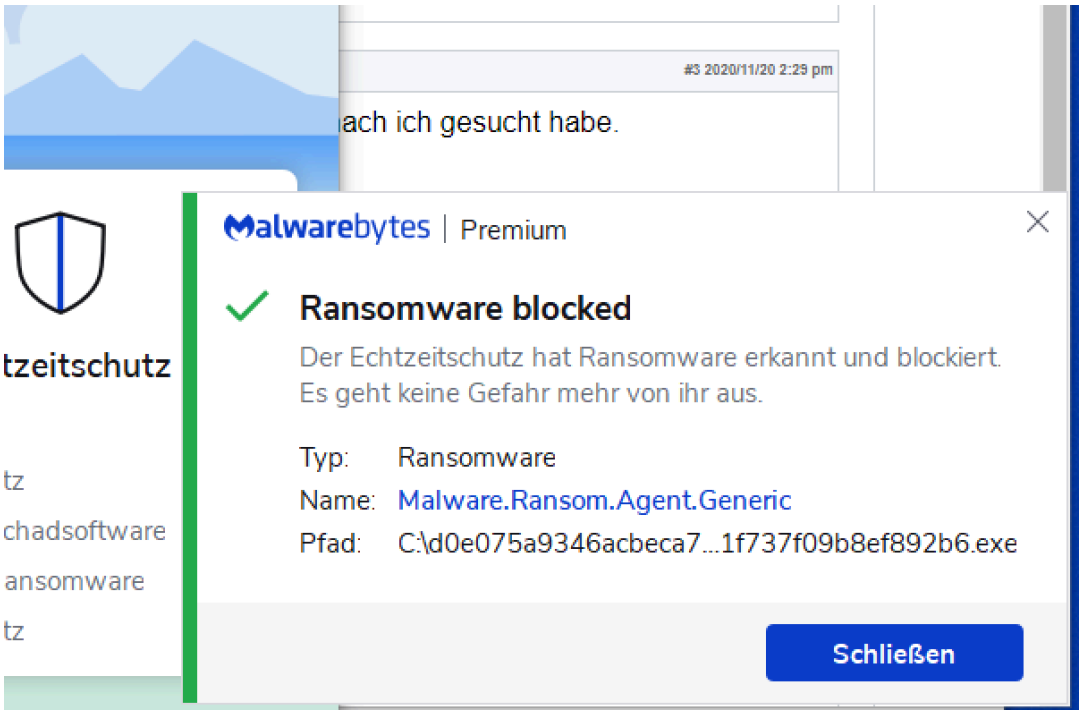
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

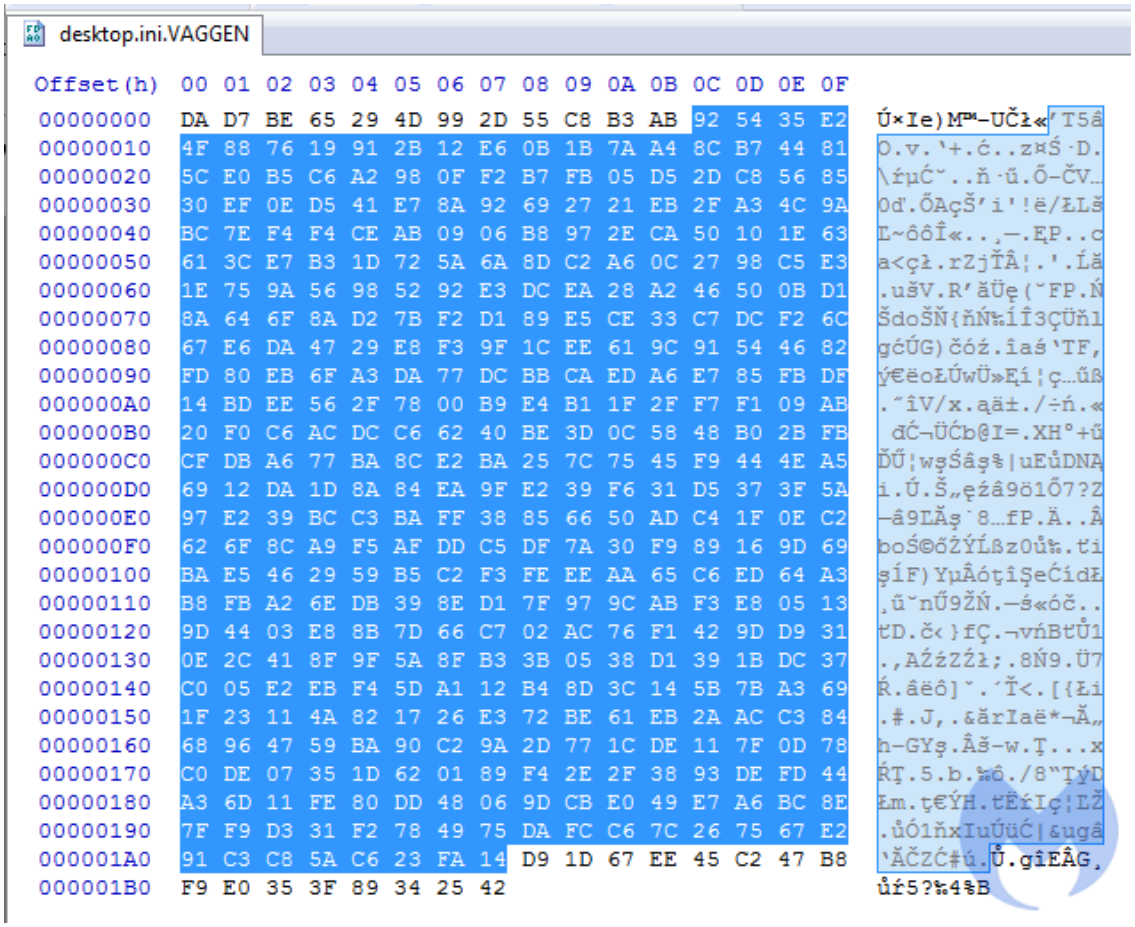
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

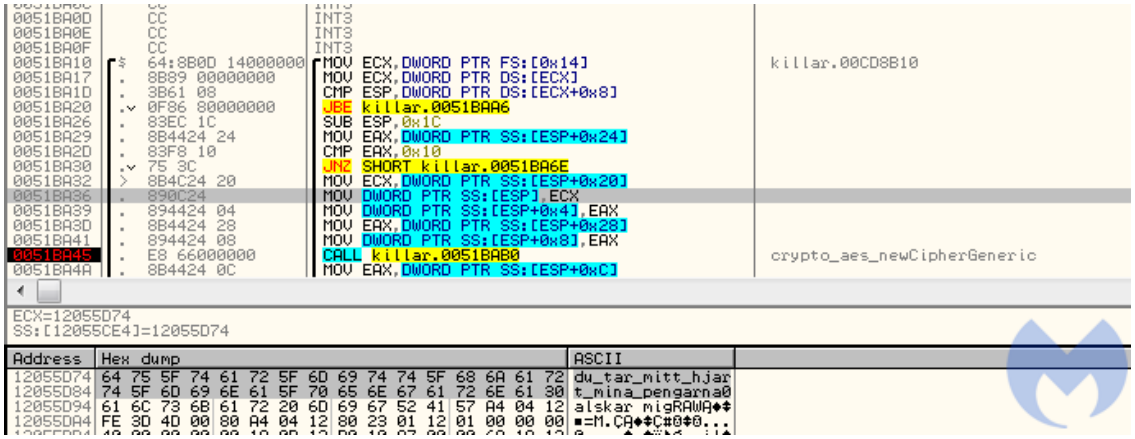


The content of the output file (with .VAGGEN extension) contains:

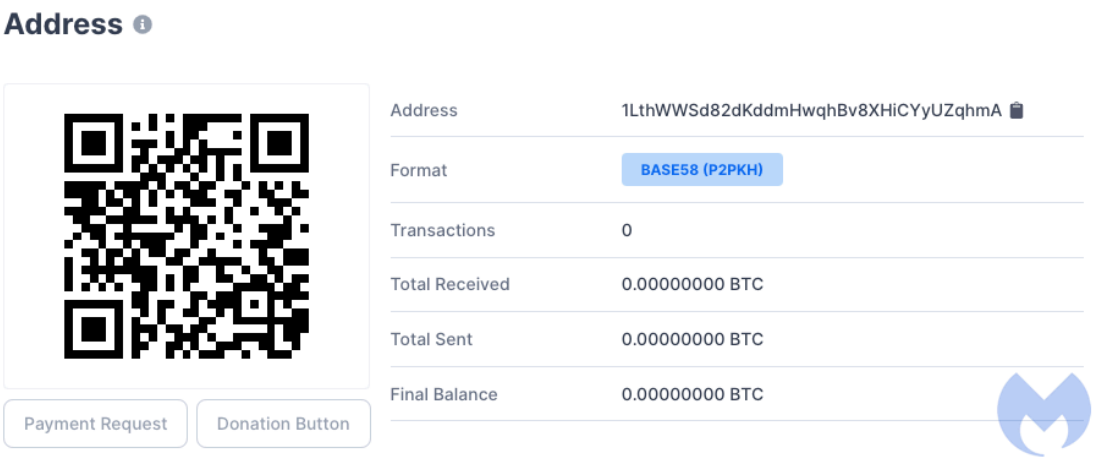
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

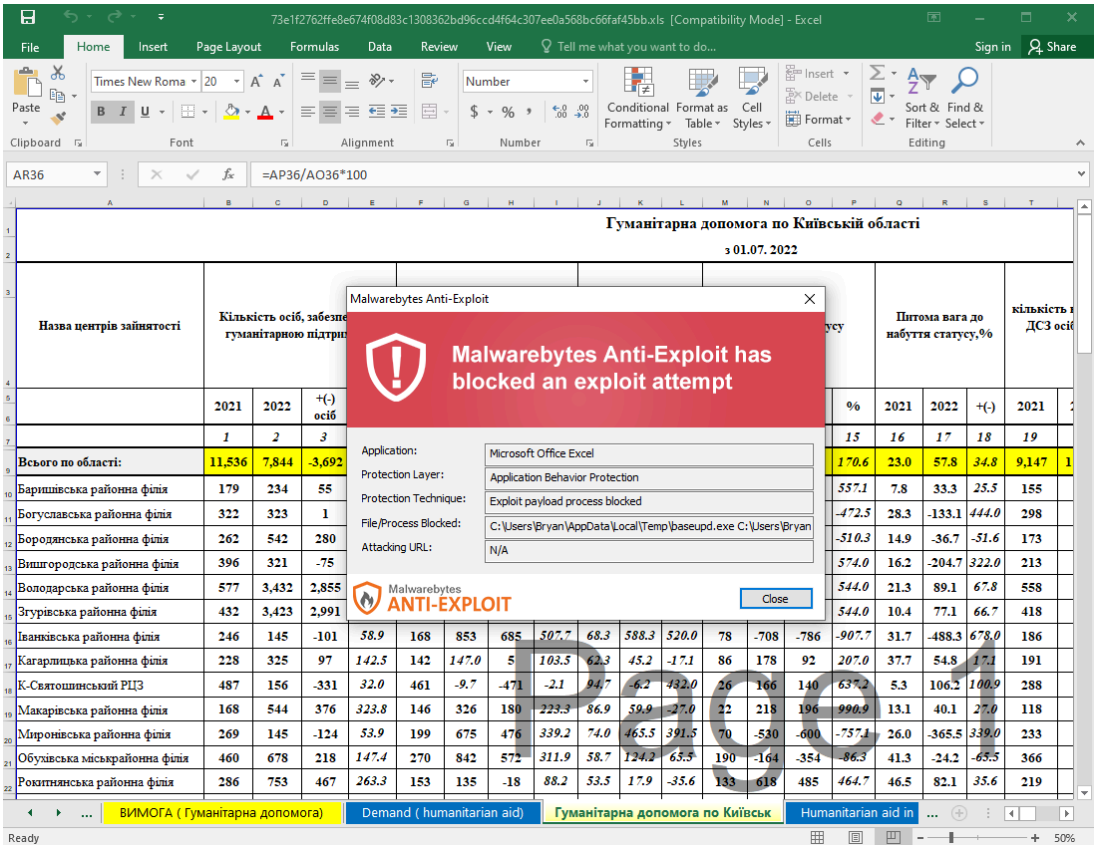
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

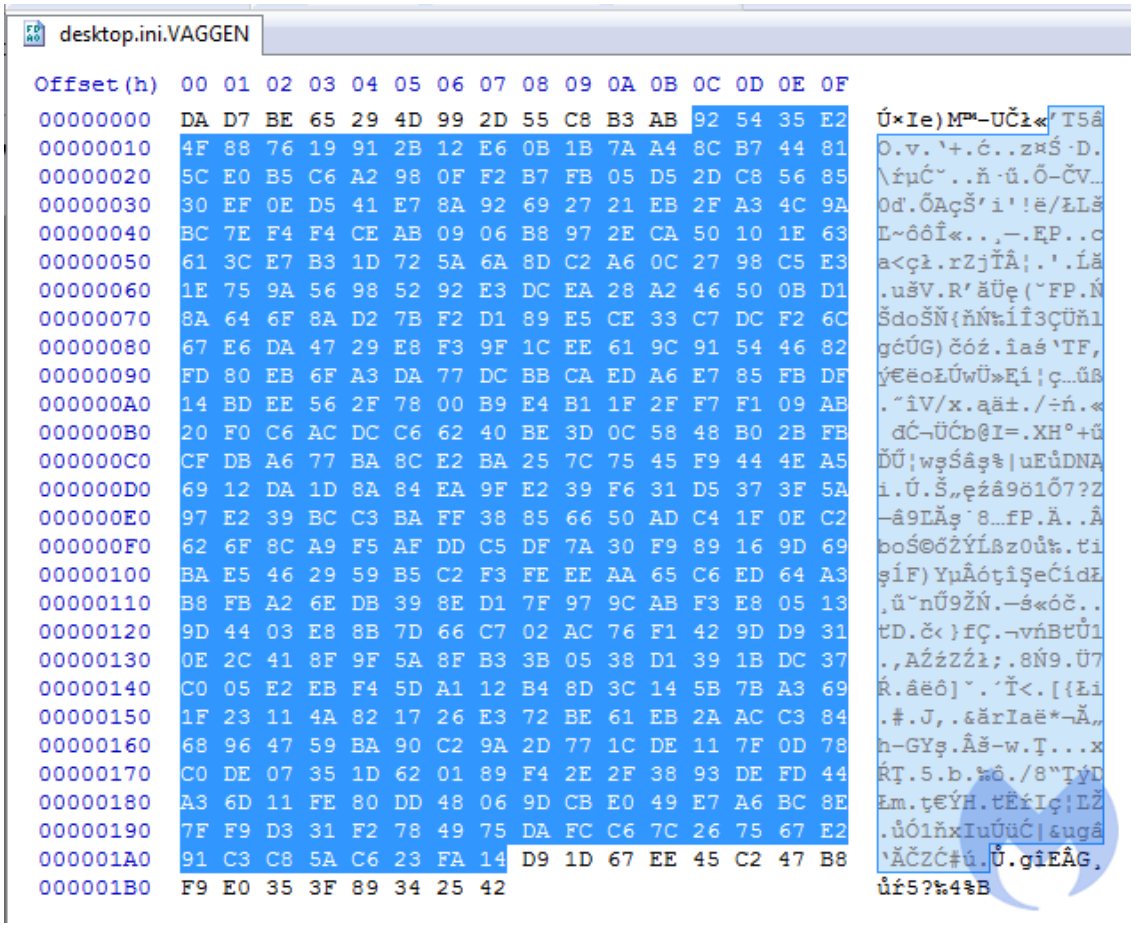
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

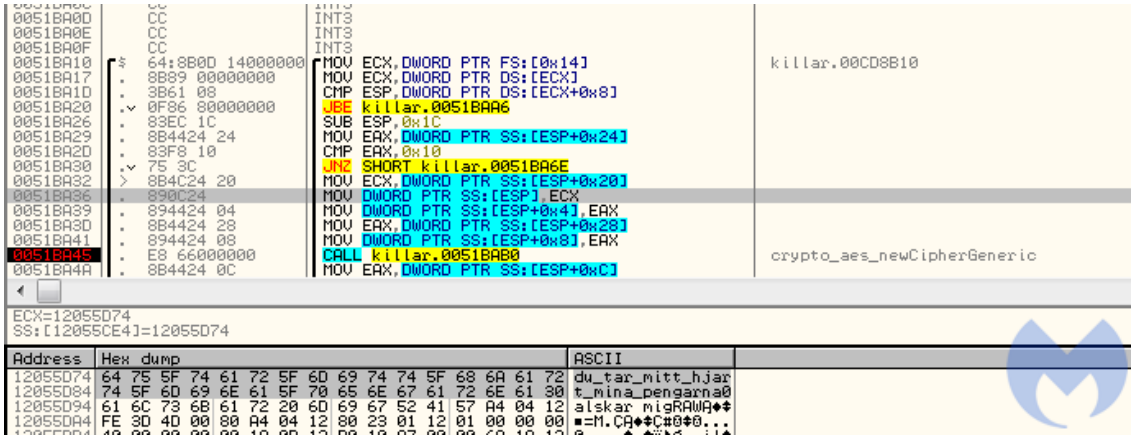
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

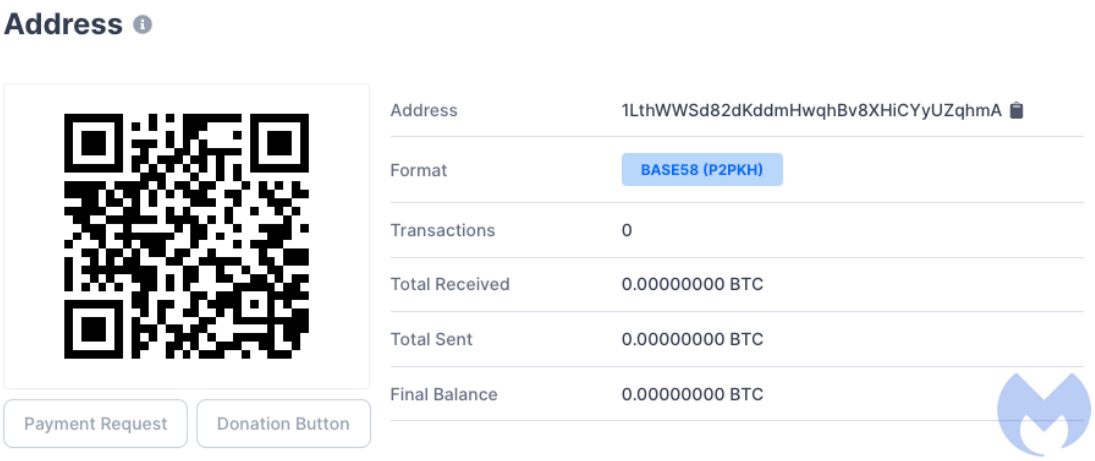
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

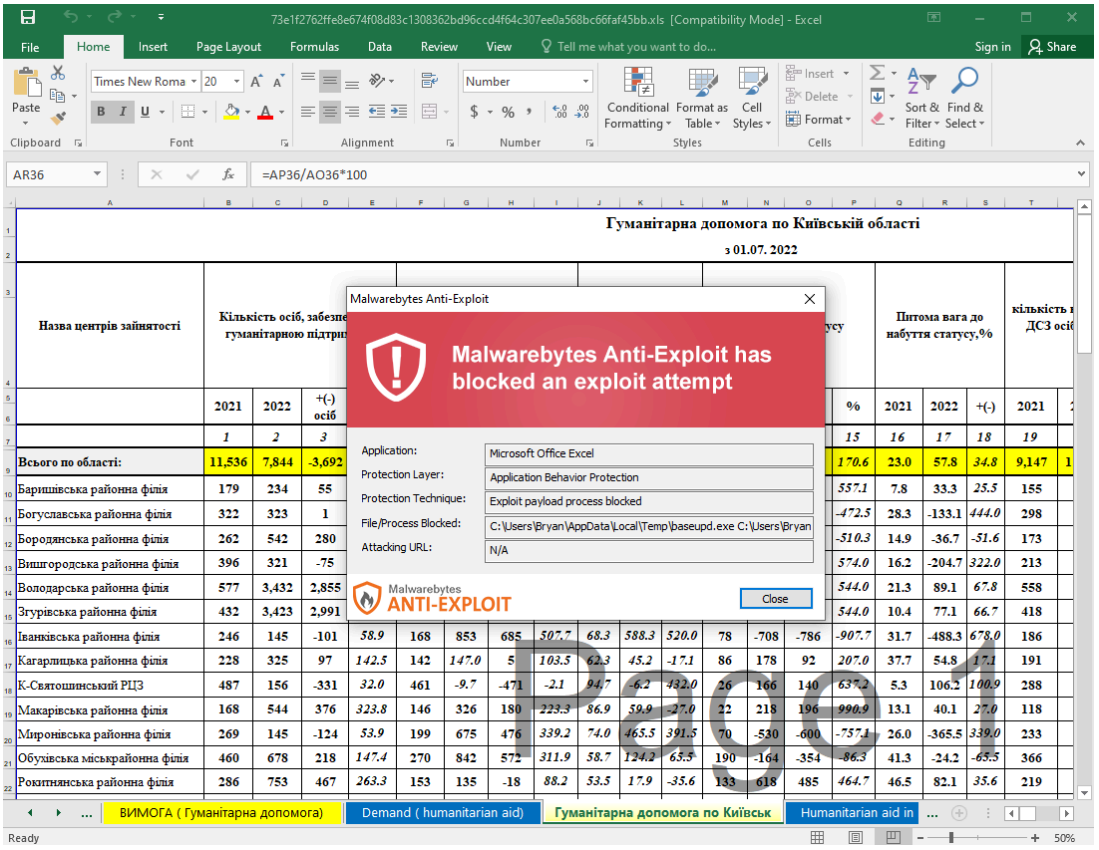
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

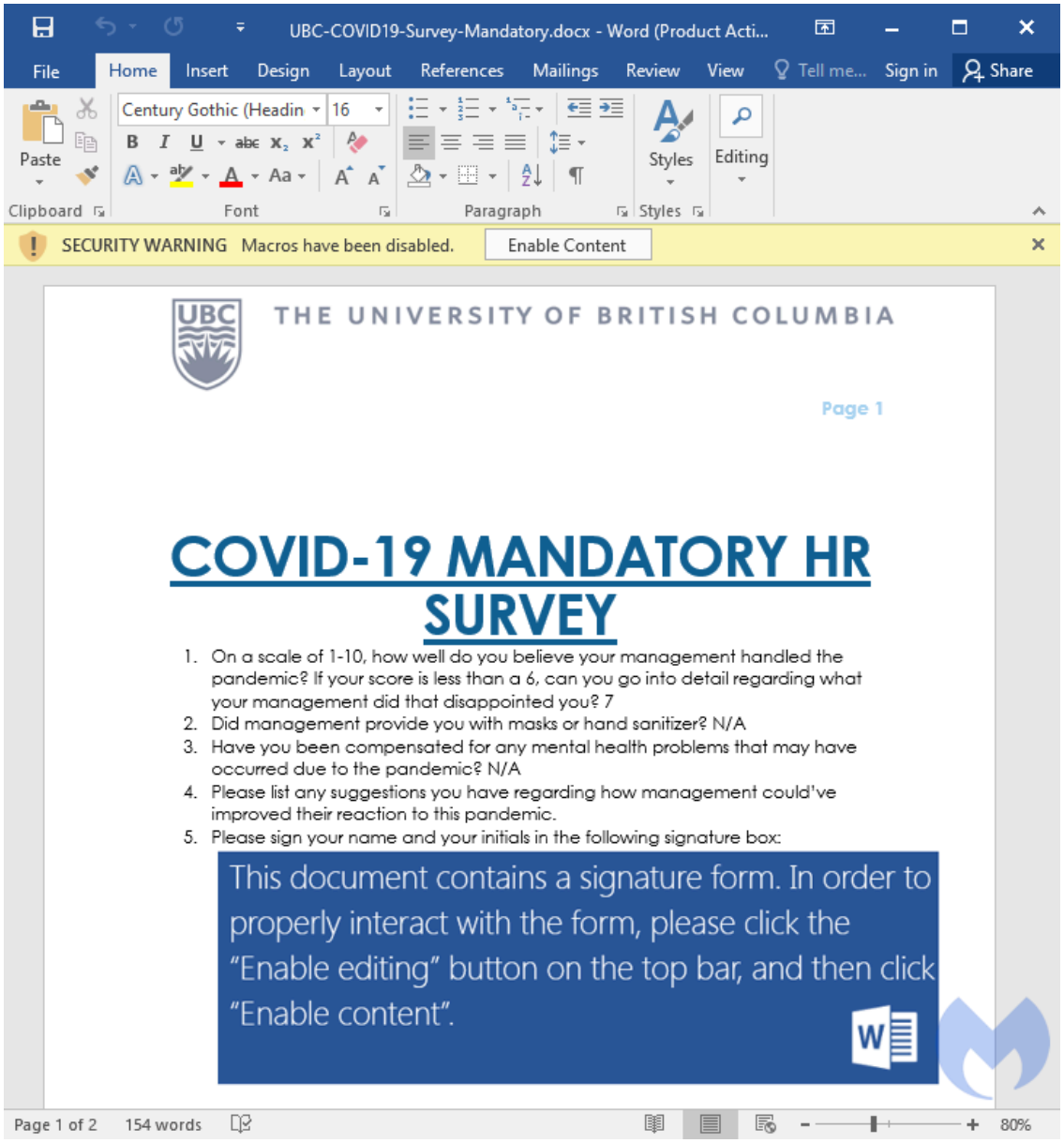
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

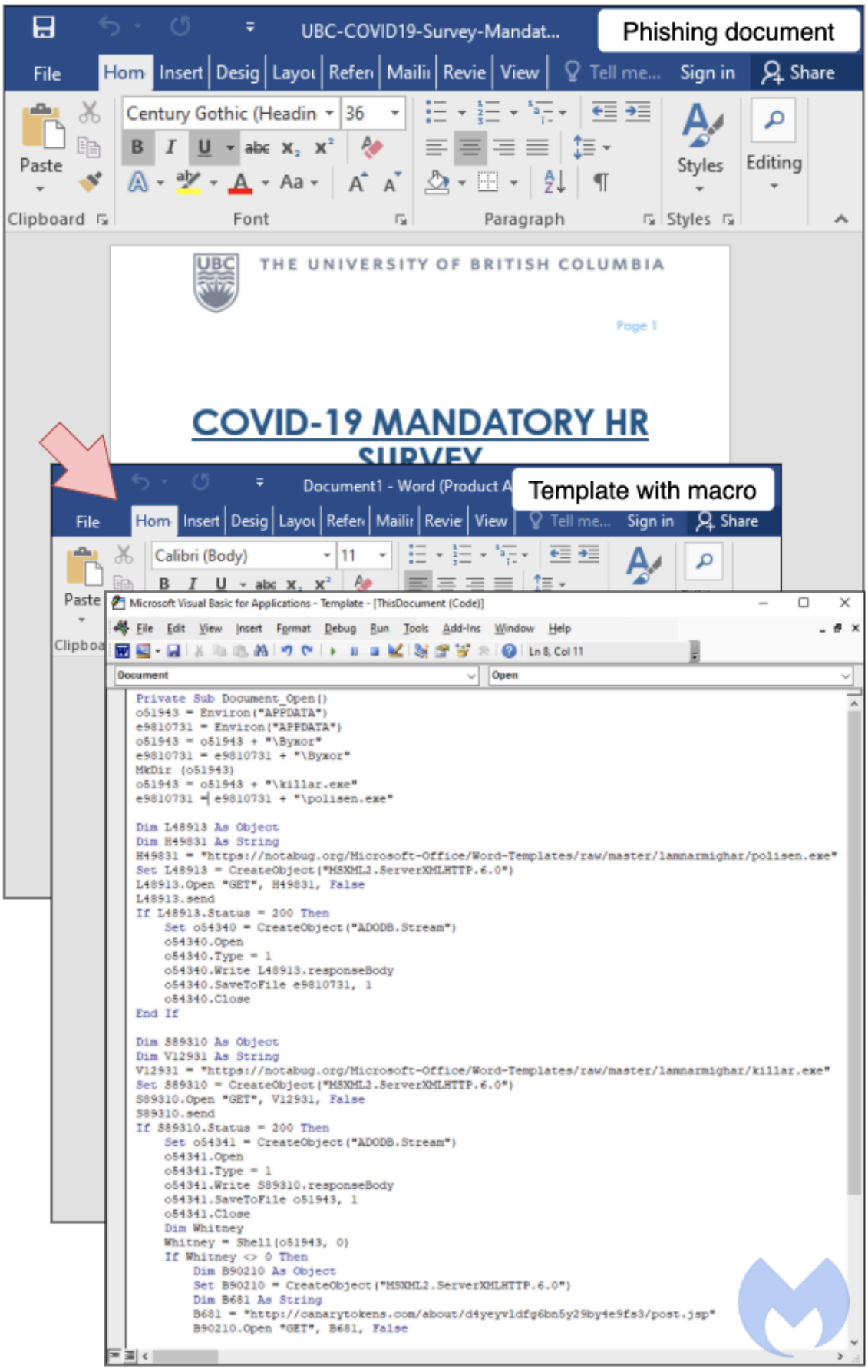
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

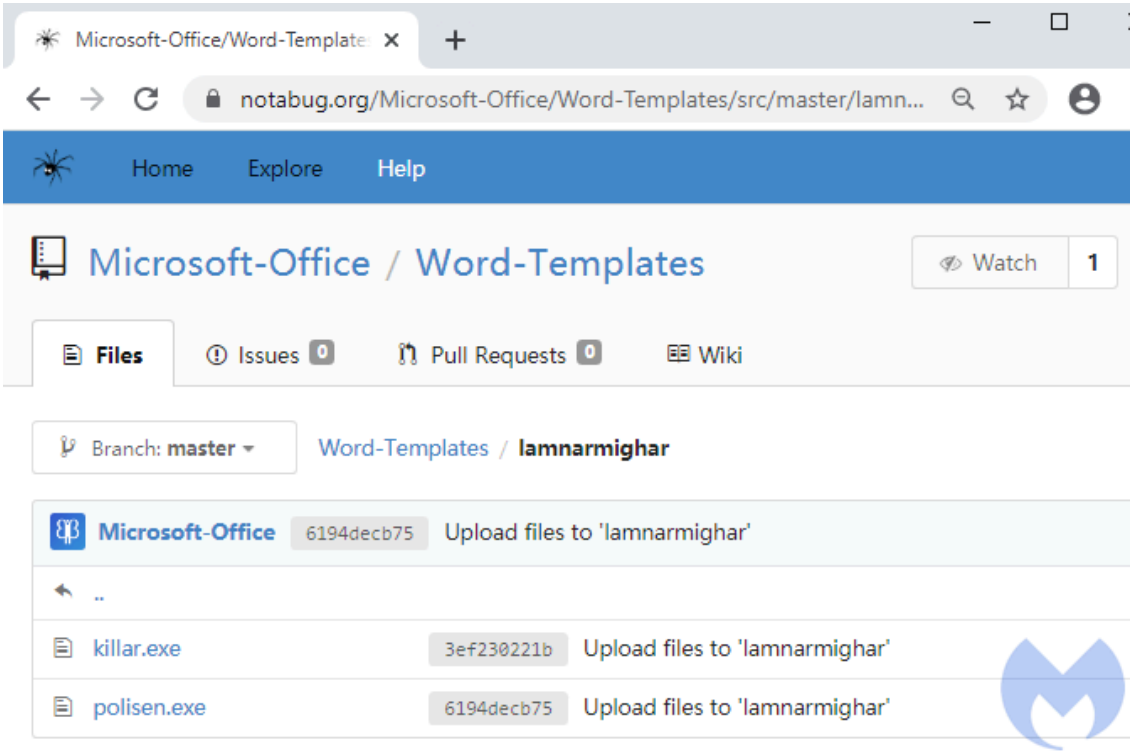
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



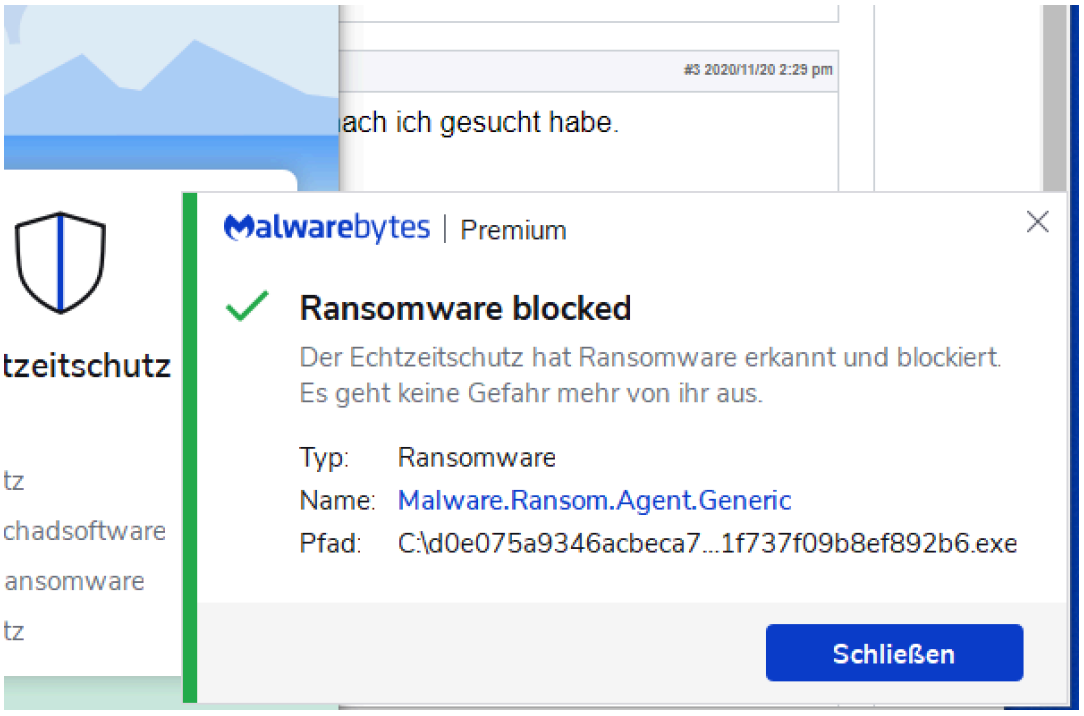
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

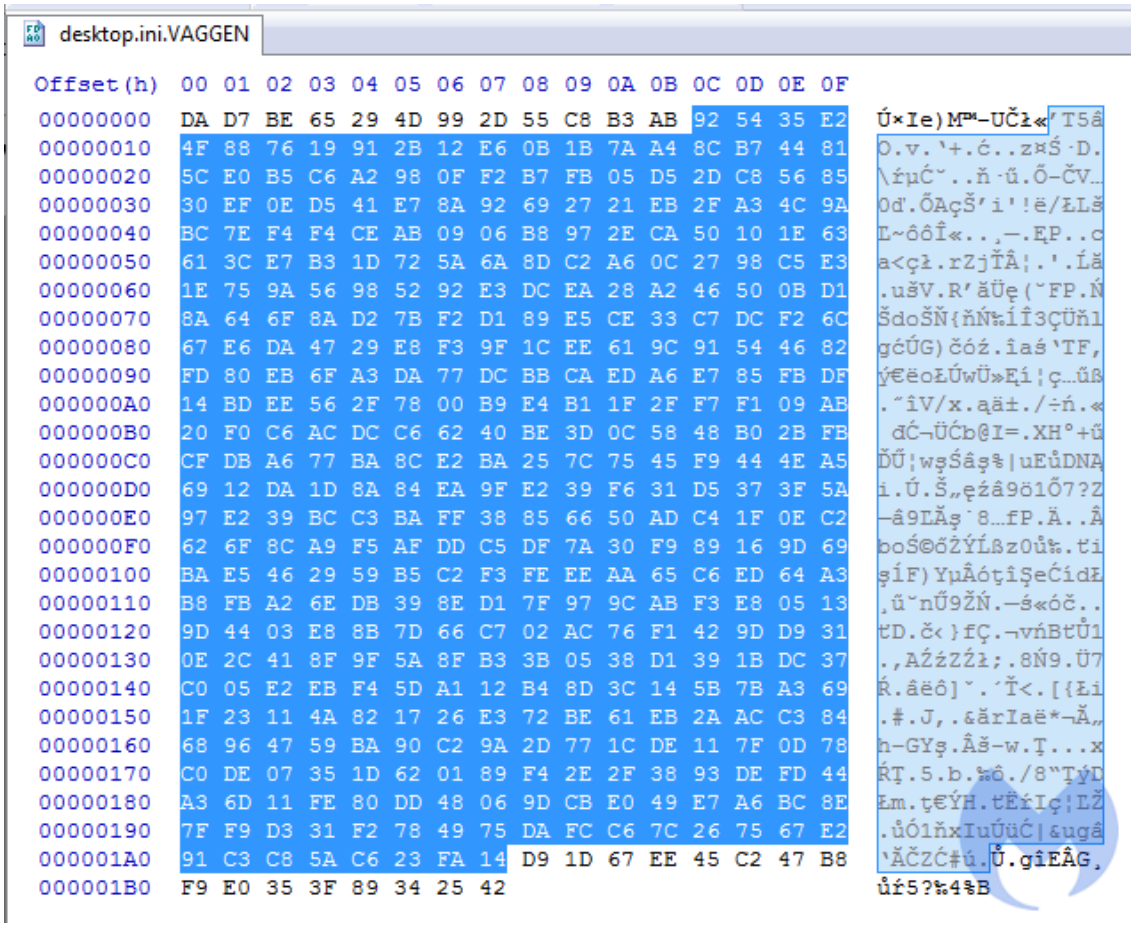
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

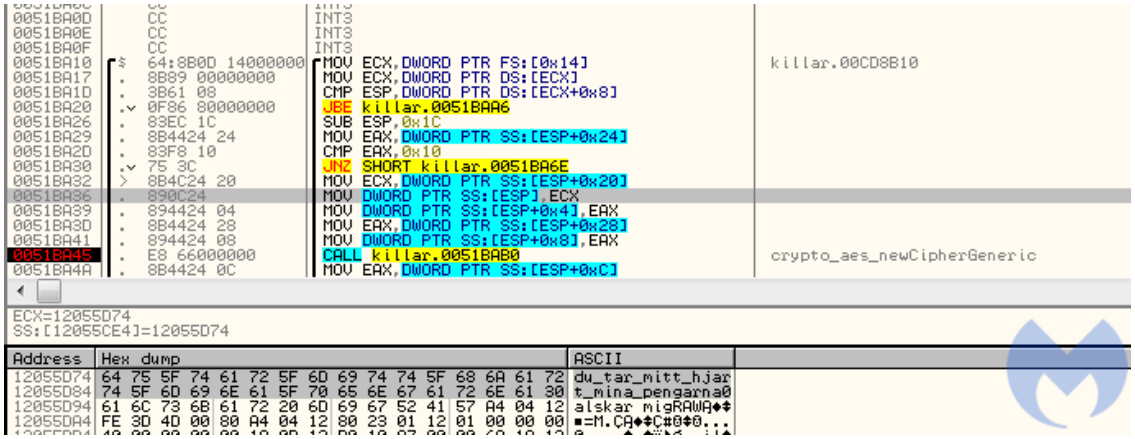


The content of the output file (with .VAGGEN extension) contains:

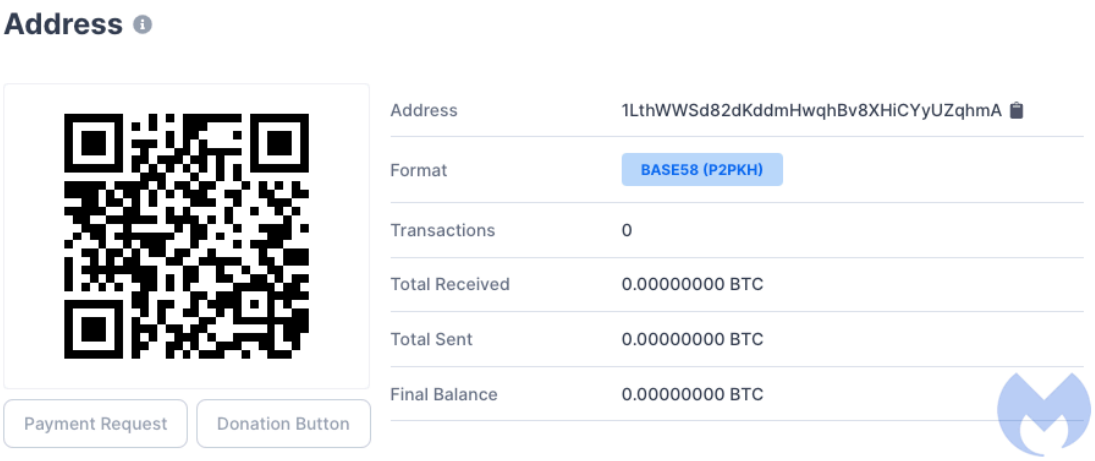
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

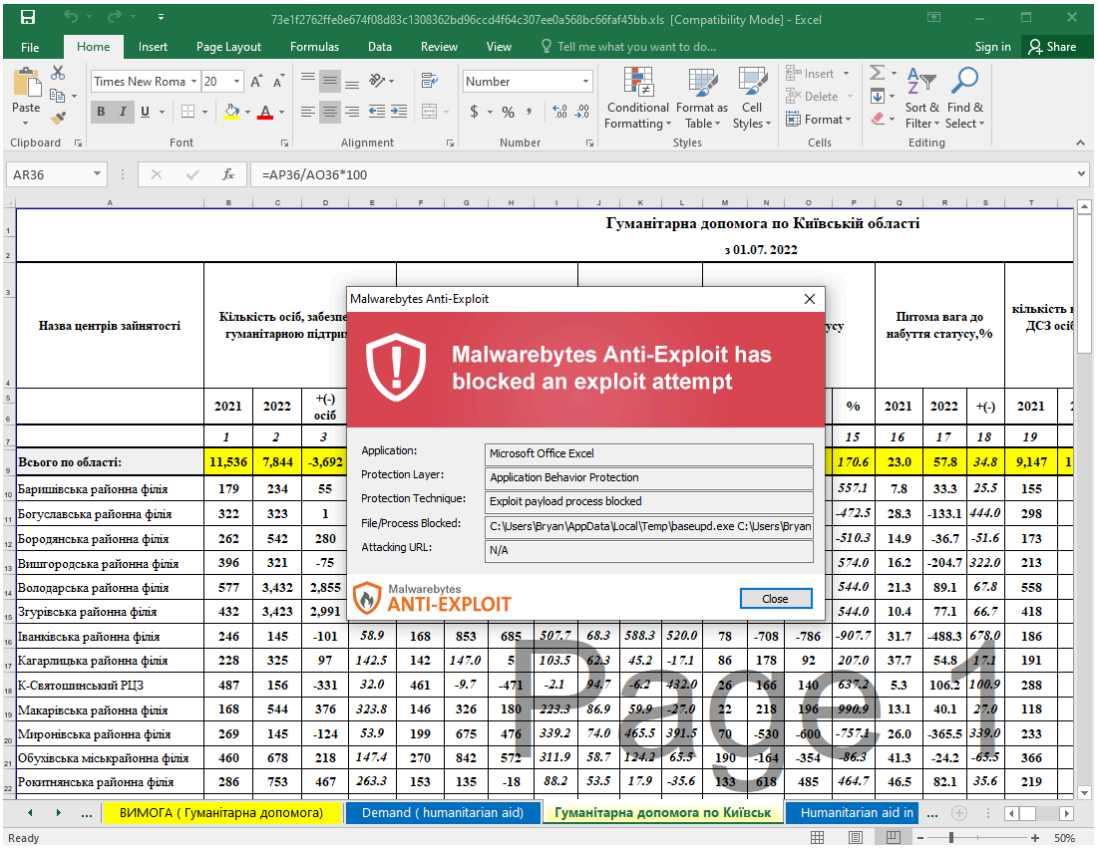
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

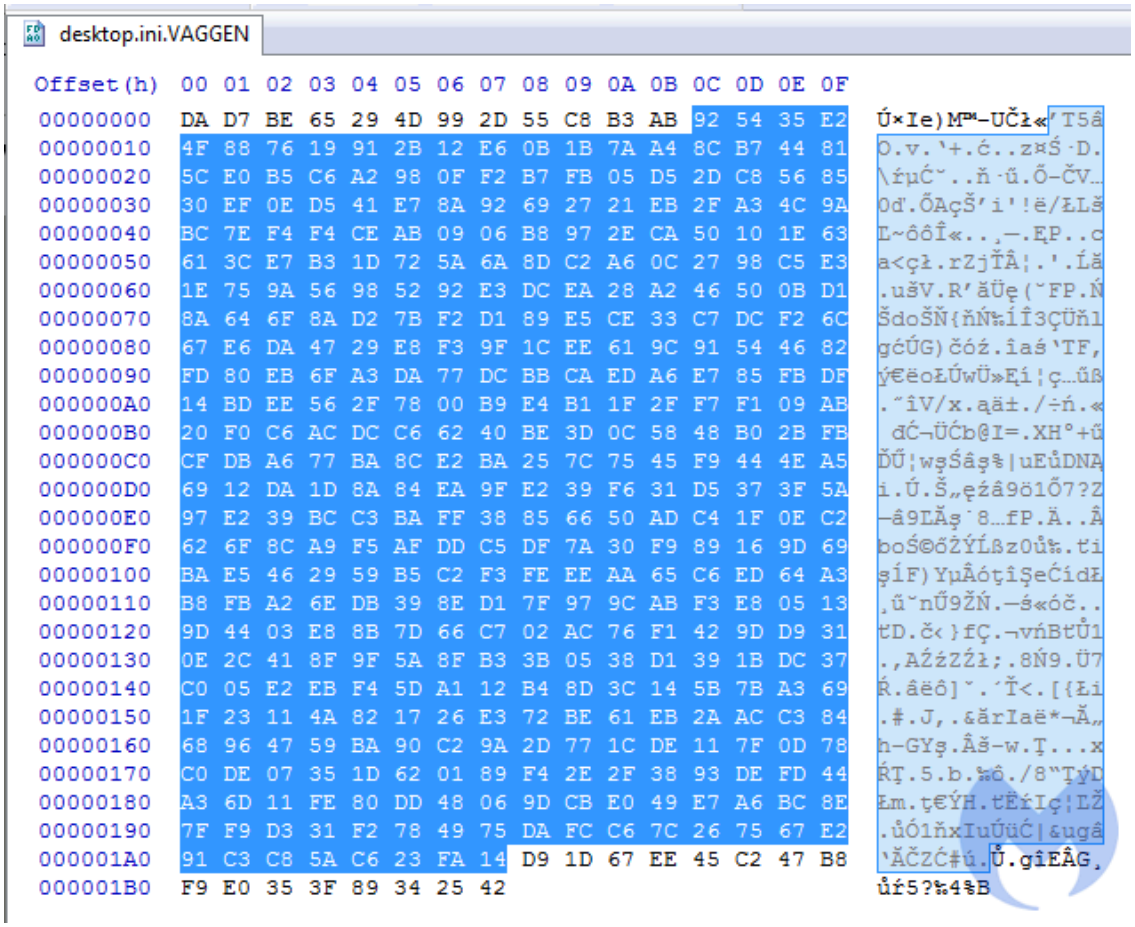
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

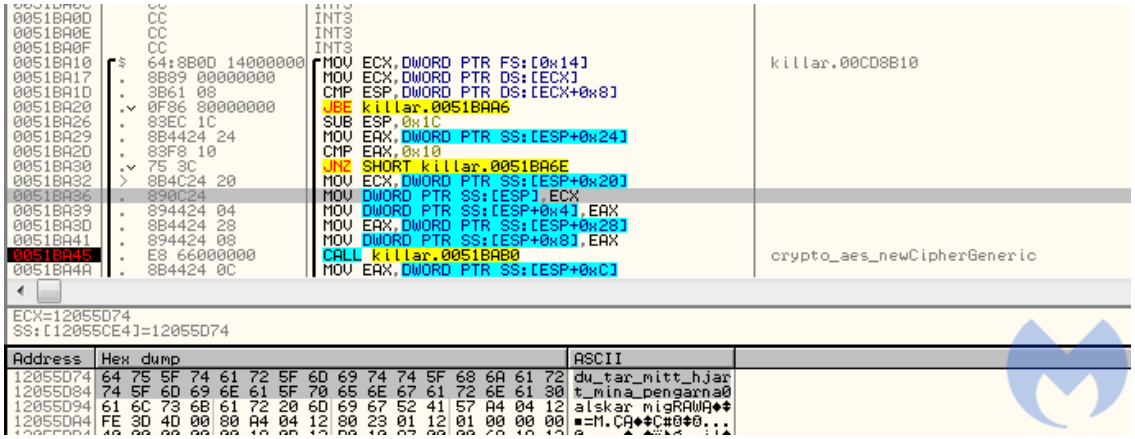


The content of the output file (with .VAGGEN extension) contains:

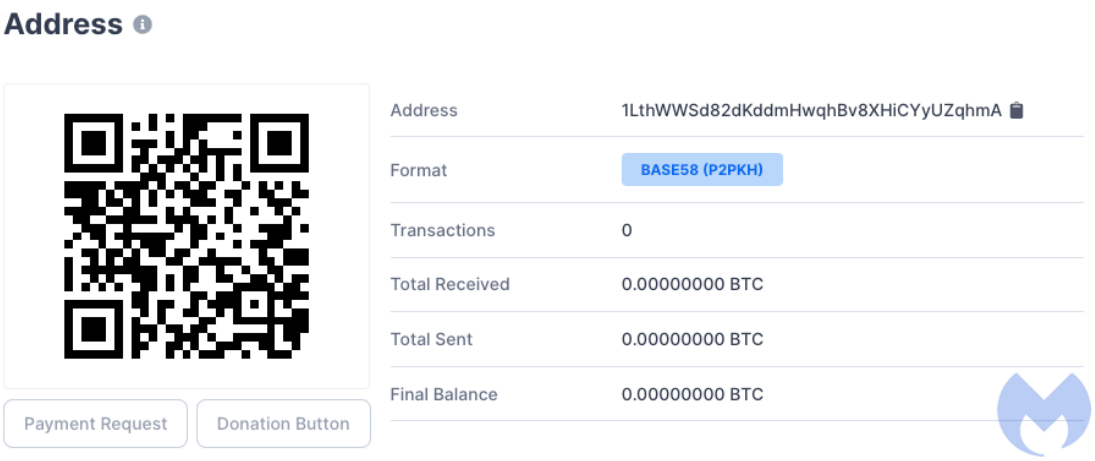
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

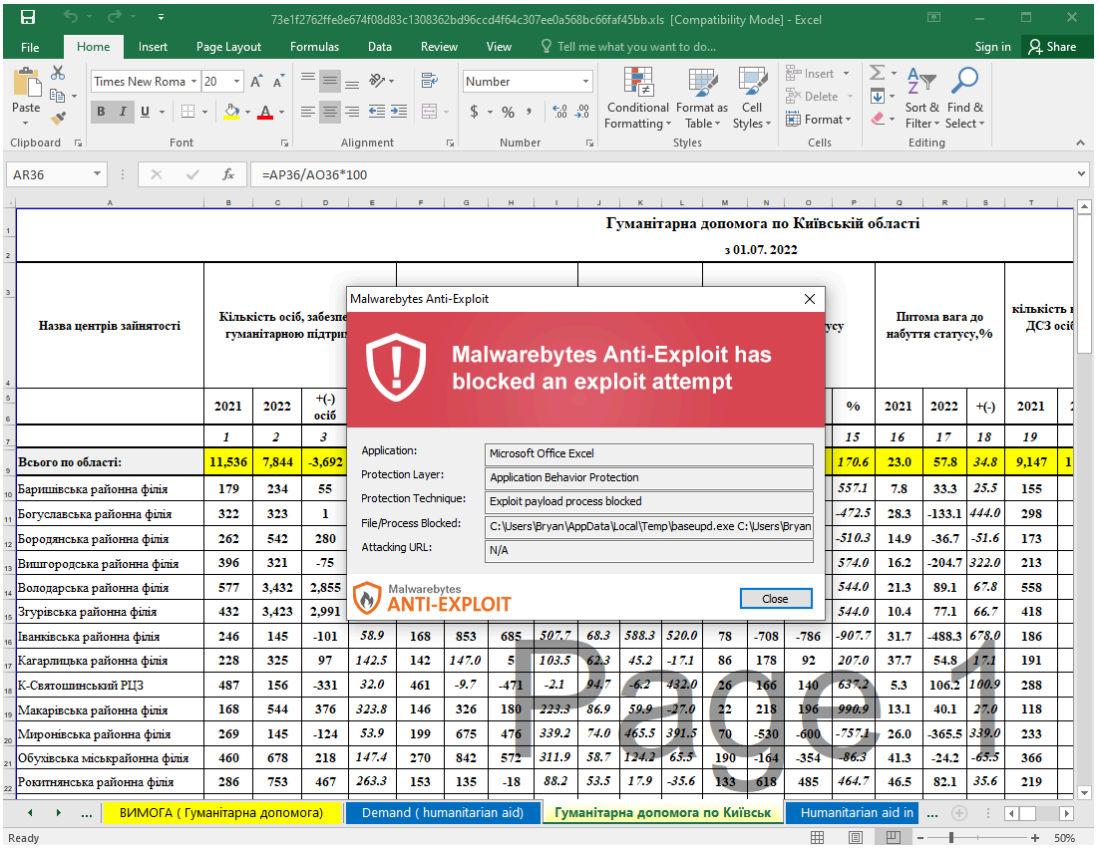
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

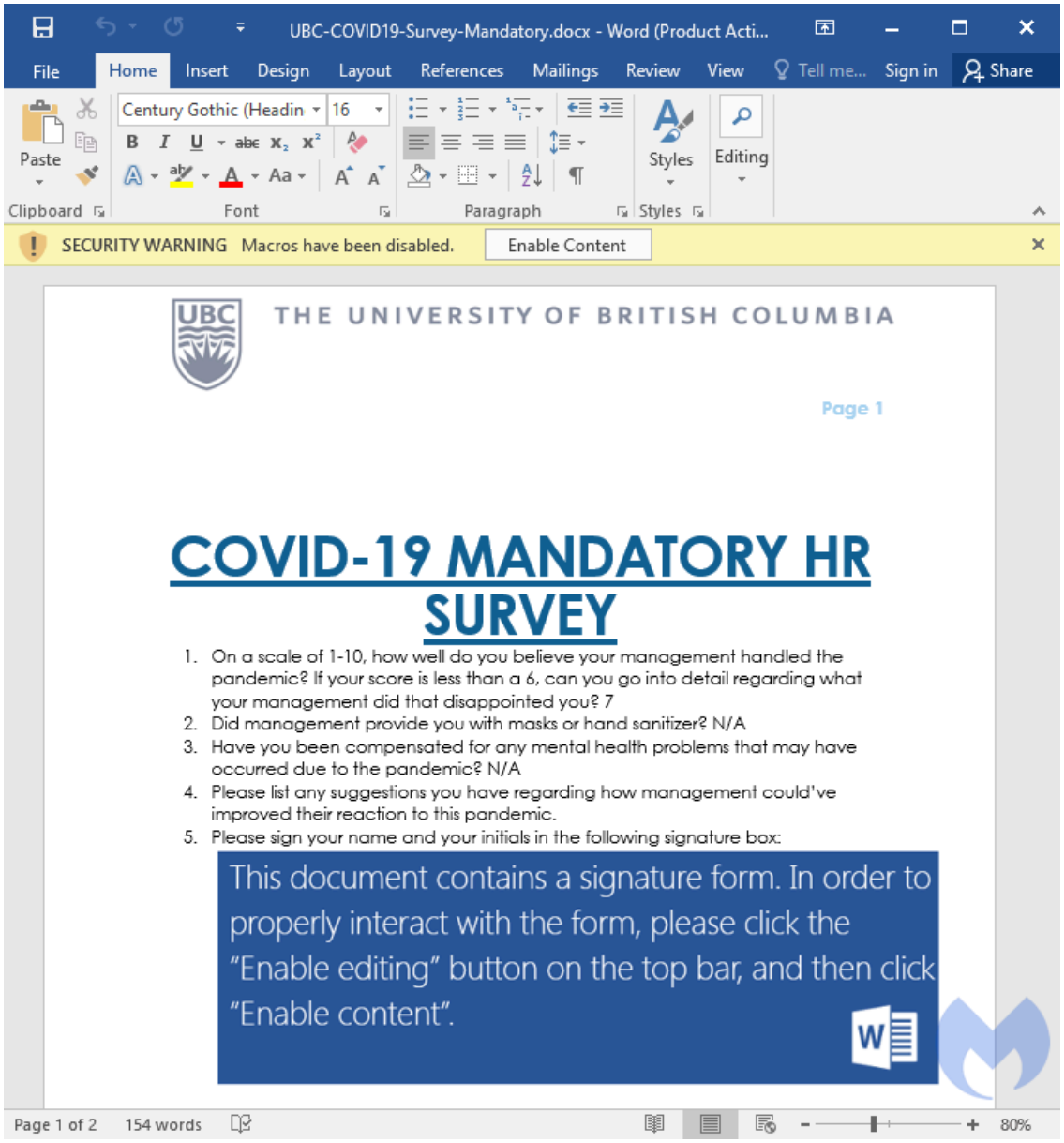
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

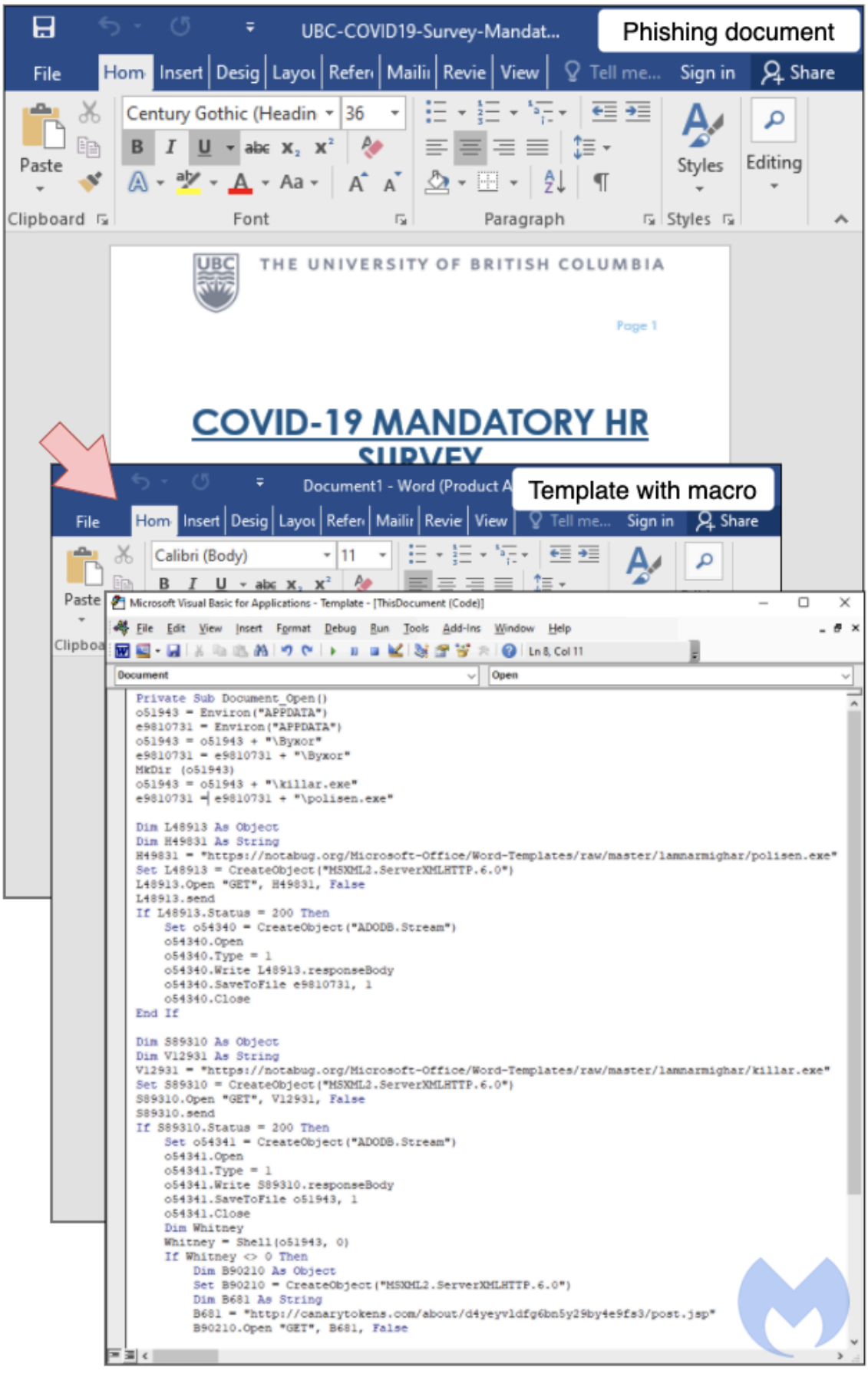
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

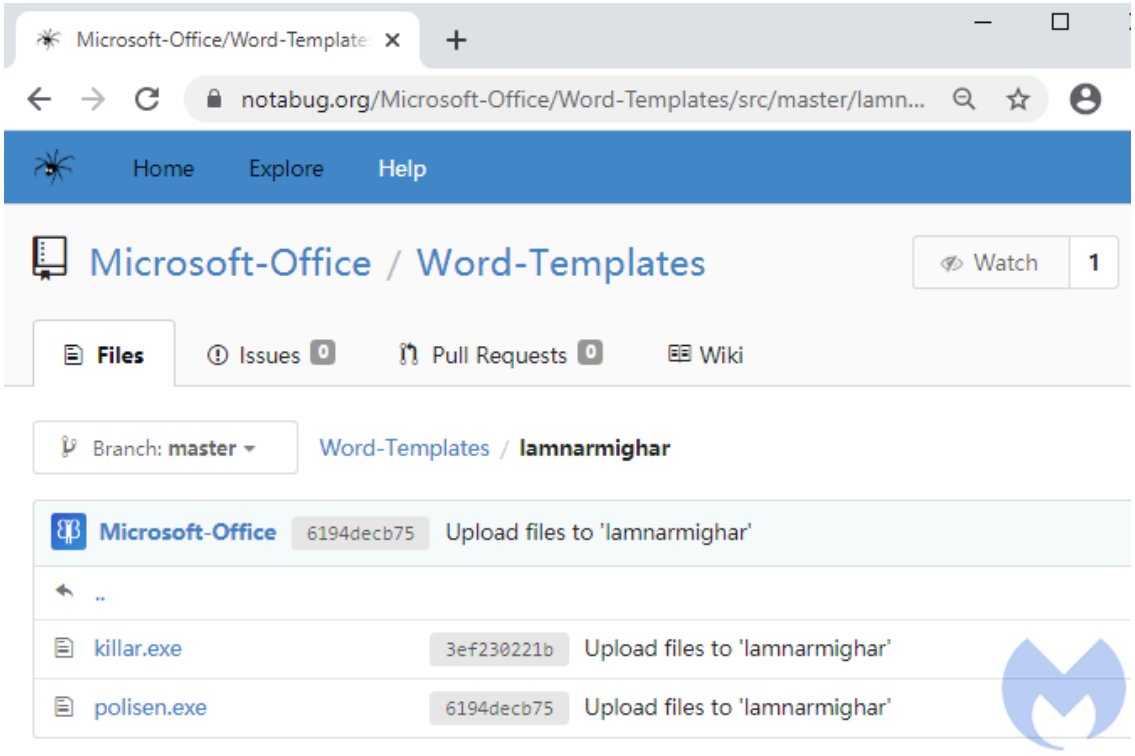
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



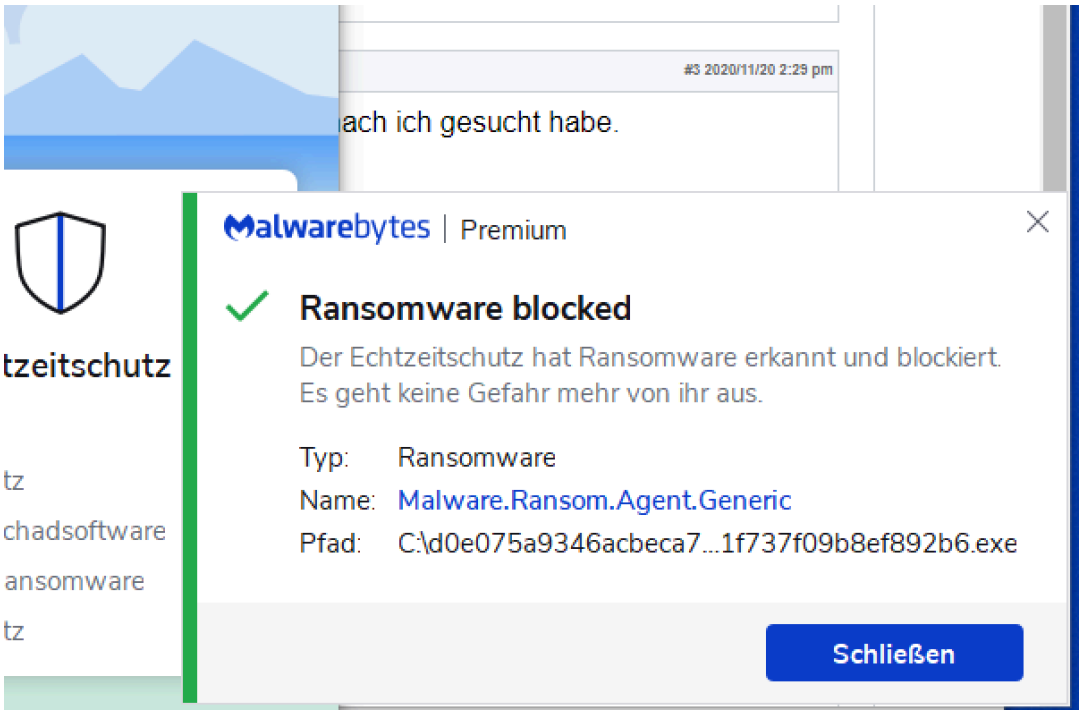
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

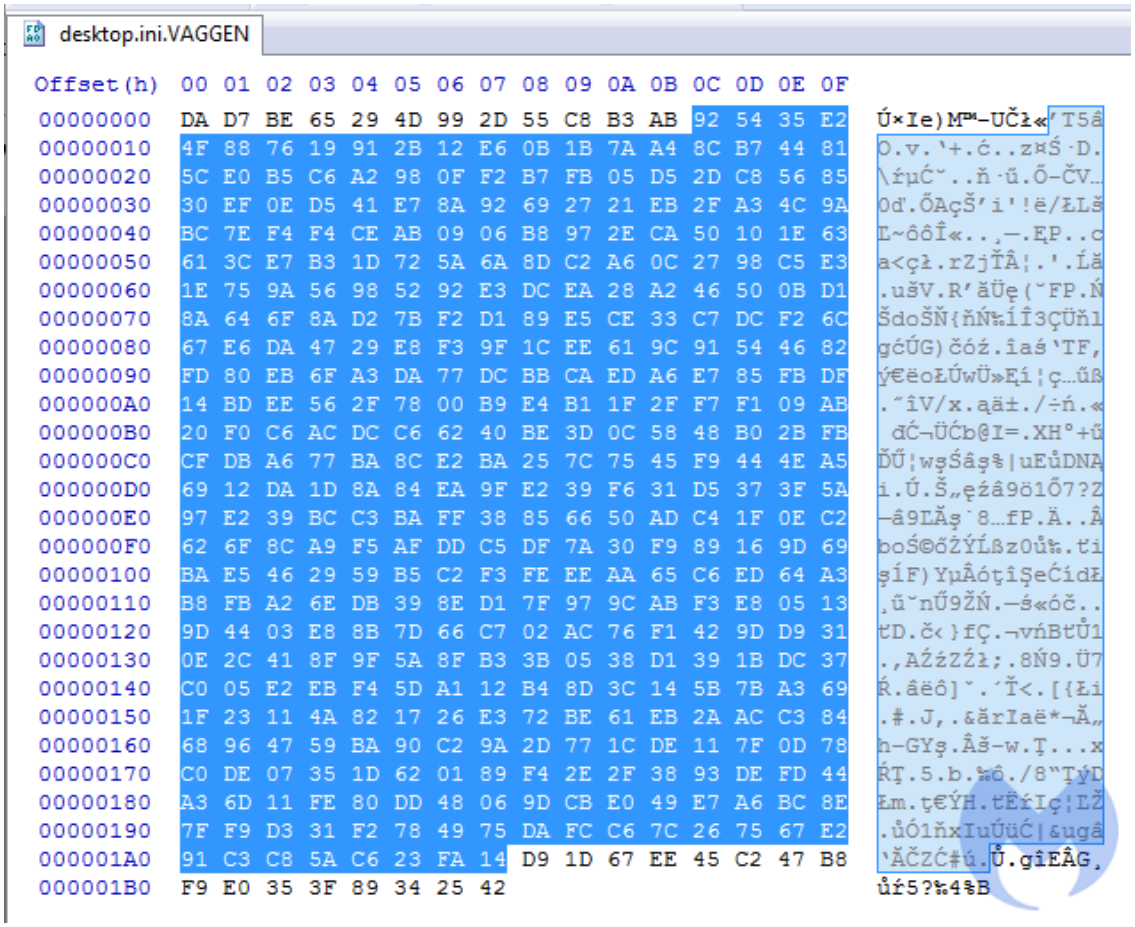
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

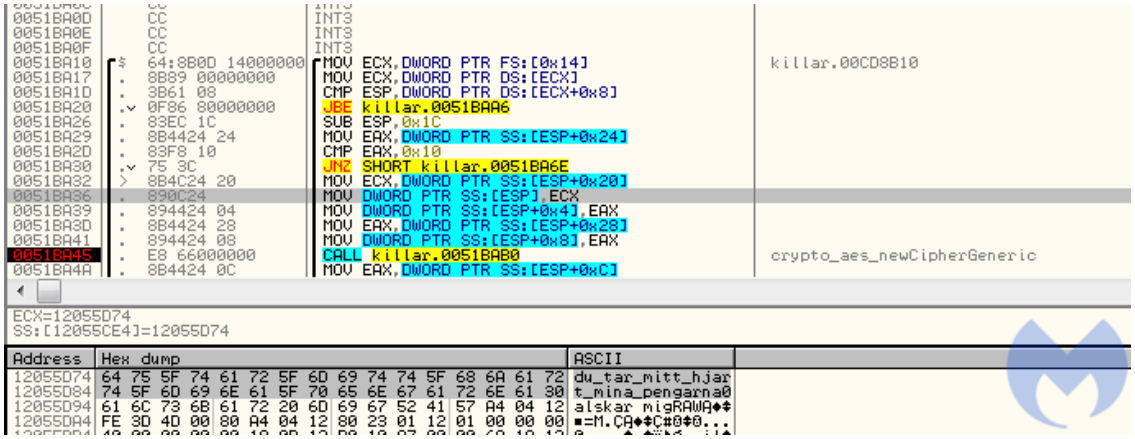


The content of the output file (with .VAGGEN extension) contains:

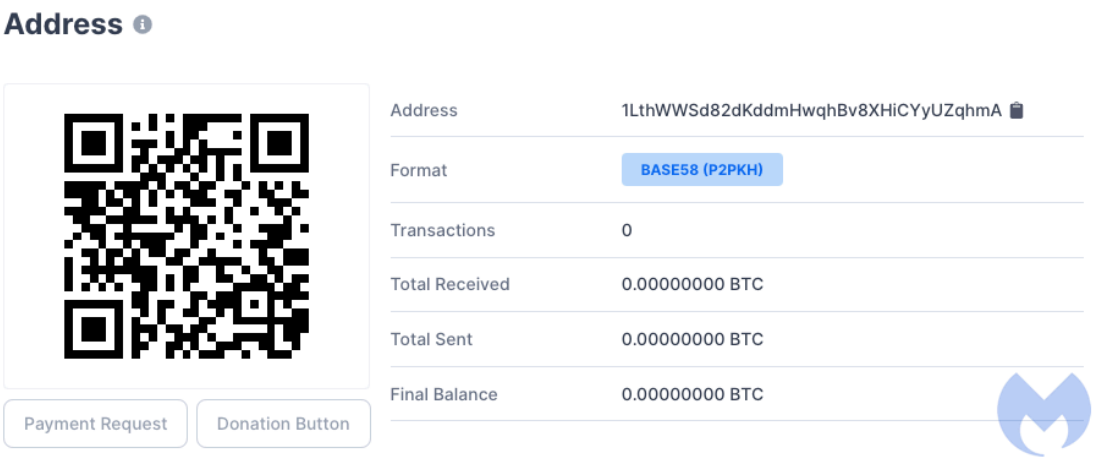
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

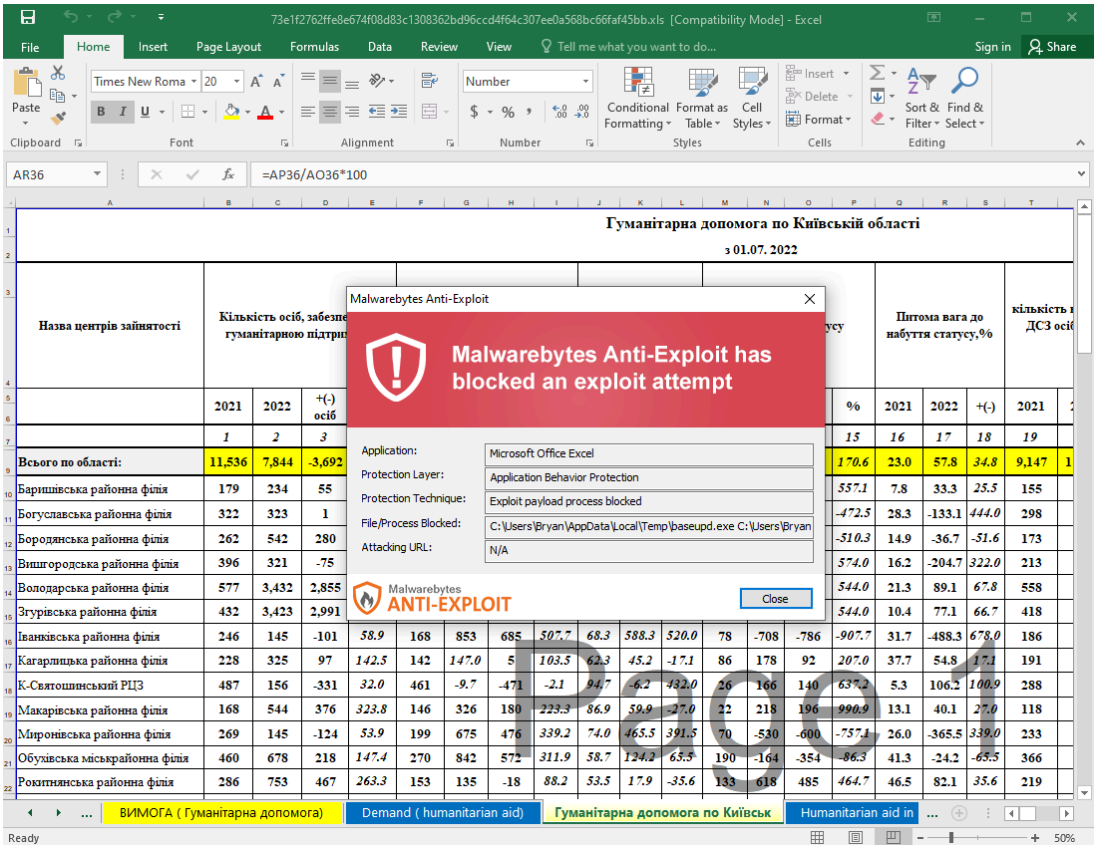
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
46  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
47    runtime_morestack_noctxt();
48  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
49  ((void (*)(void))loc_45CDA0)();
50  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
51  v38 = v4;
52  v30 = v5;
53  v31 = v6;
54  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
55  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
56  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
57  v32 = v10;
58  v11 = *(void (__cdecl **)(int))(v10 + 16);
59  v39 = v12;
60  v11(v12);
61  v29 = v13;
62  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
63  v40 = v15;
64  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
65  v17 = v27;
66  if ( v27 )
67  {
68    if ( v27 )
69      v17 = *(_DWORD *)(v27 + 4);
70    v43 = v17;
71    v44 = v28;
72    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
73  }
```

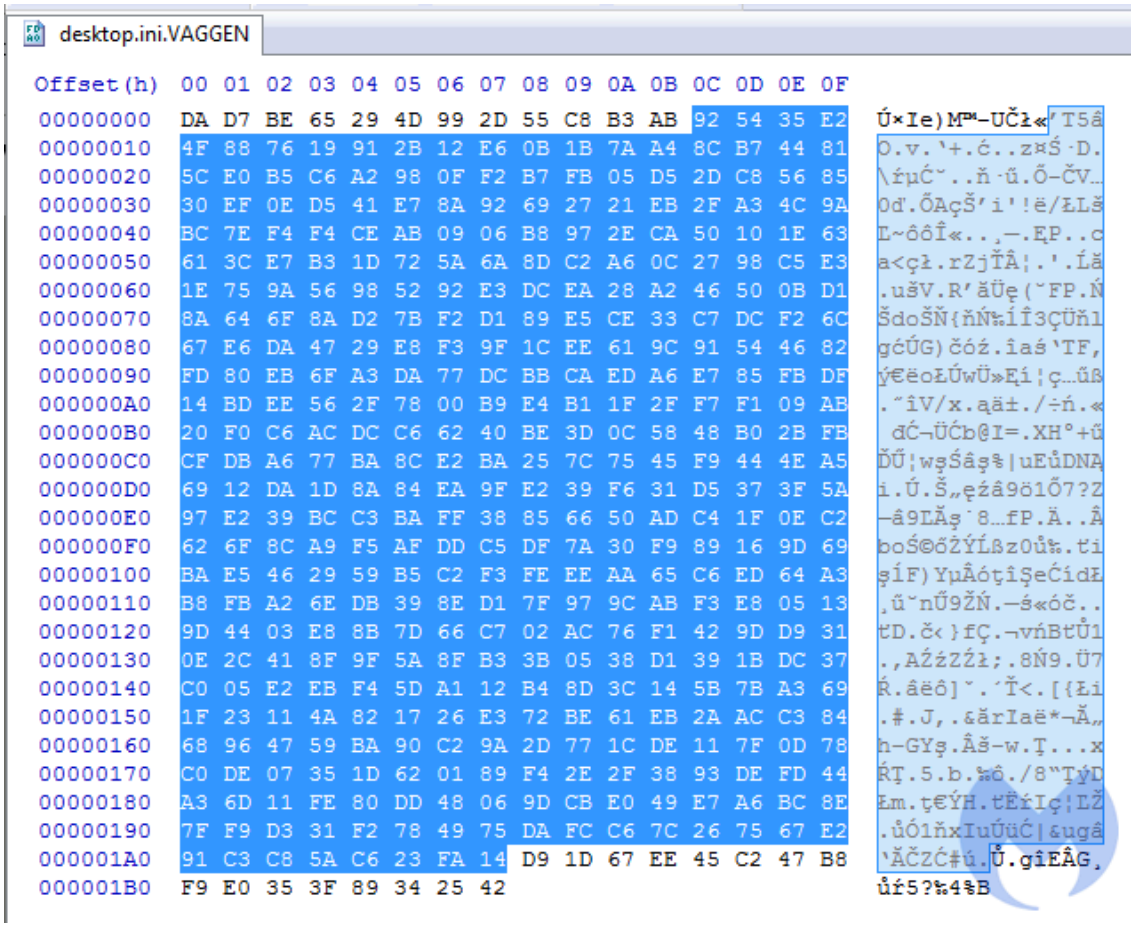
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



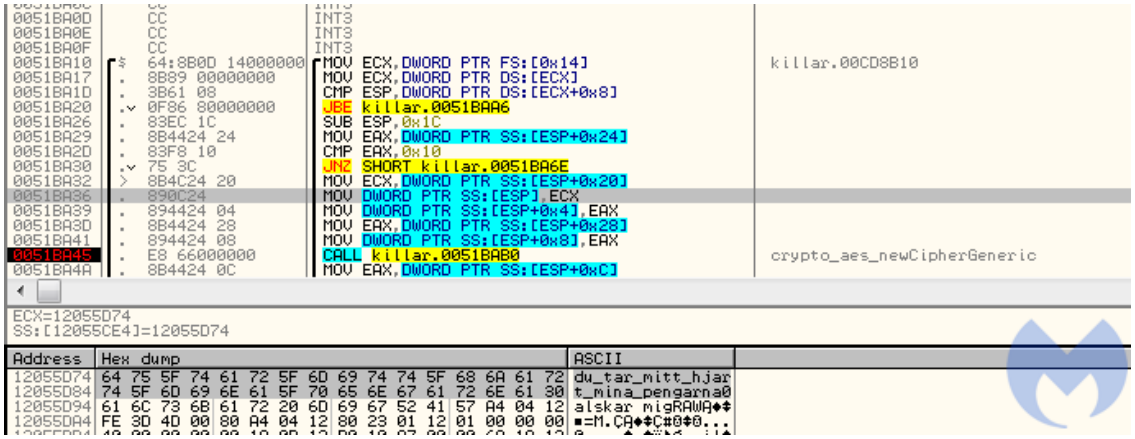
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

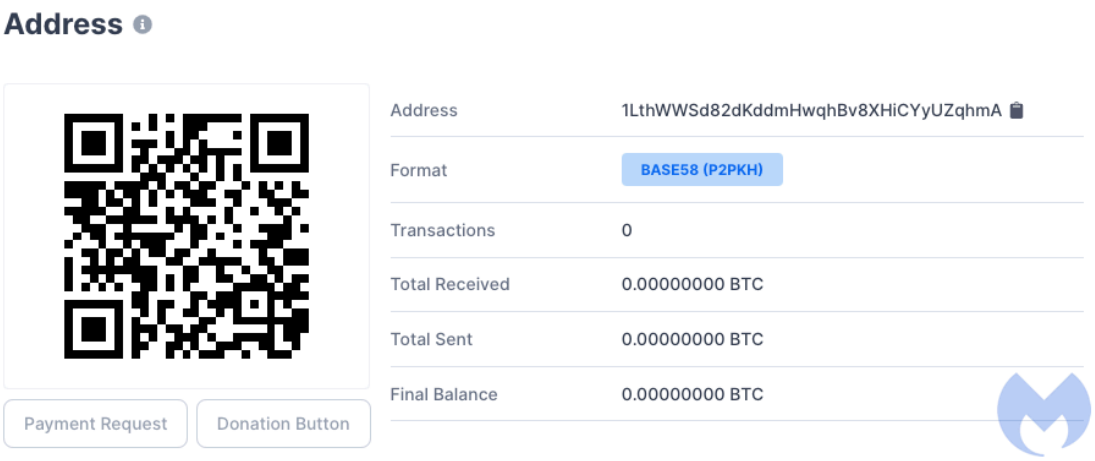
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

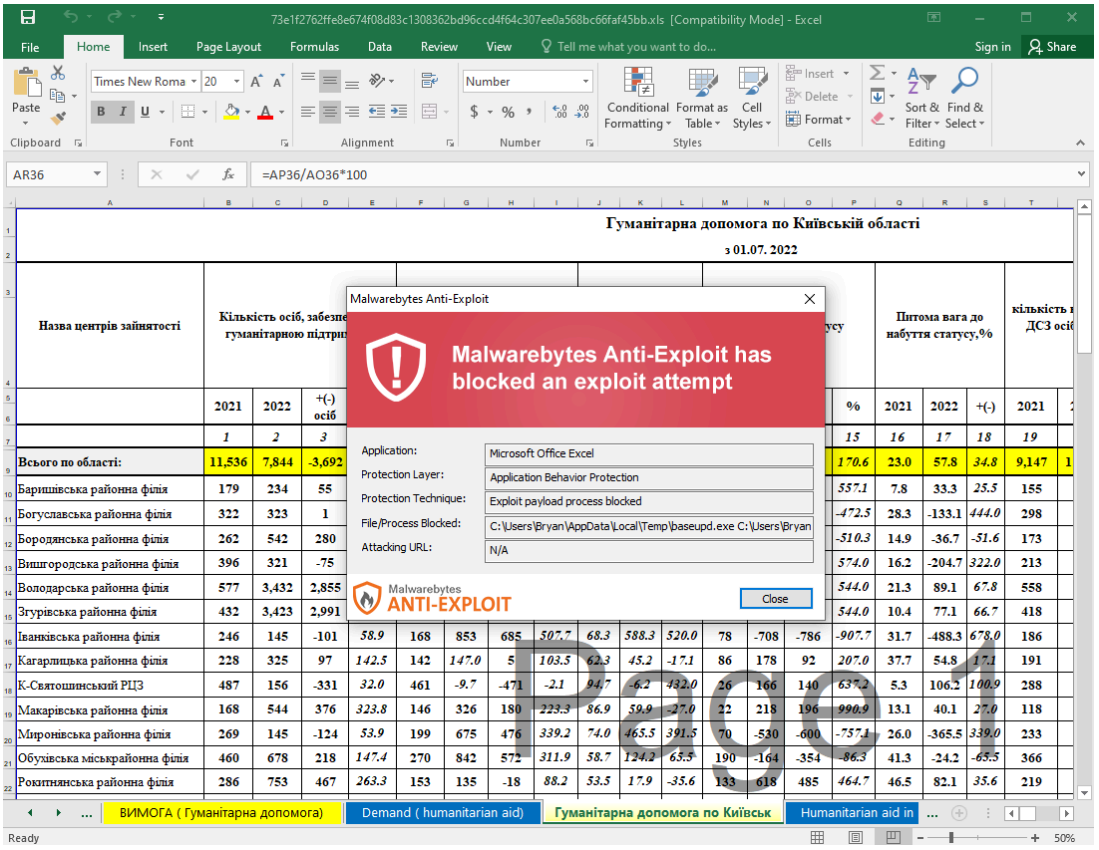
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

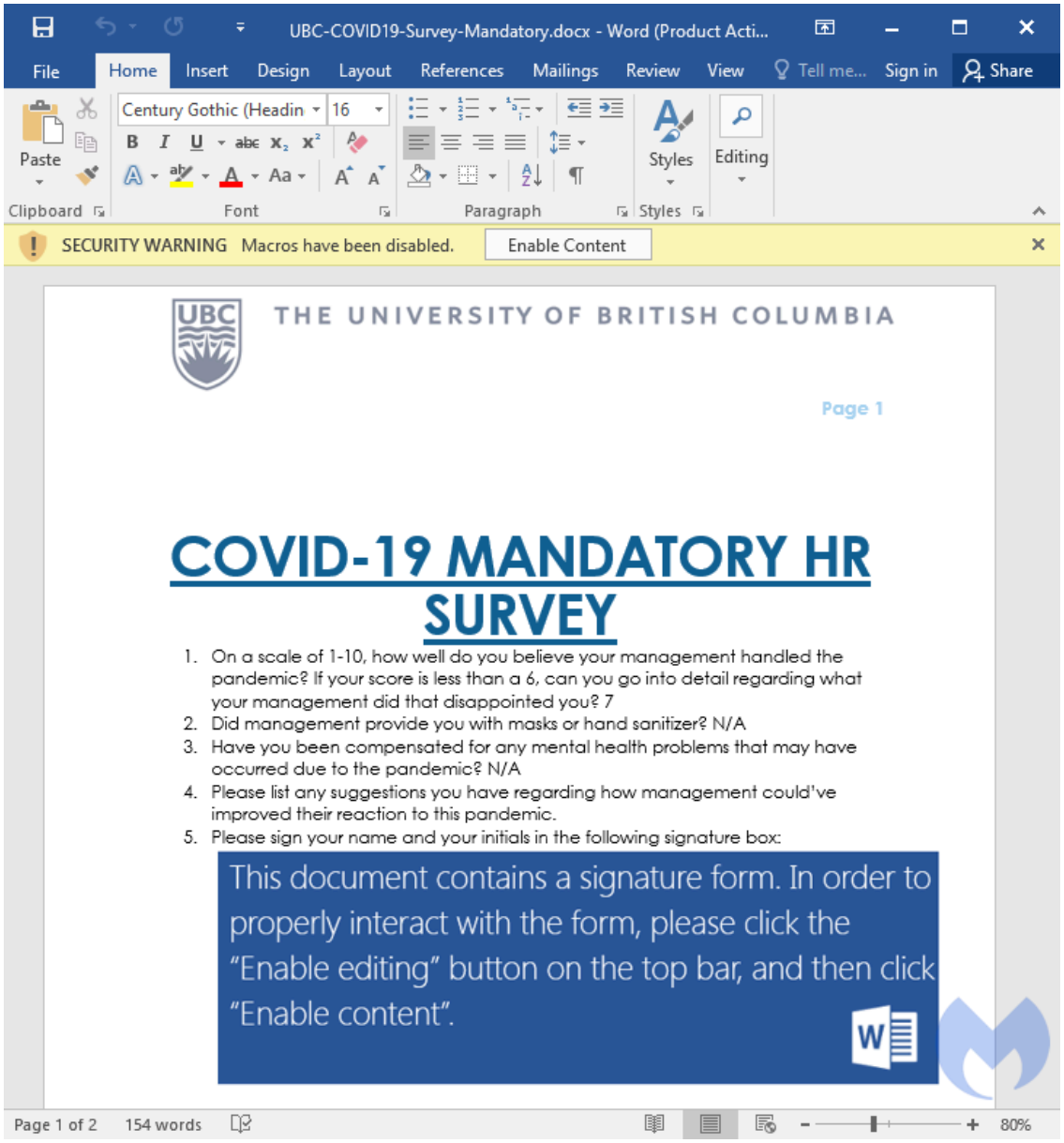
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

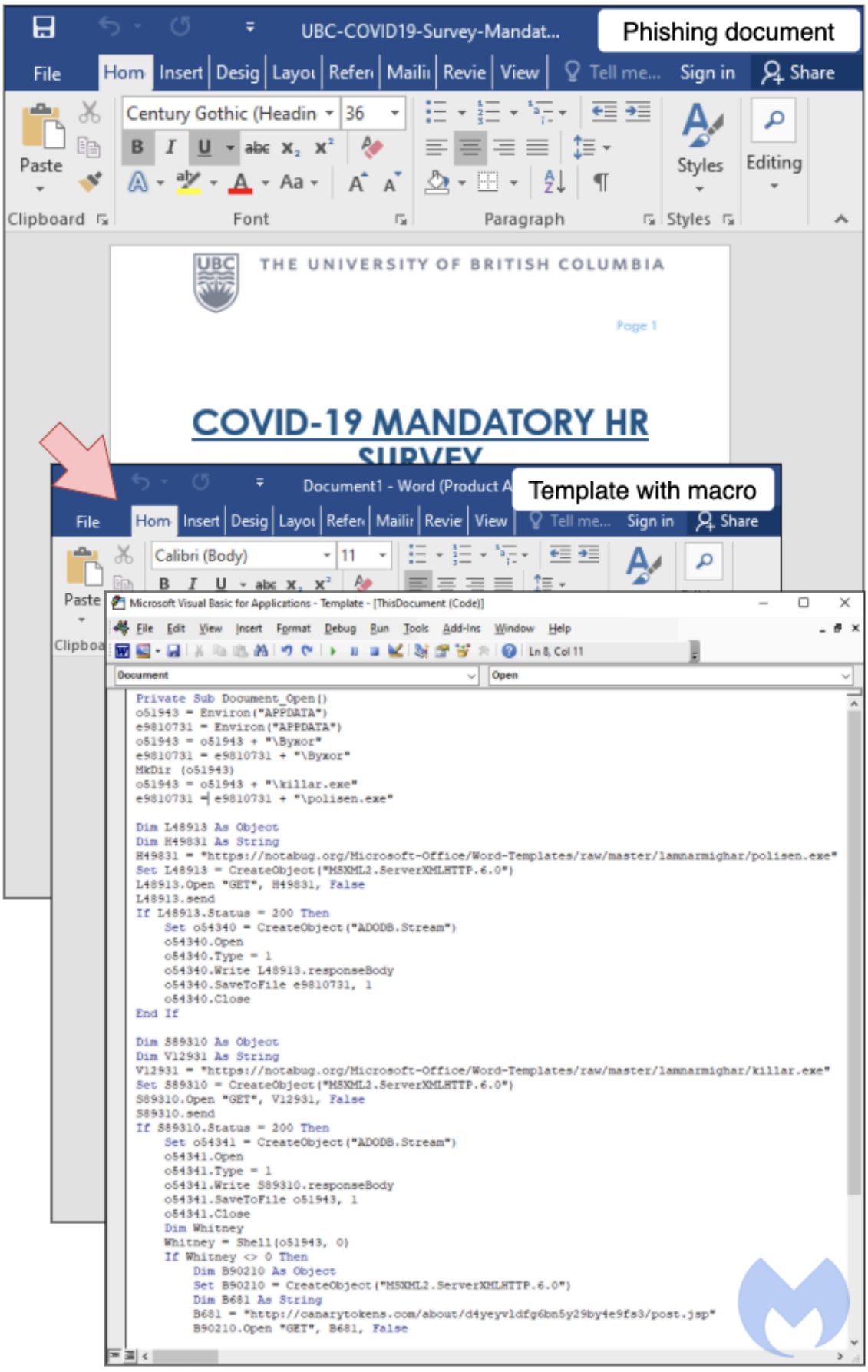
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

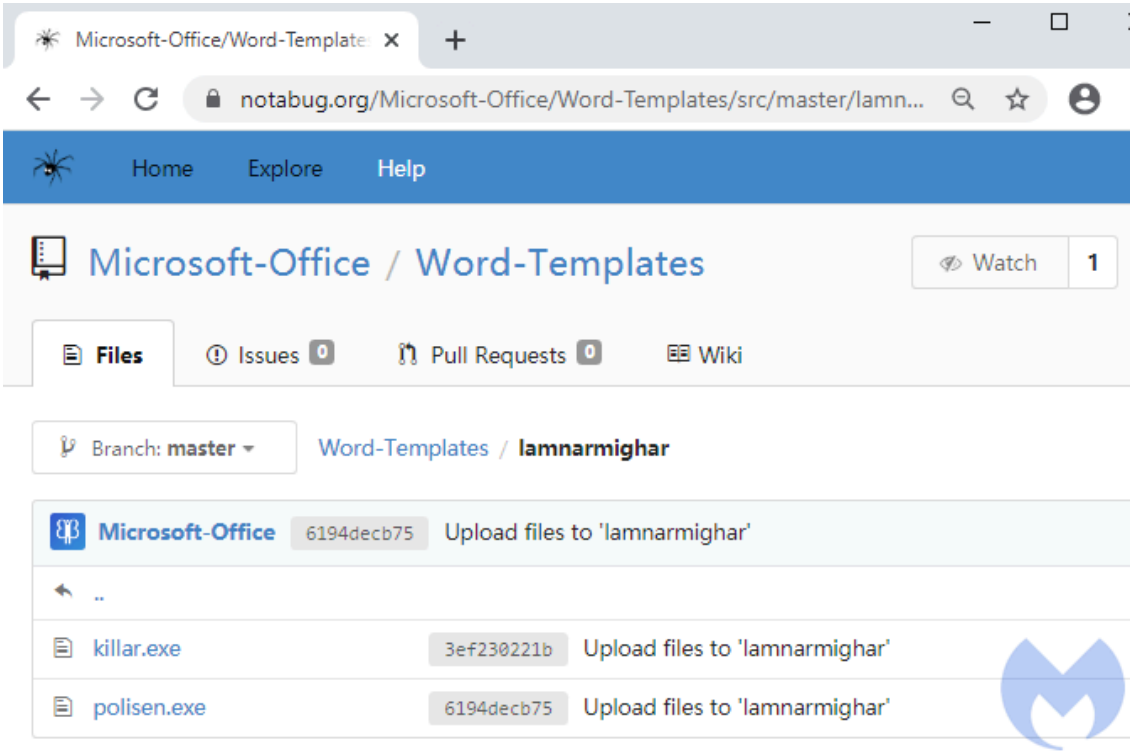
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



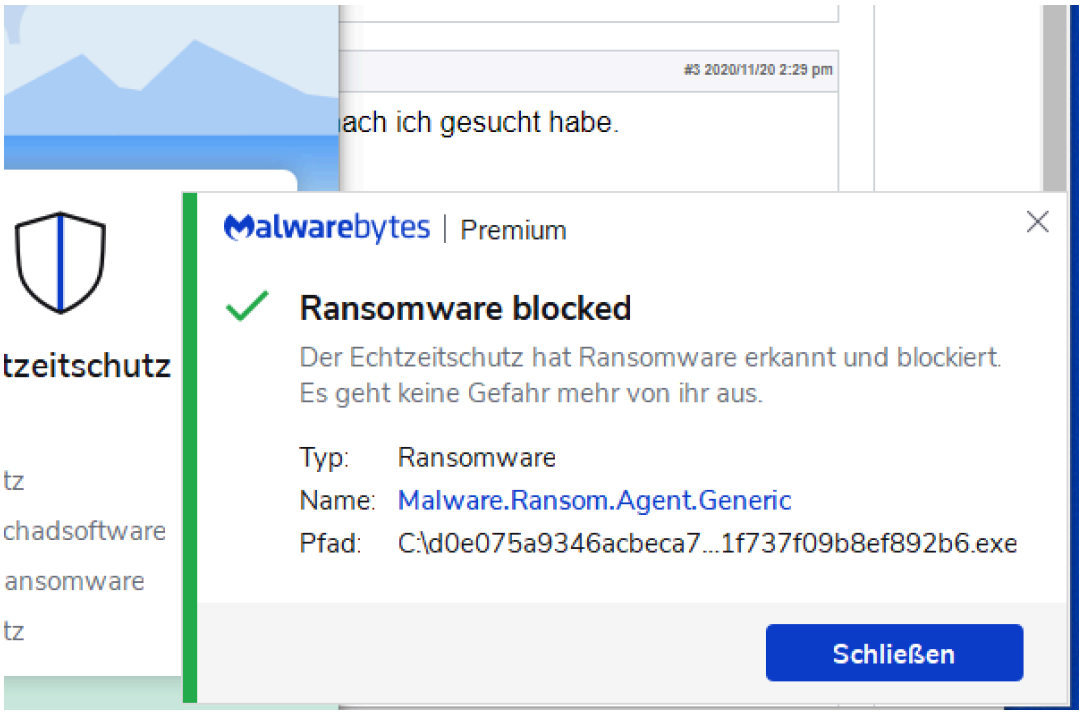
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

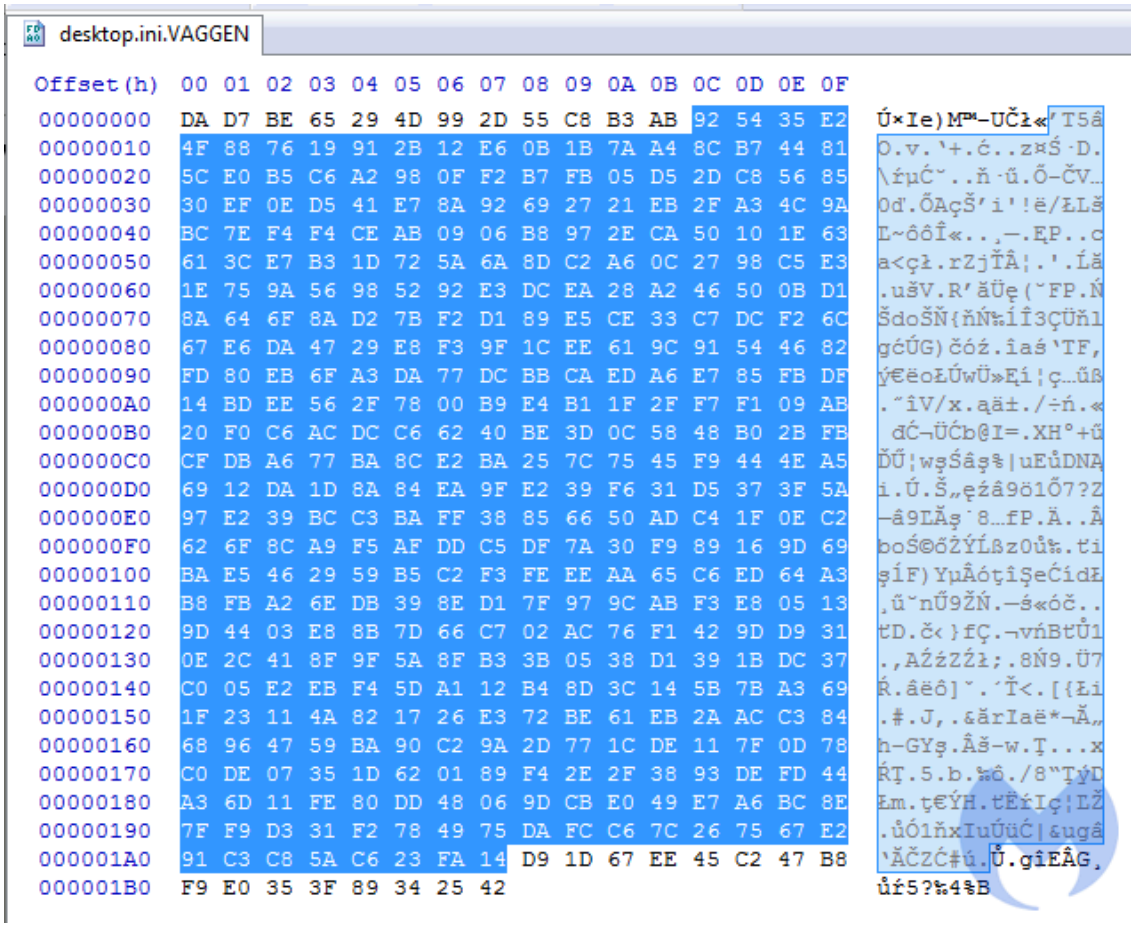
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

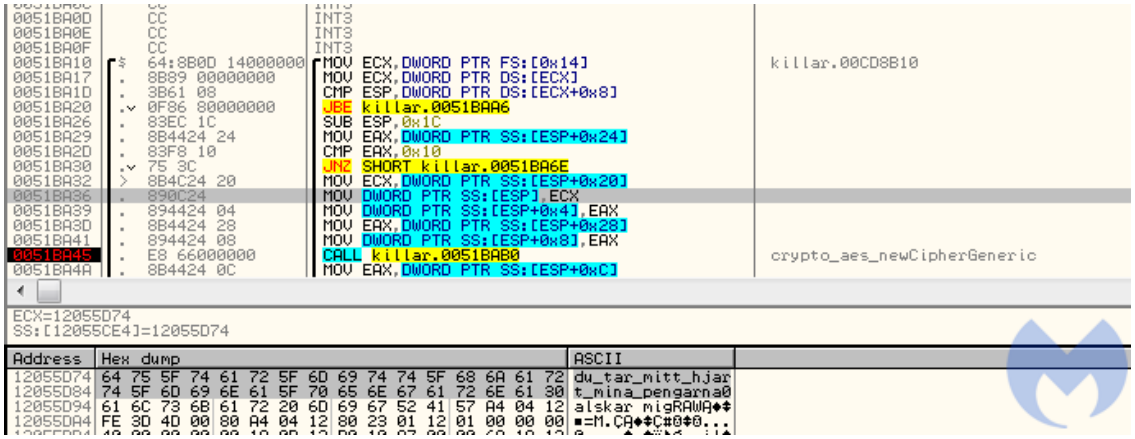


The content of the output file (with .VAGGEN extension) contains:

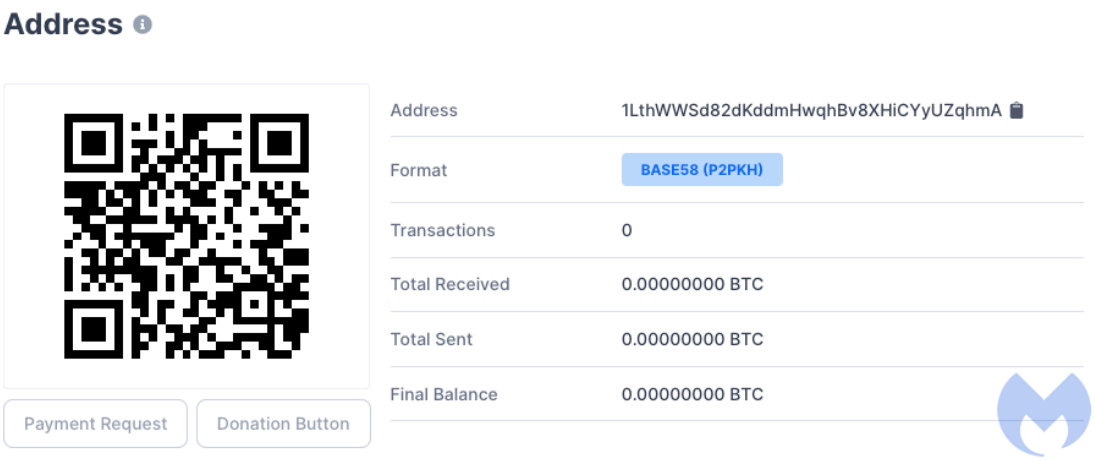
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

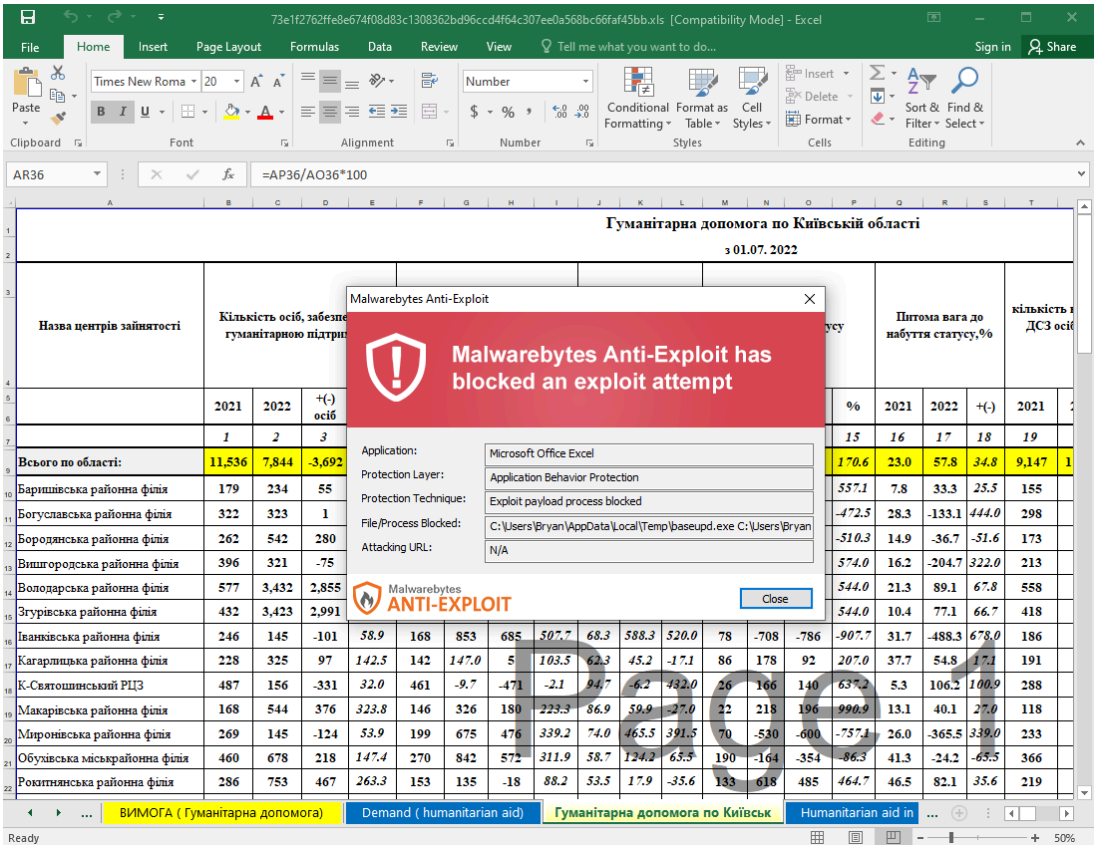
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

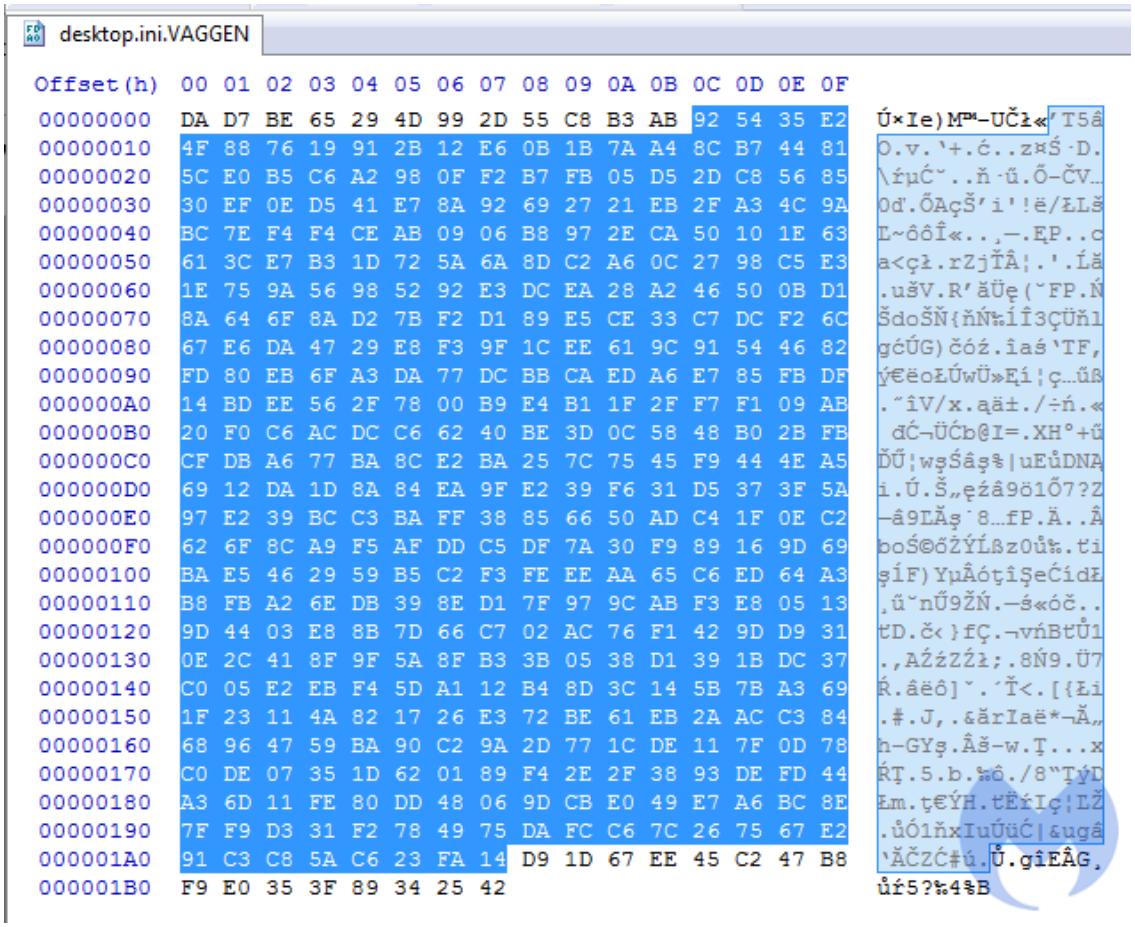
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

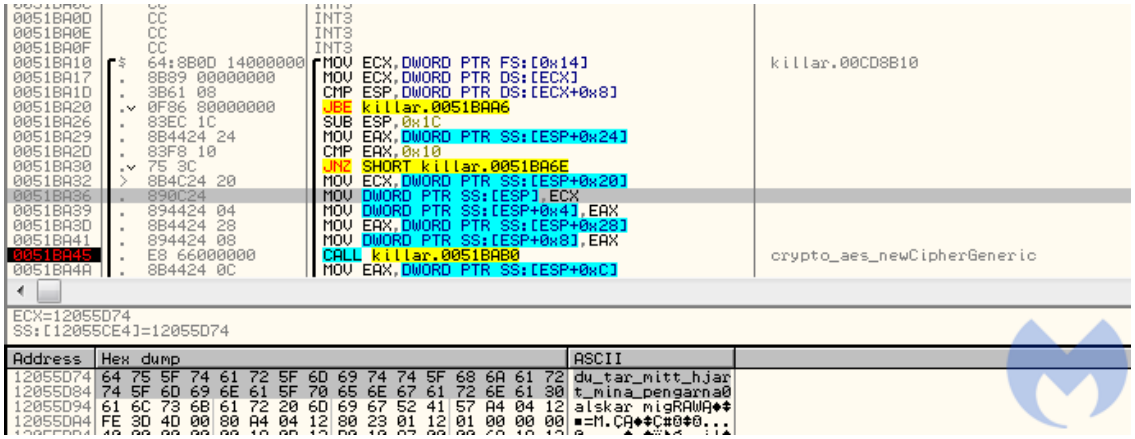


The content of the output file (with .VAGGEN extension) contains:

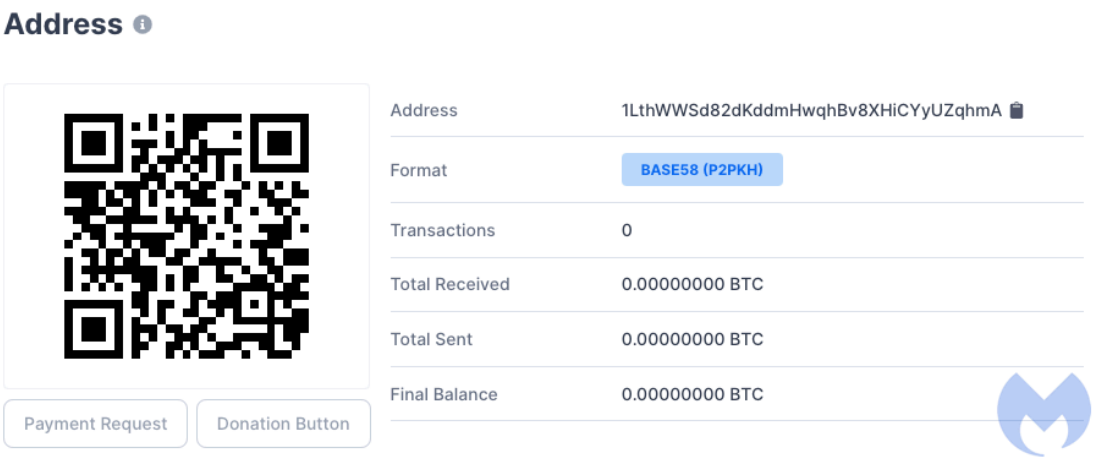
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

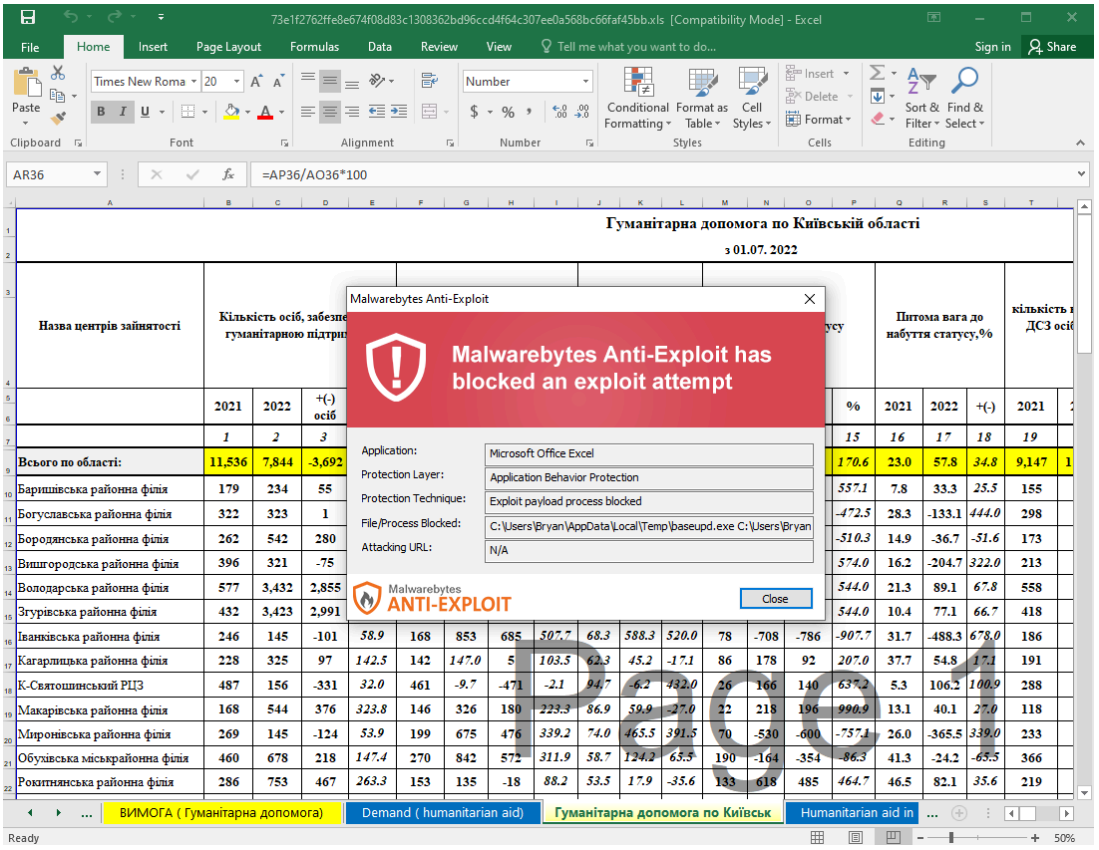
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

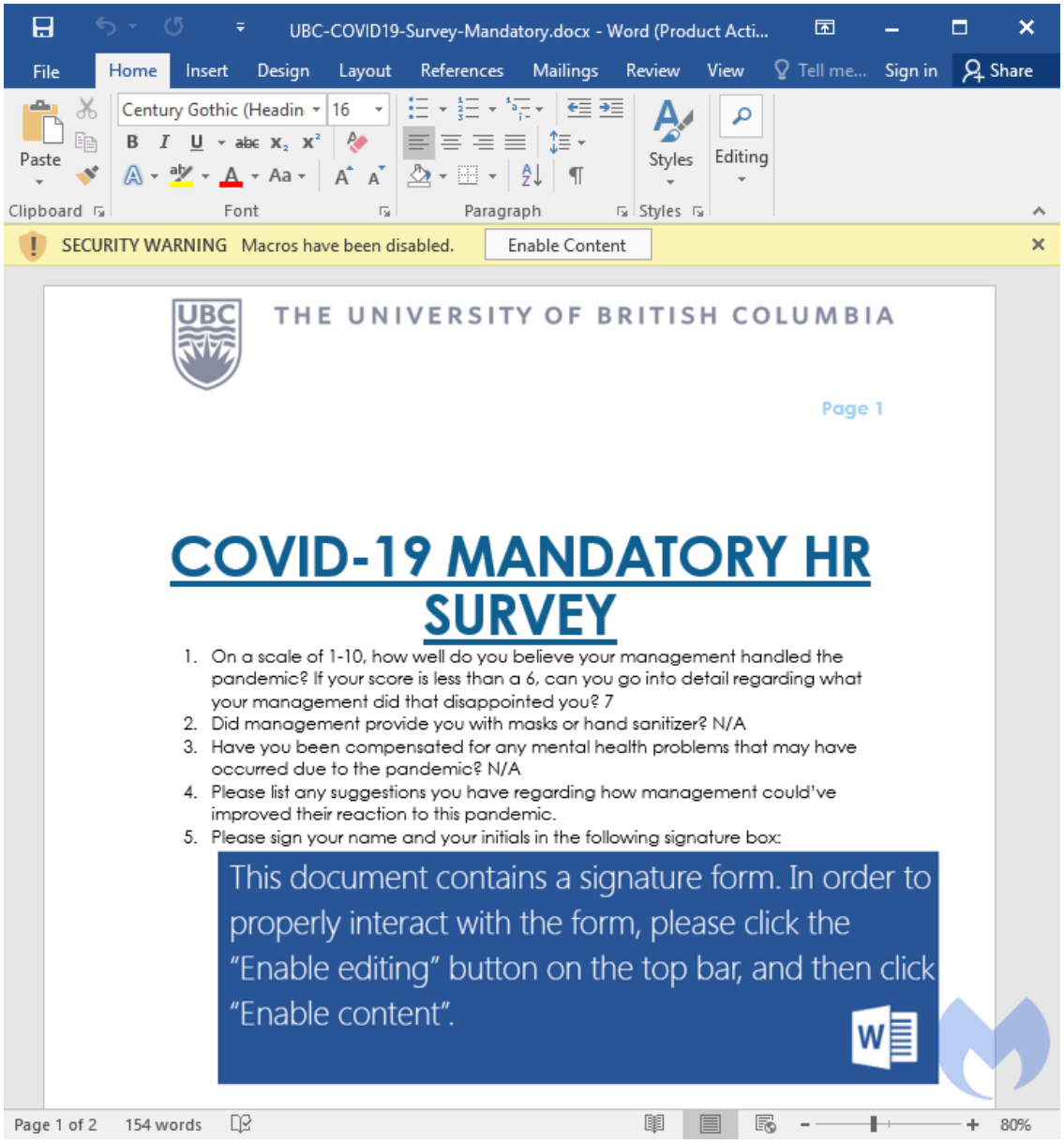
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

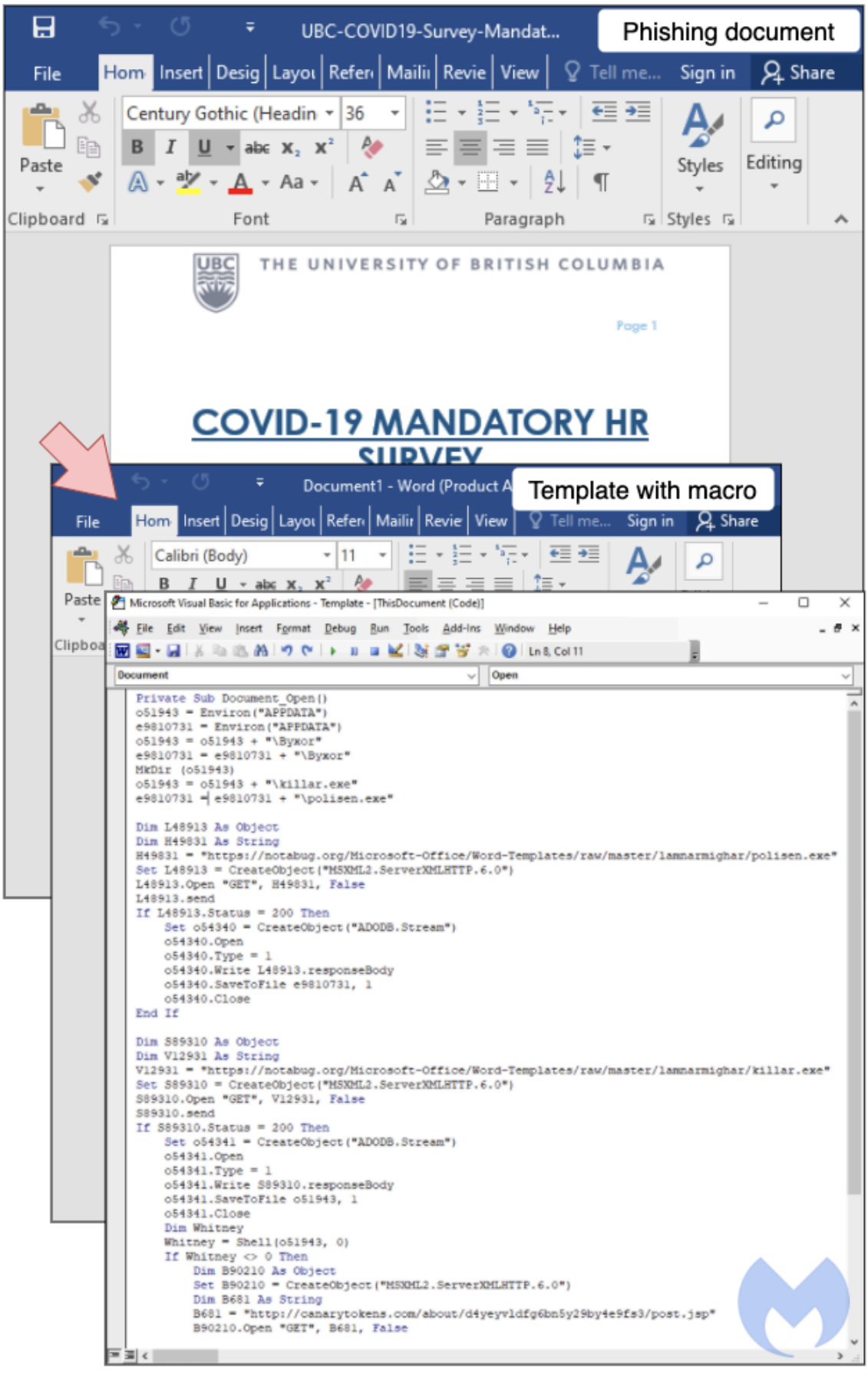
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

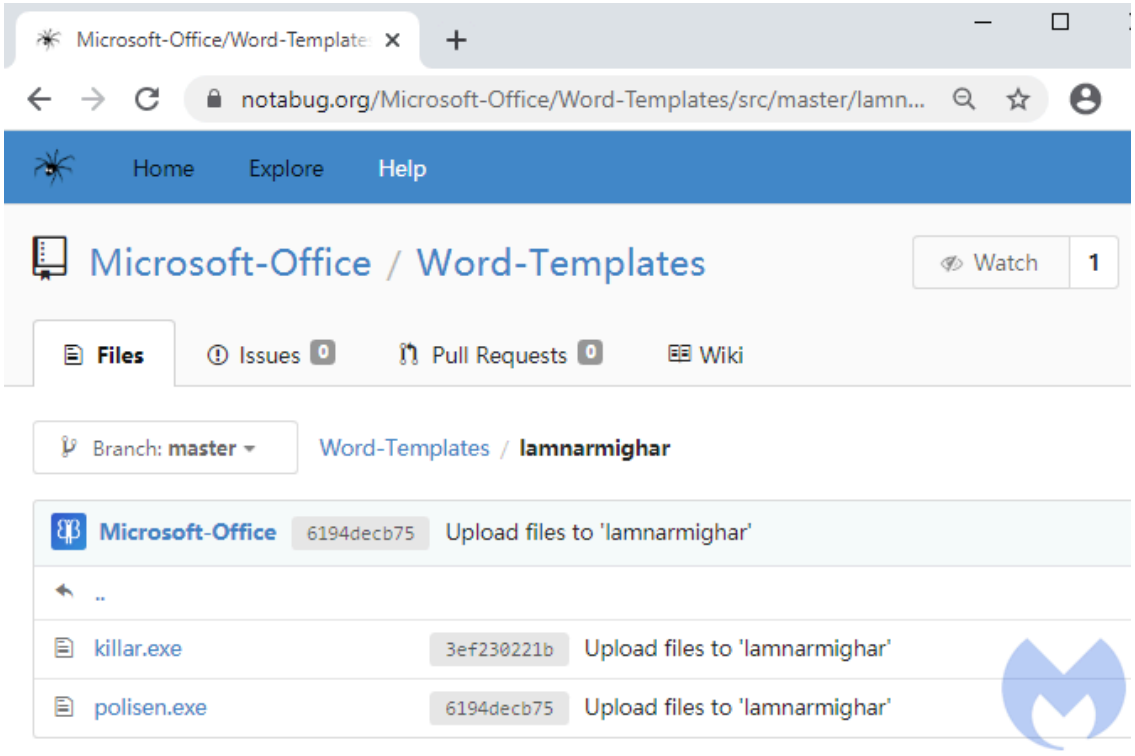
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



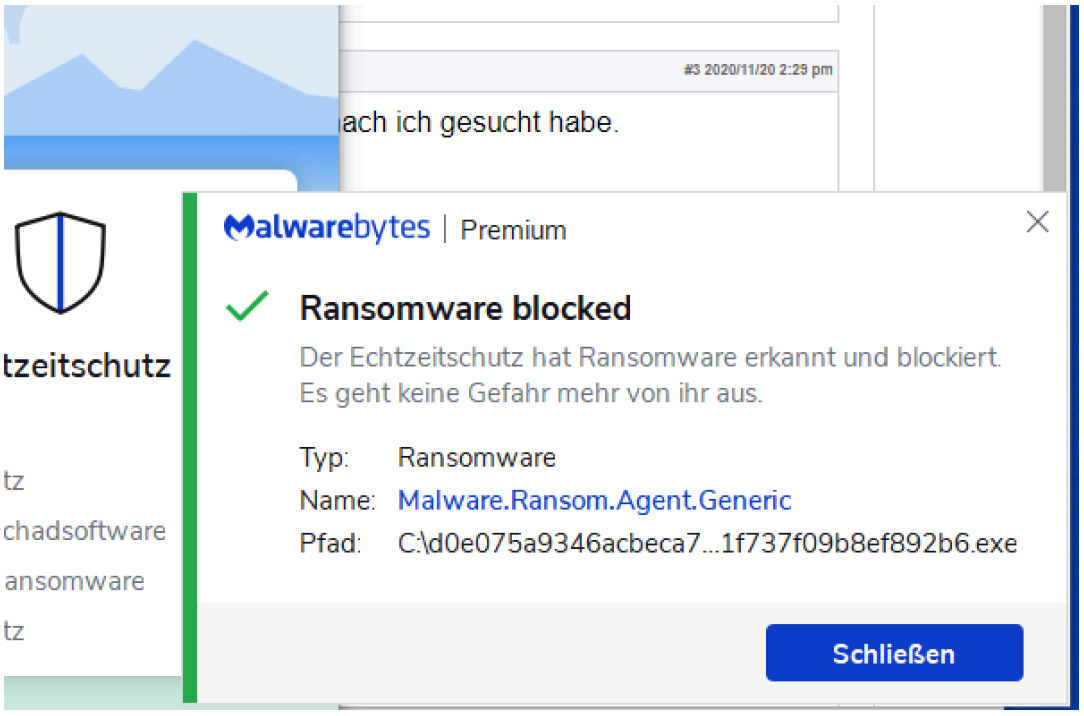
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

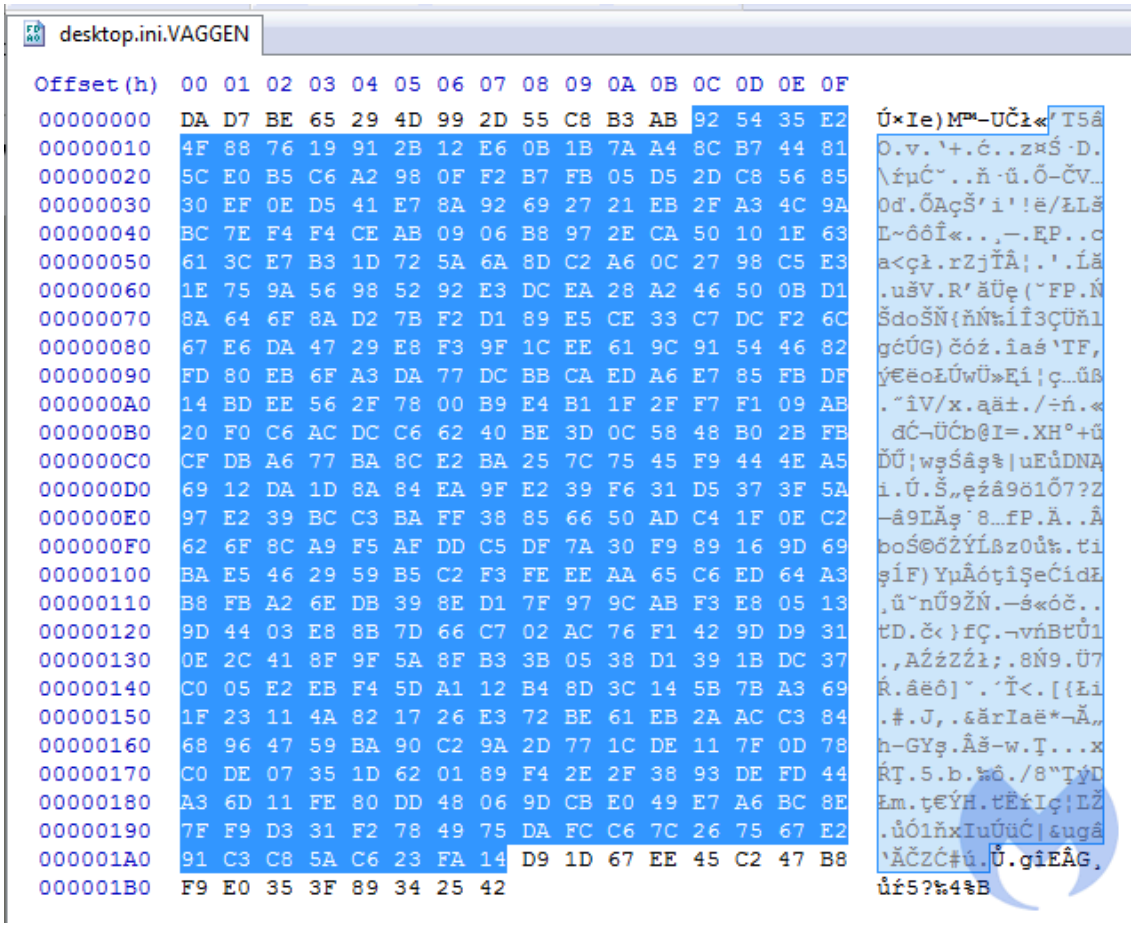
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

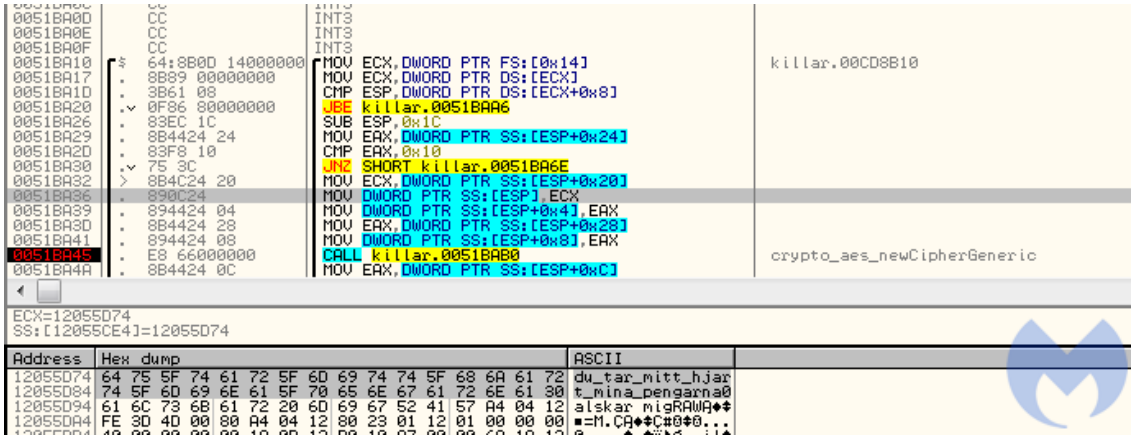


The content of the output file (with .VAGGEN extension) contains:

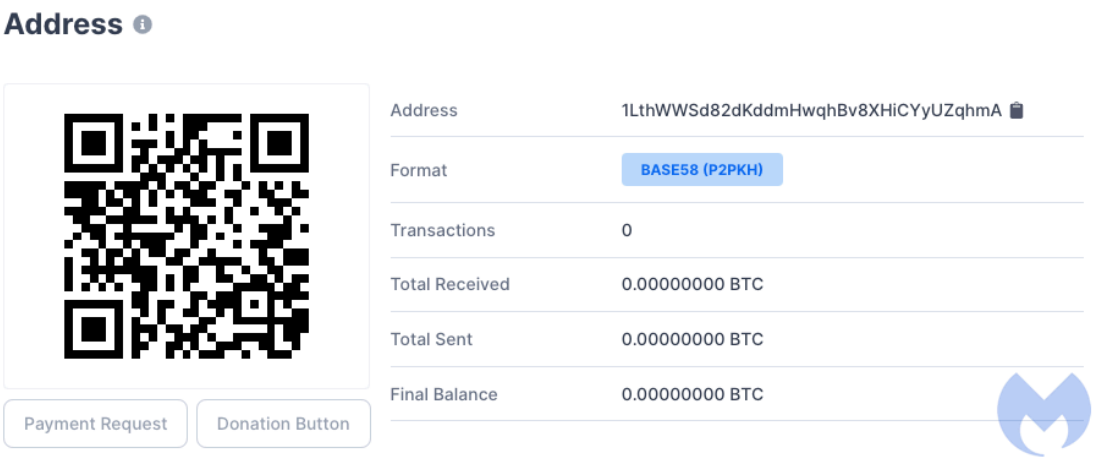
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

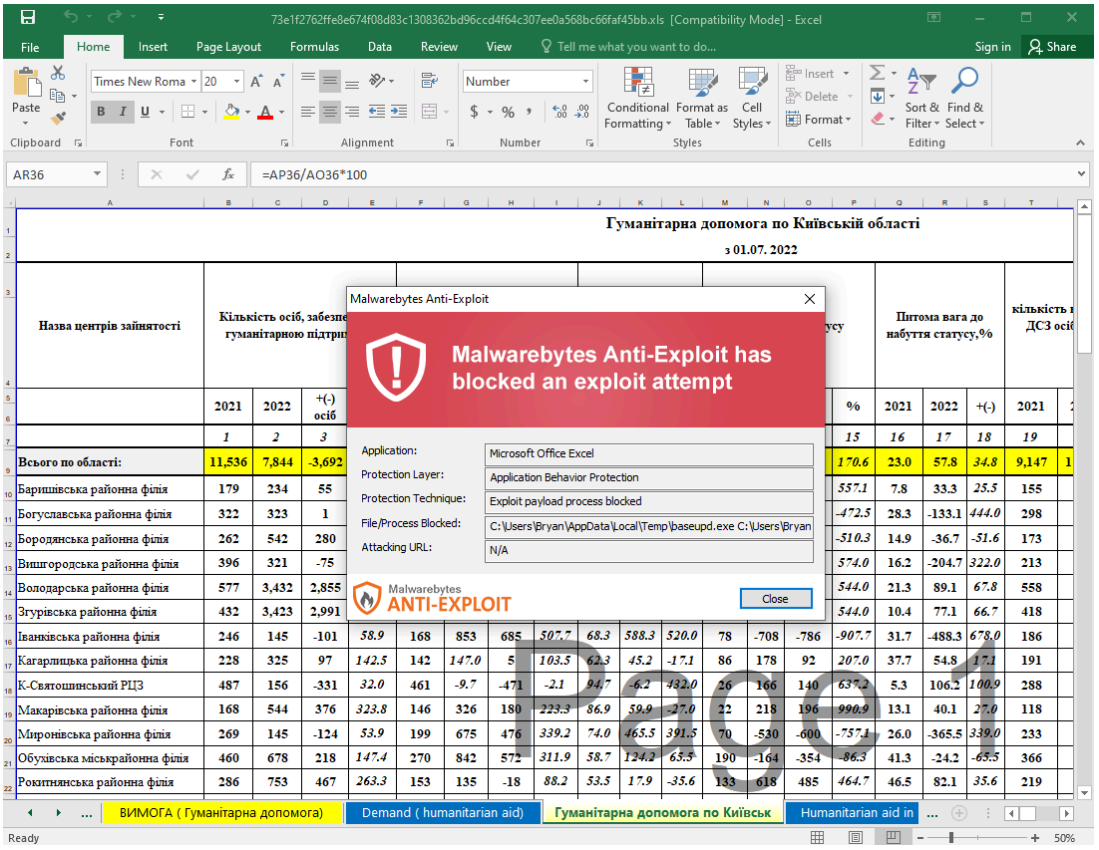
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

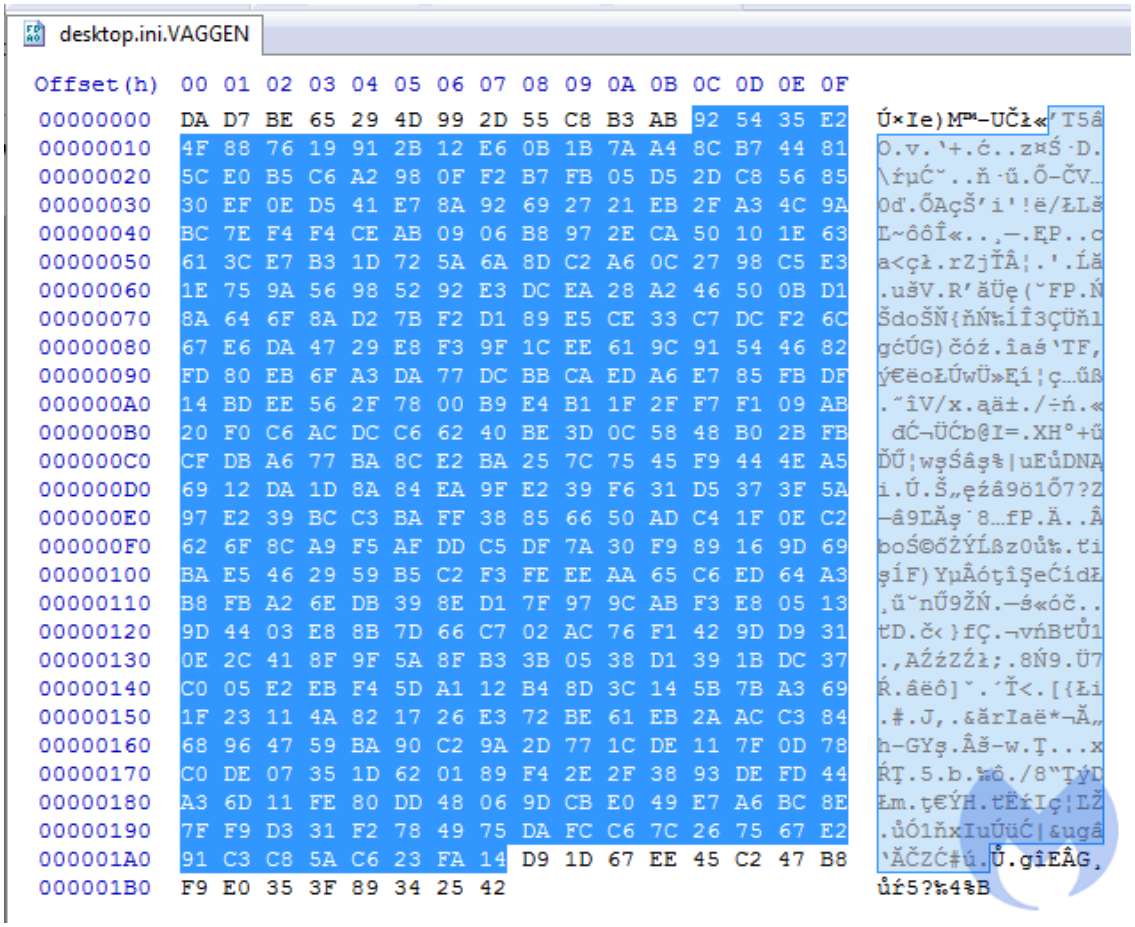
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

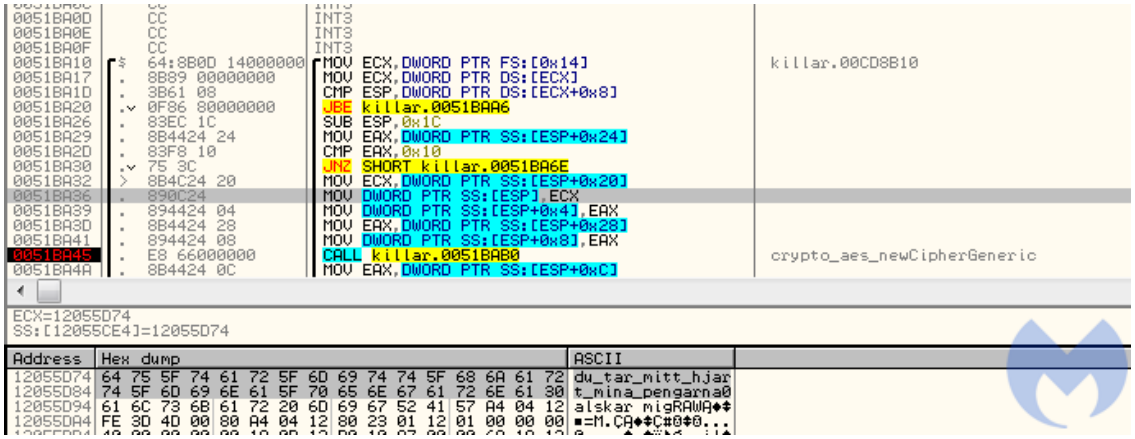


The content of the output file (with .VAGGEN extension) contains:

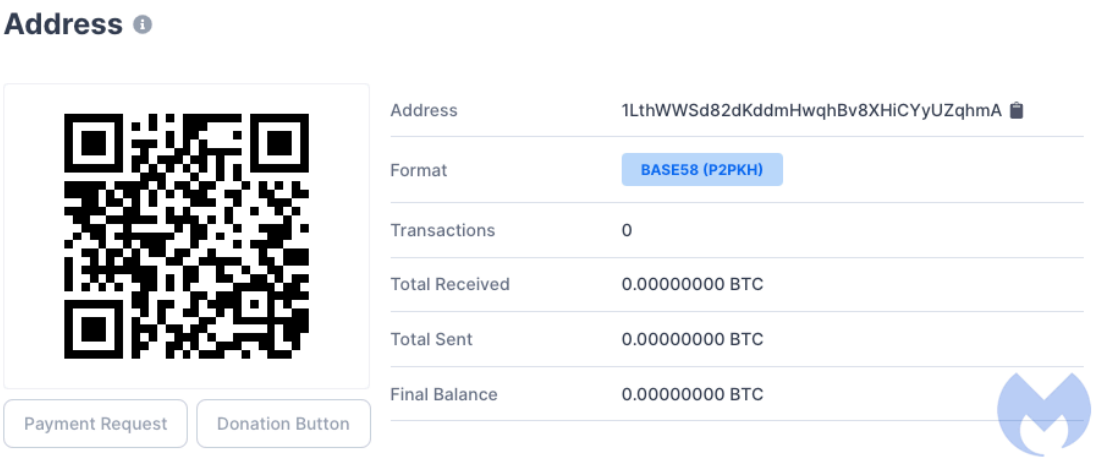
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

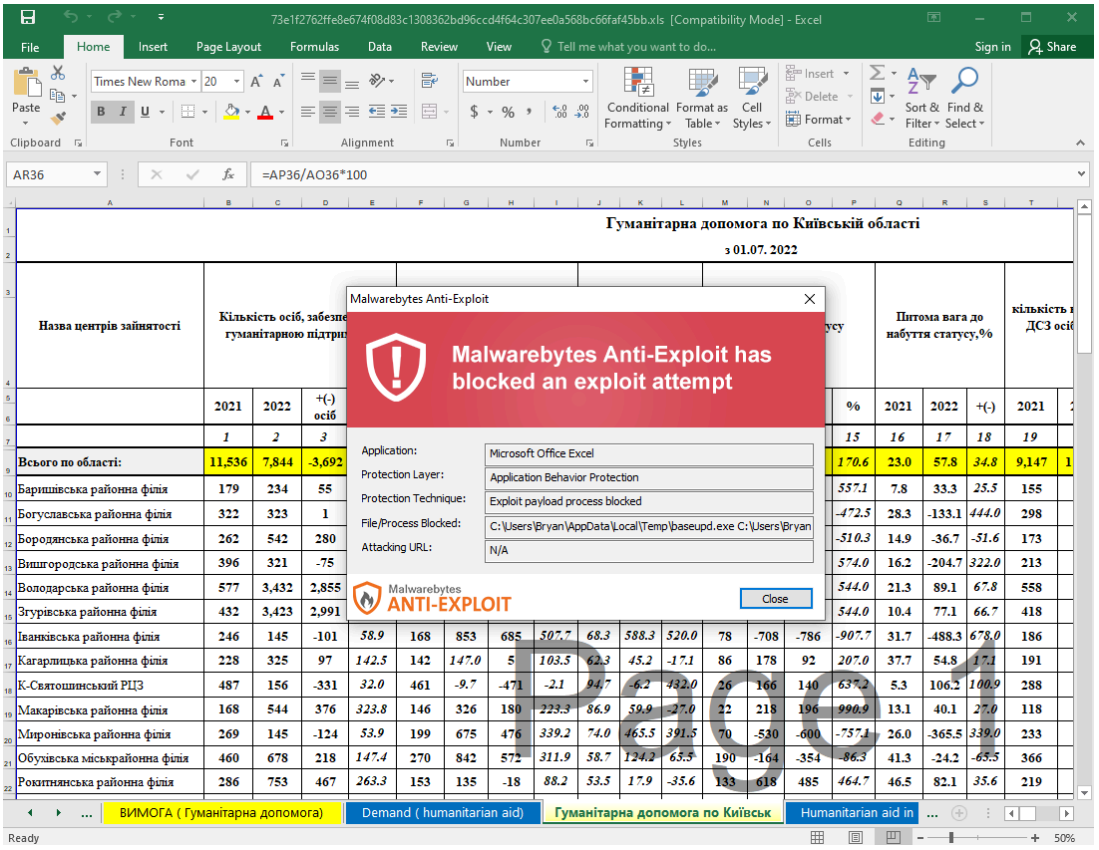
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

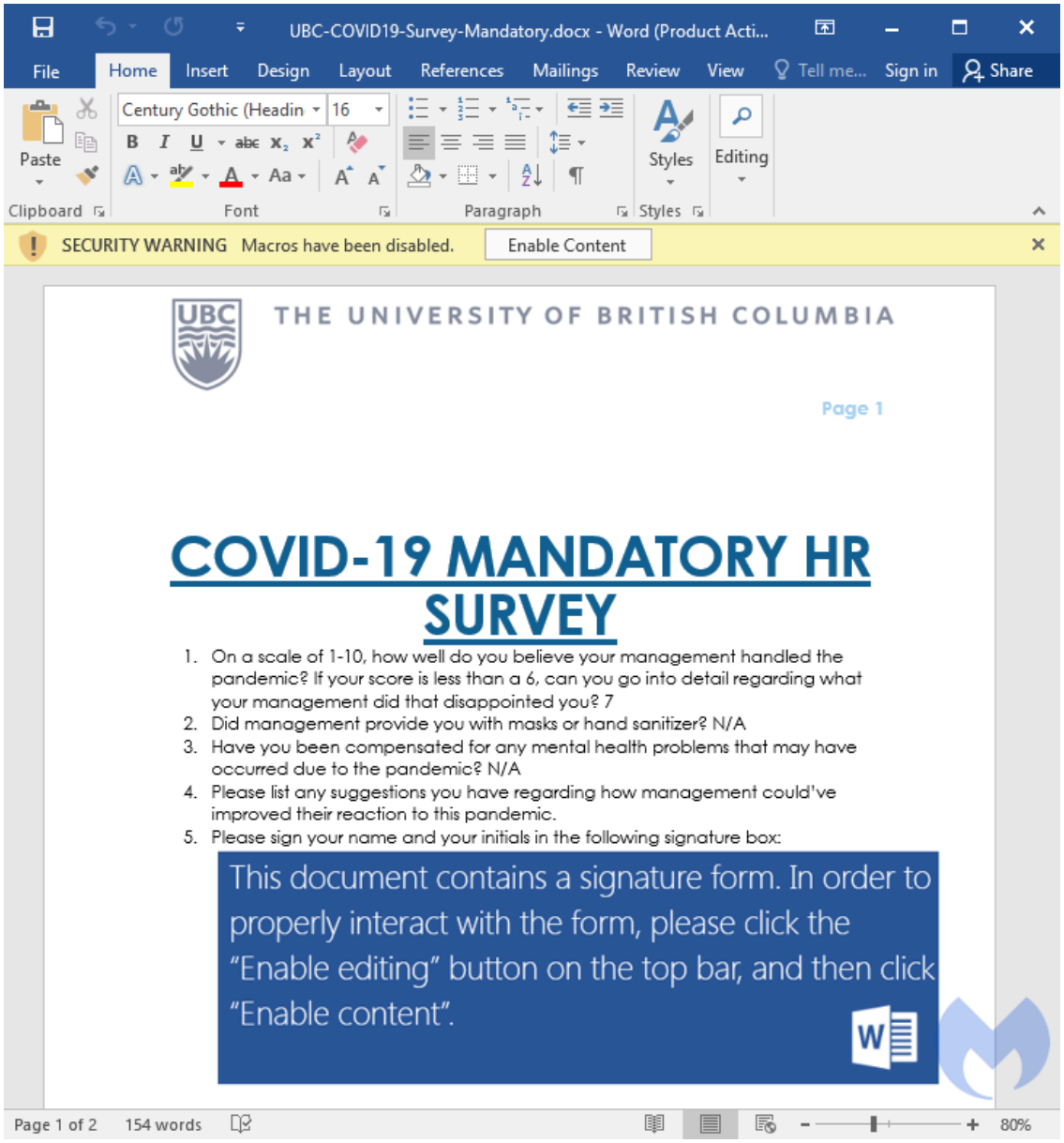
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

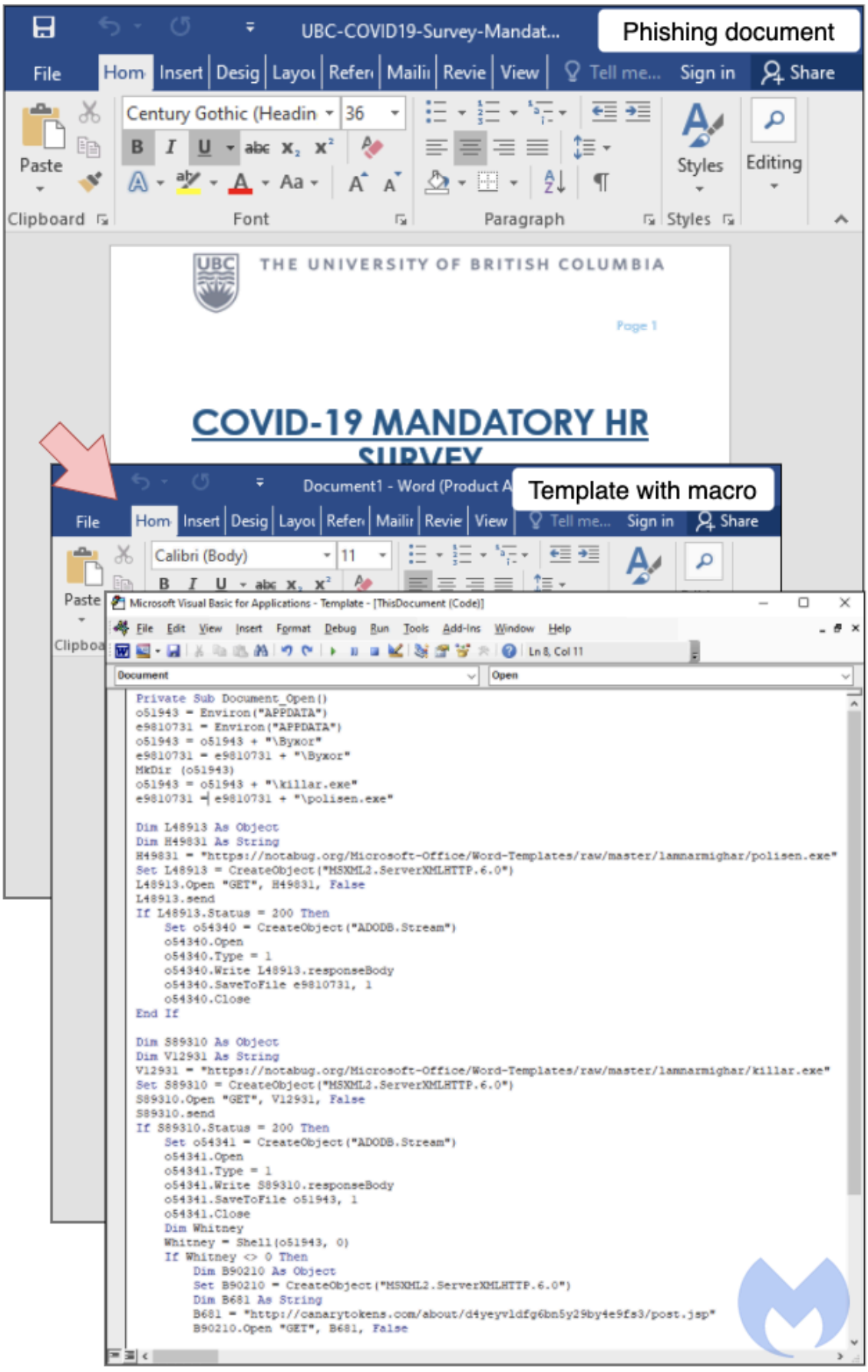
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

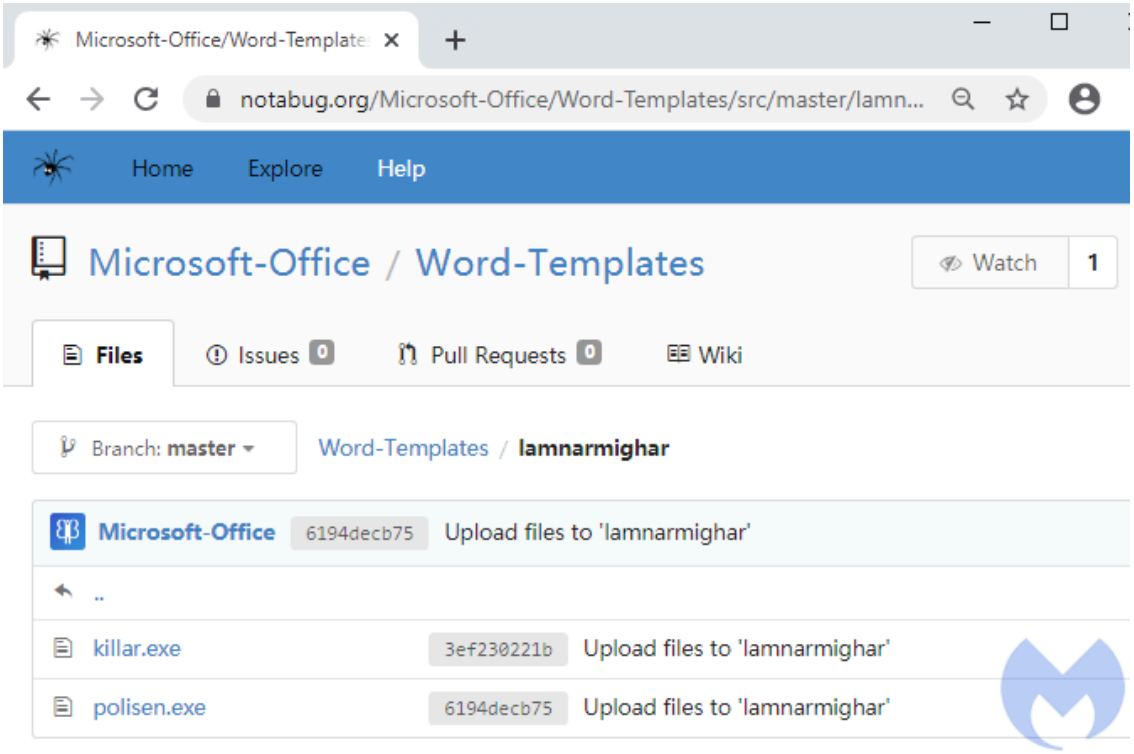
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



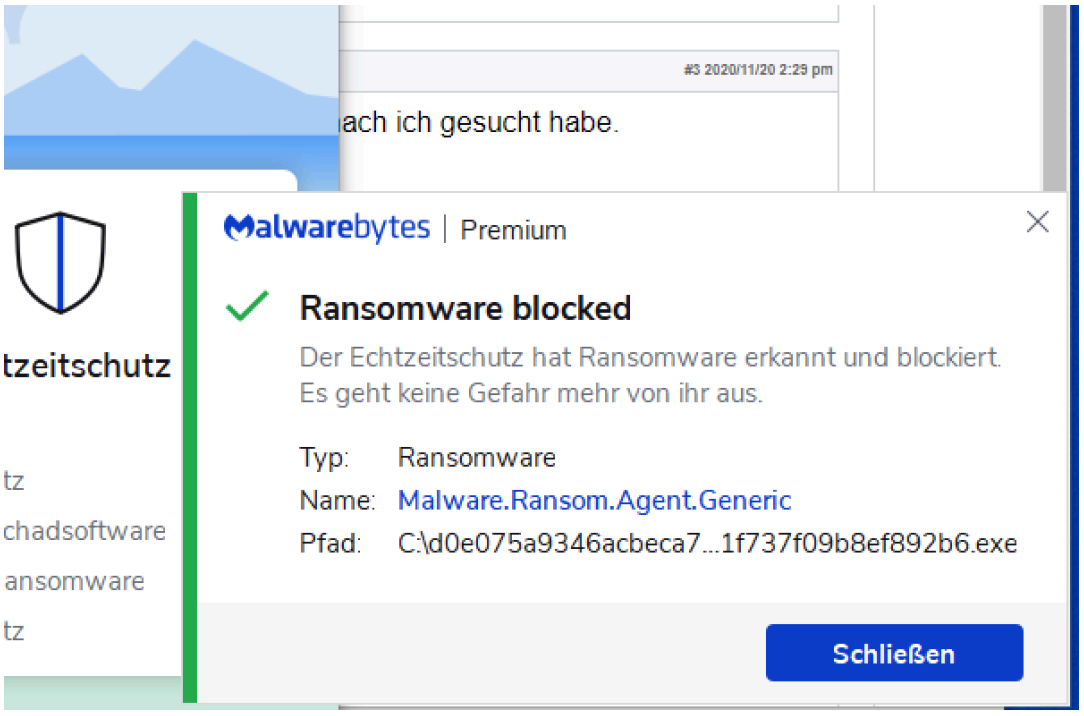
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

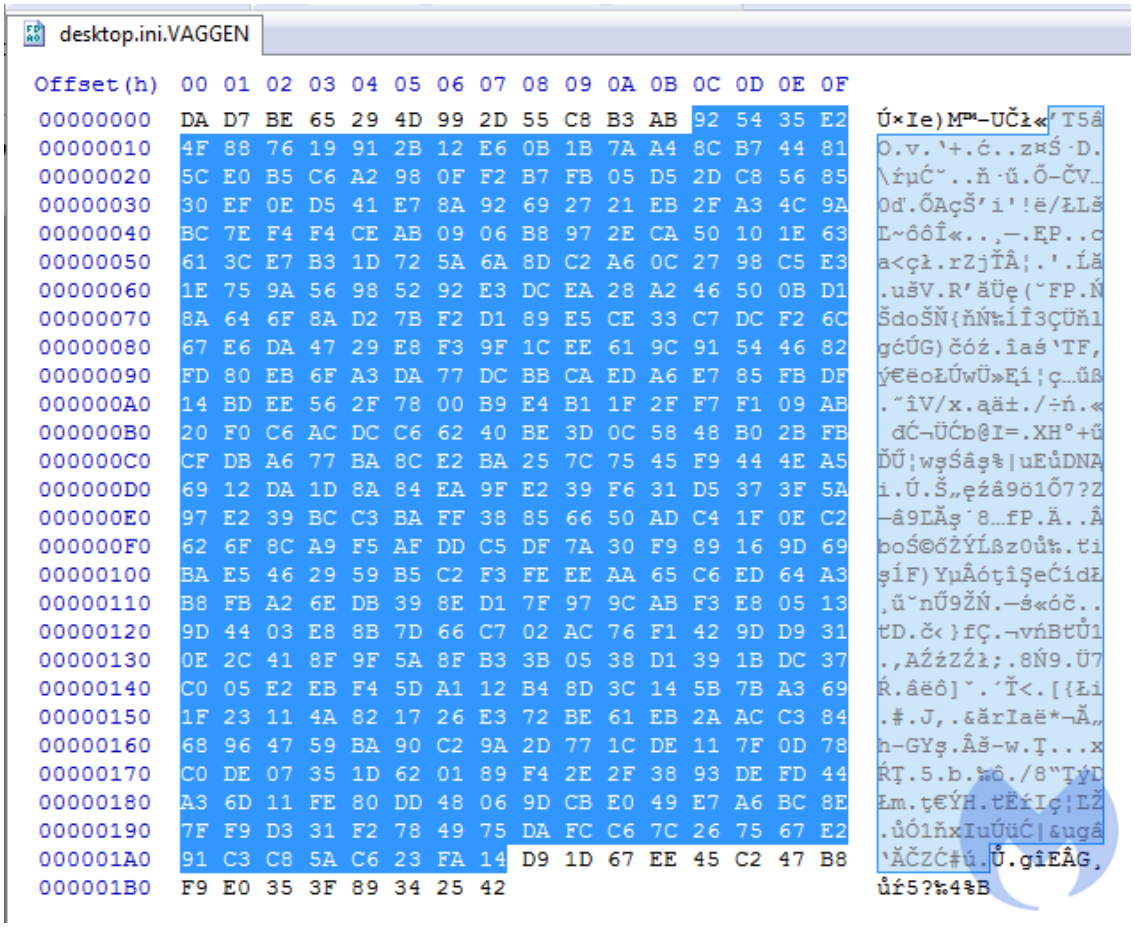
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

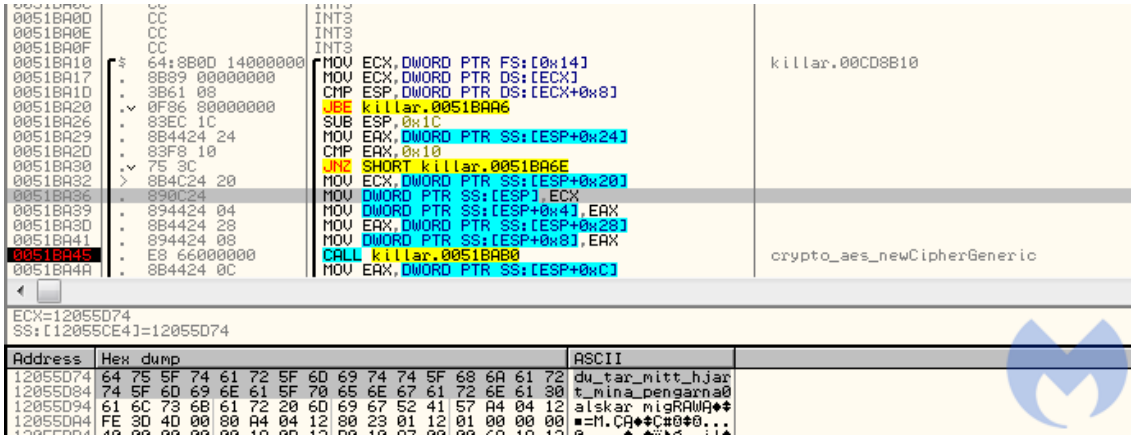


The content of the output file (with .VAGGEN extension) contains:

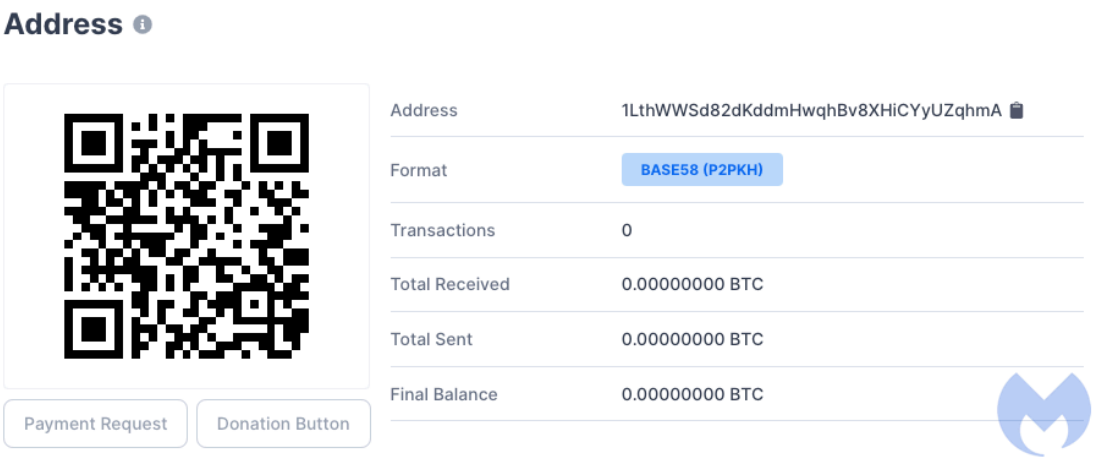
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

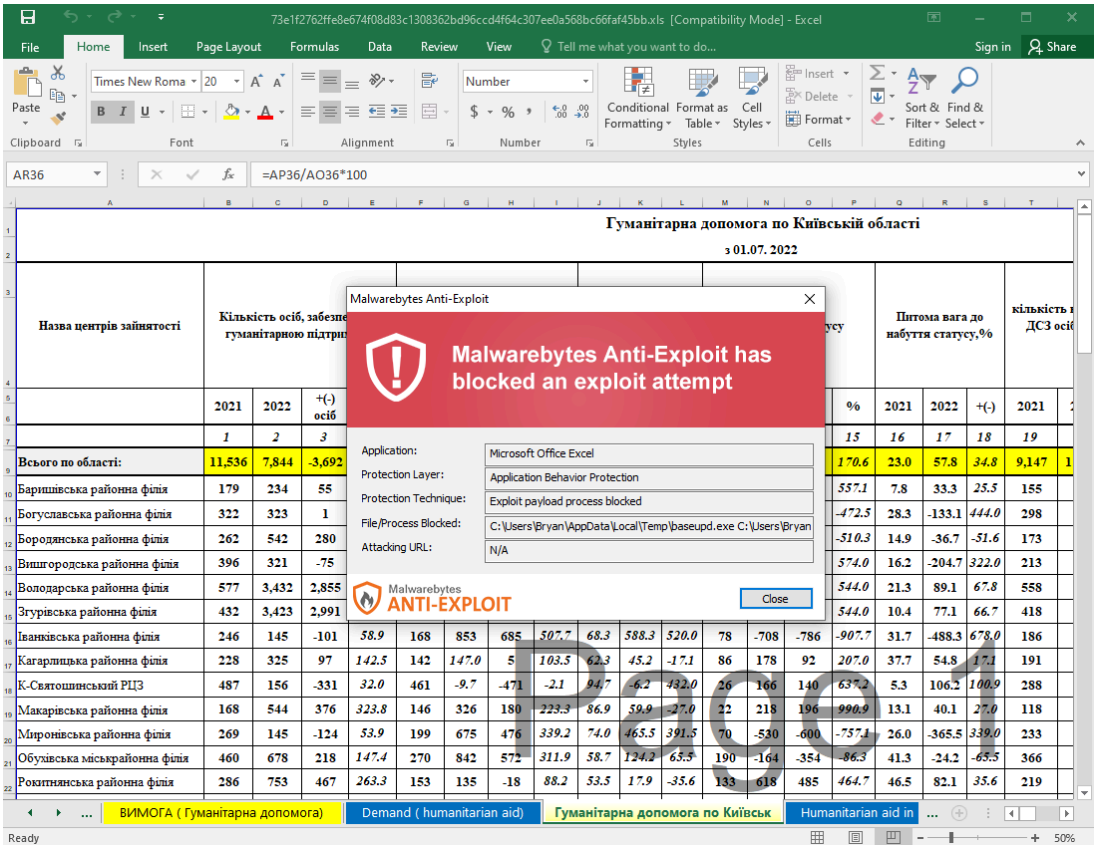
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

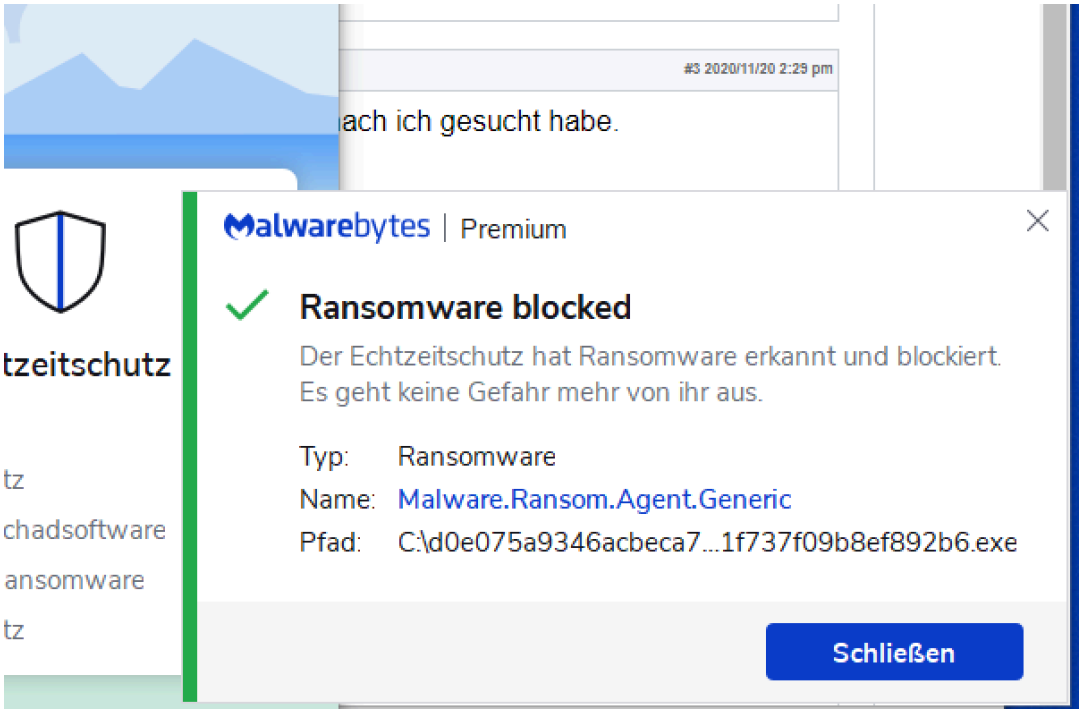
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

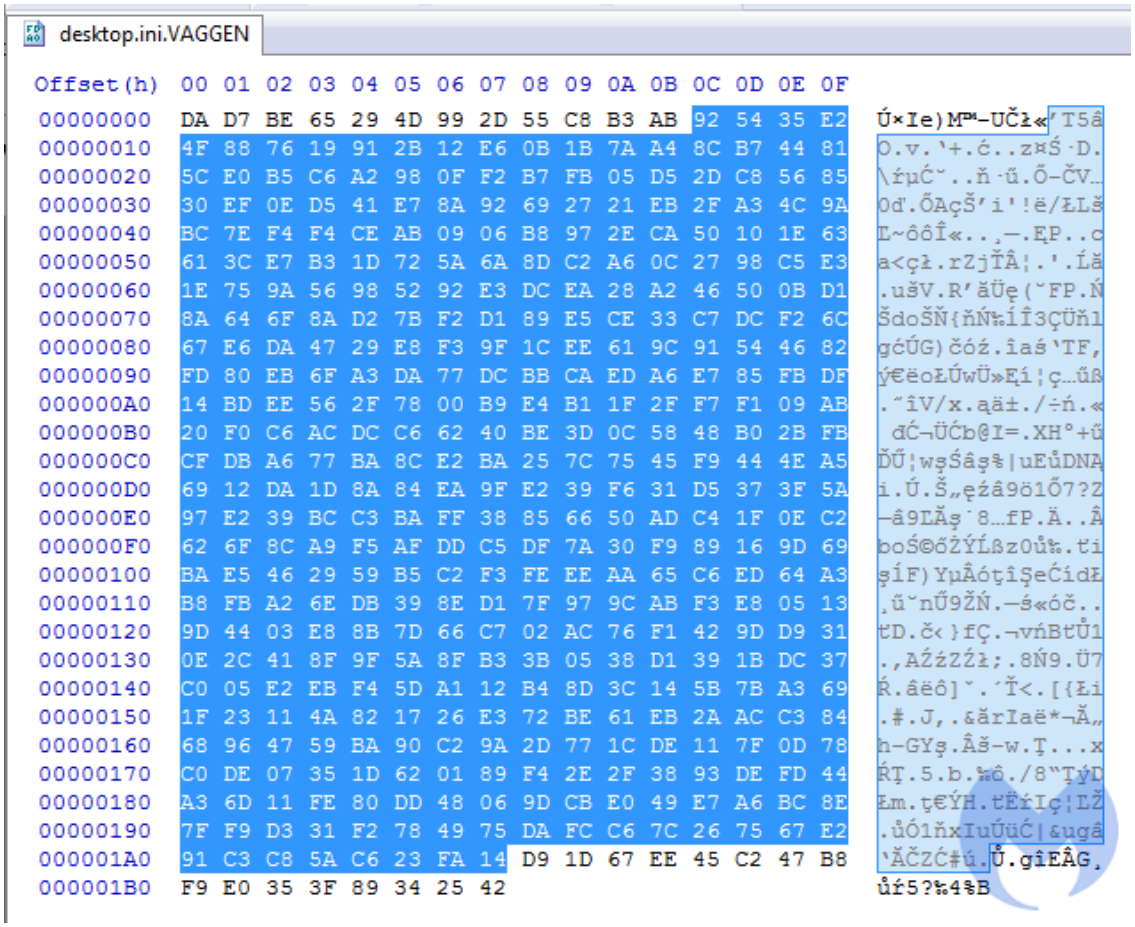
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



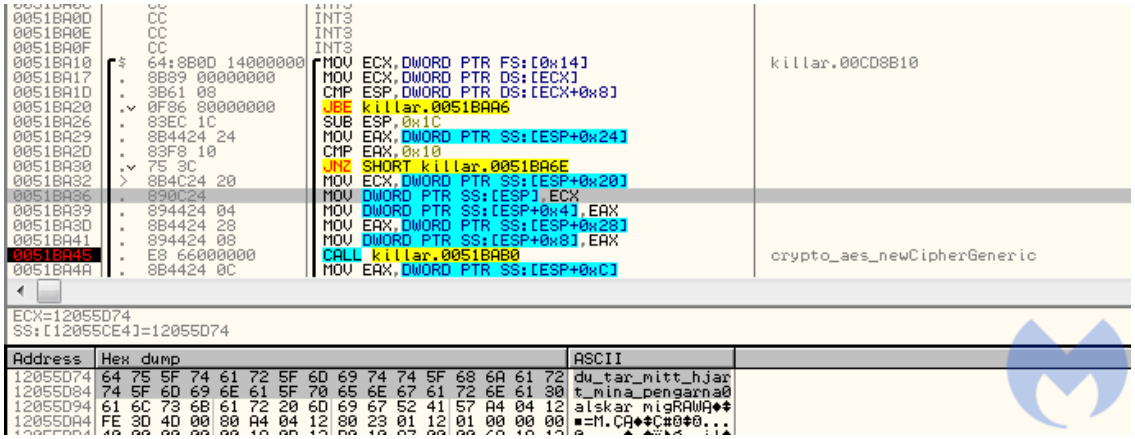
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

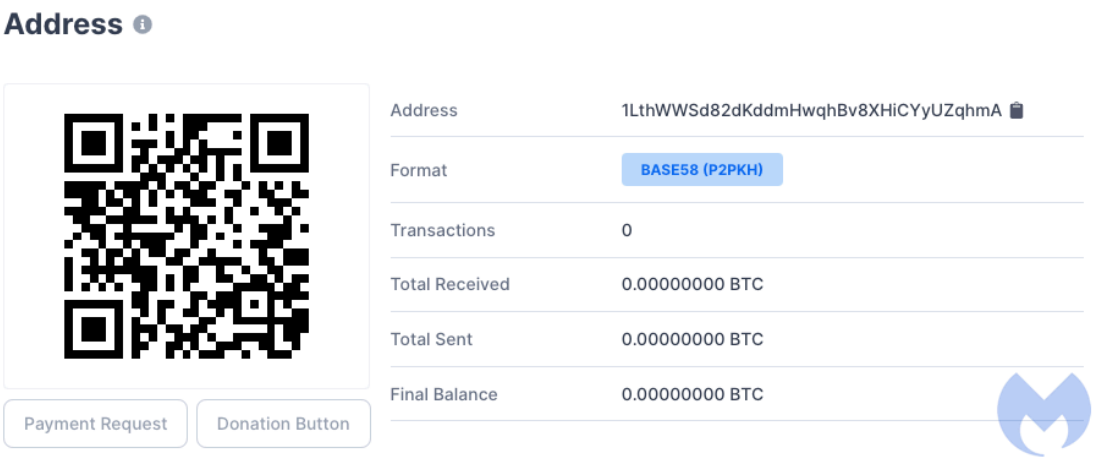
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

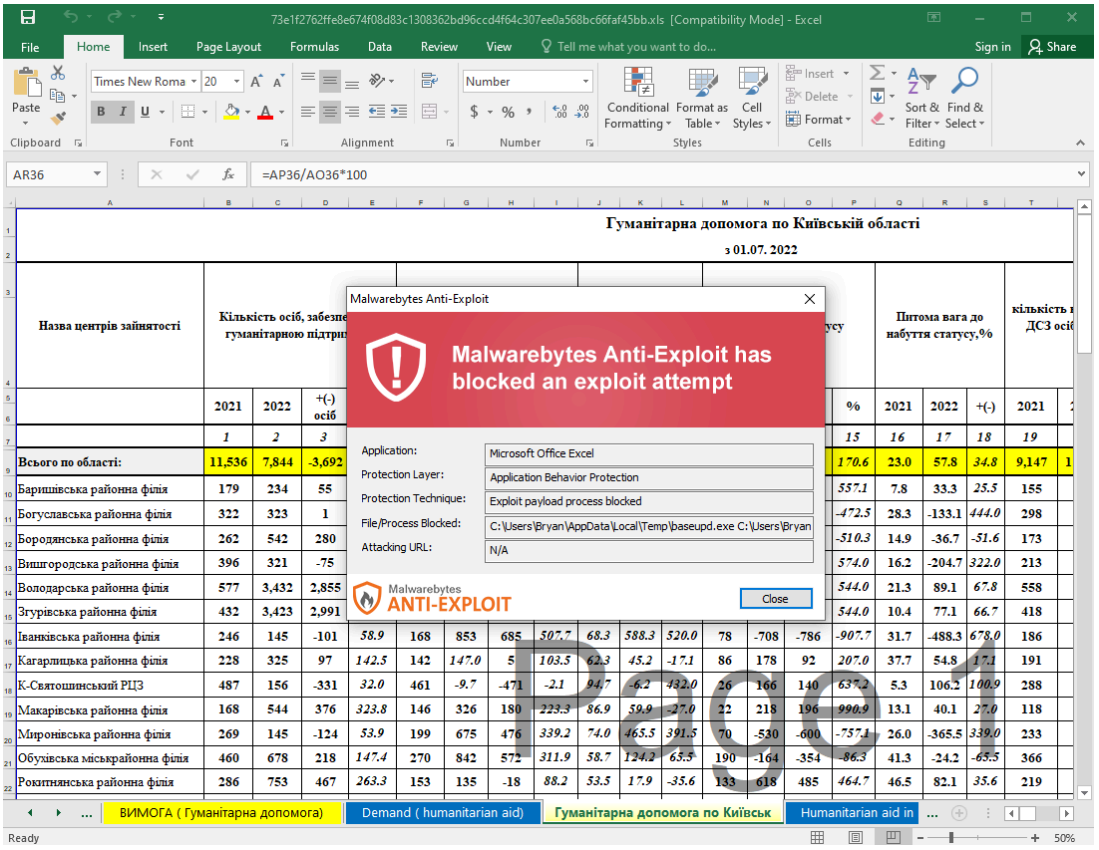
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

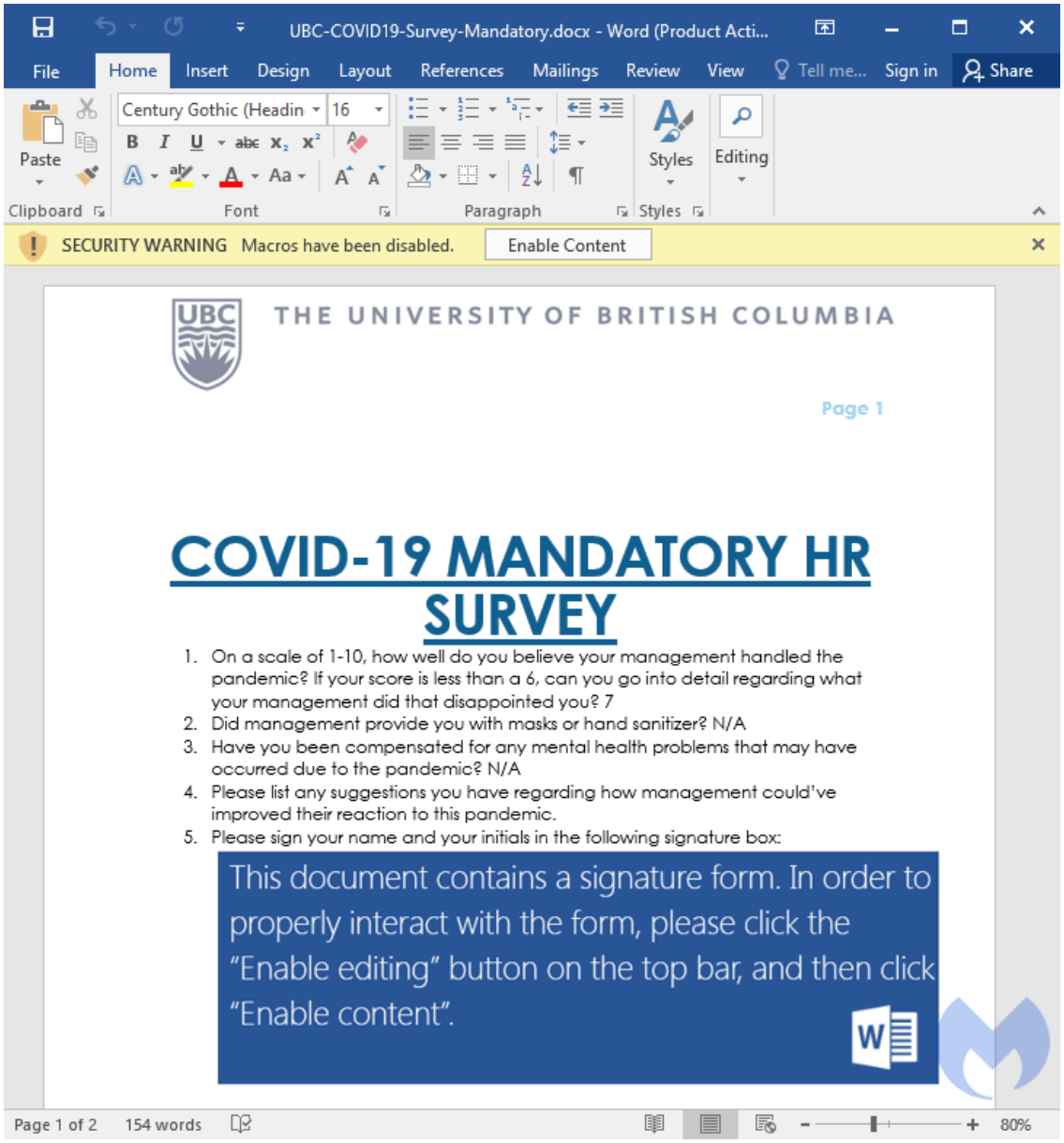
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

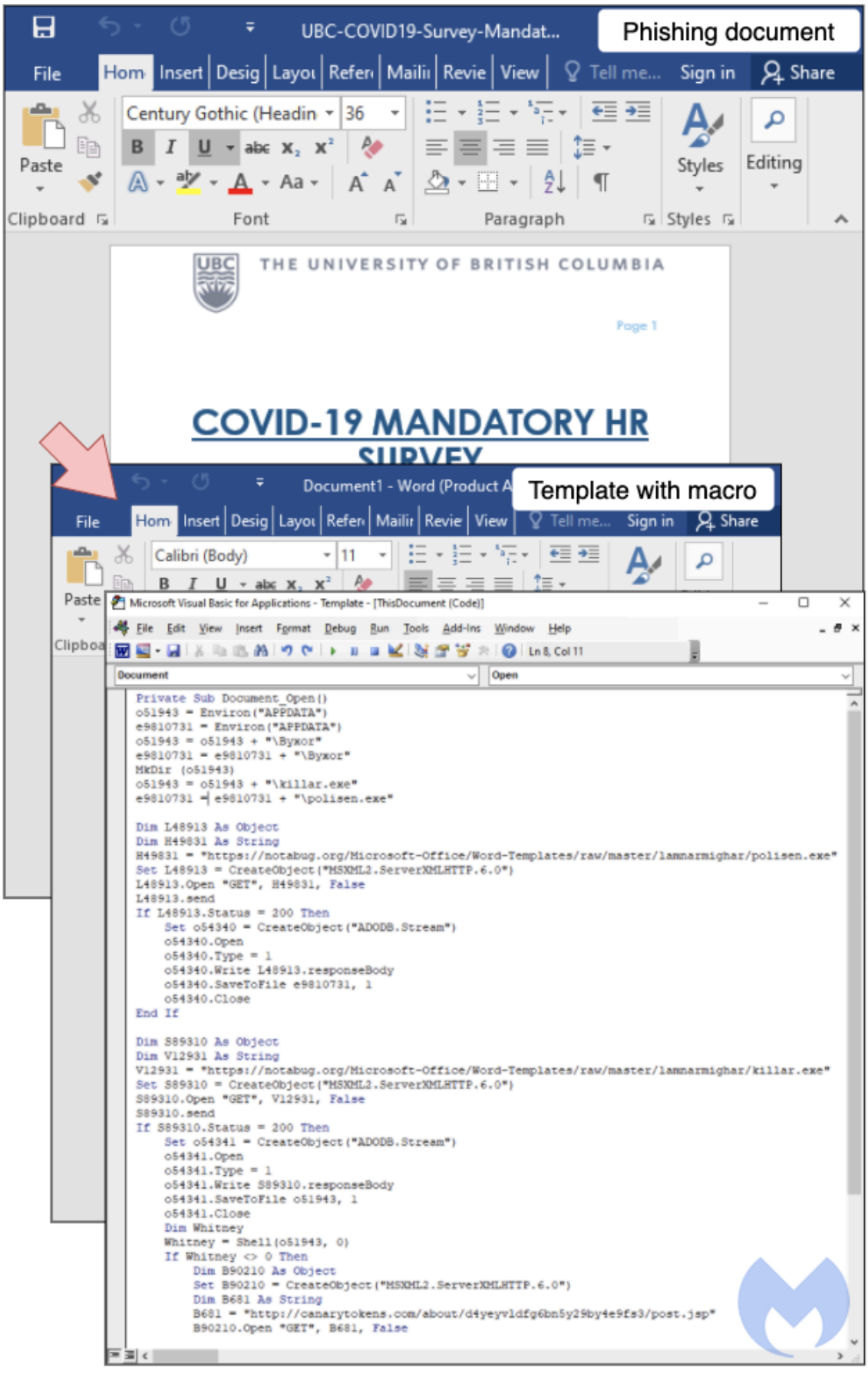
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

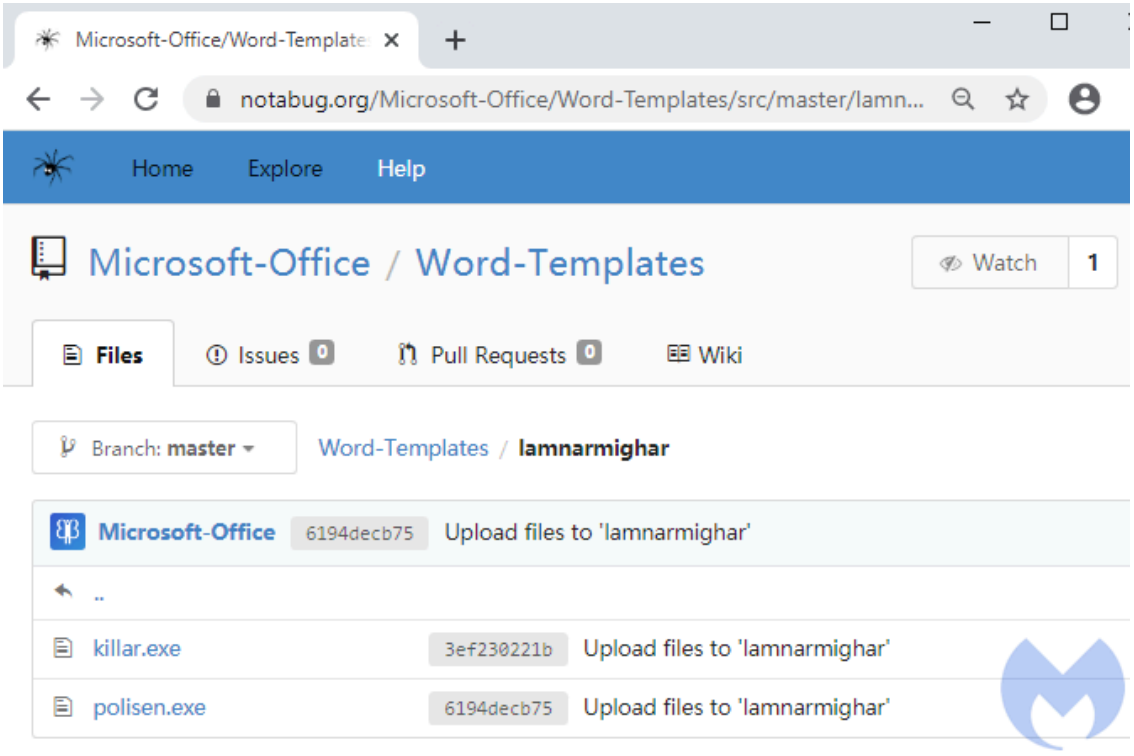
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



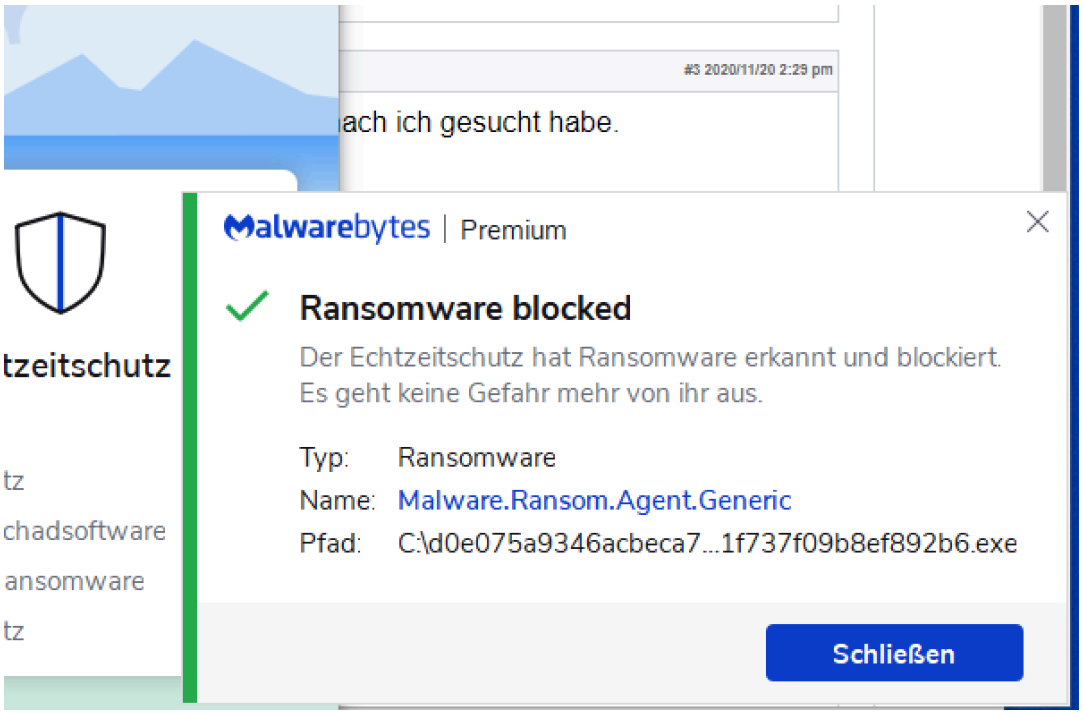
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

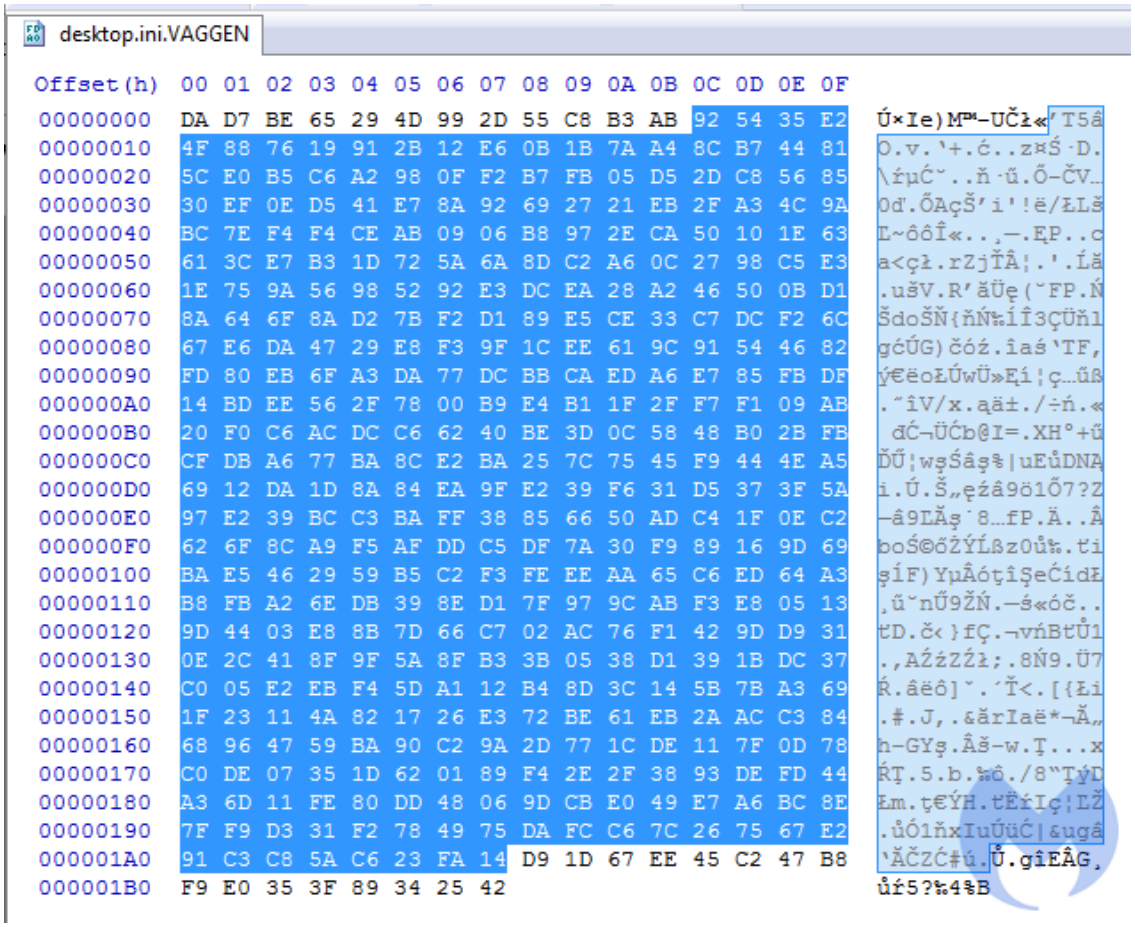
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

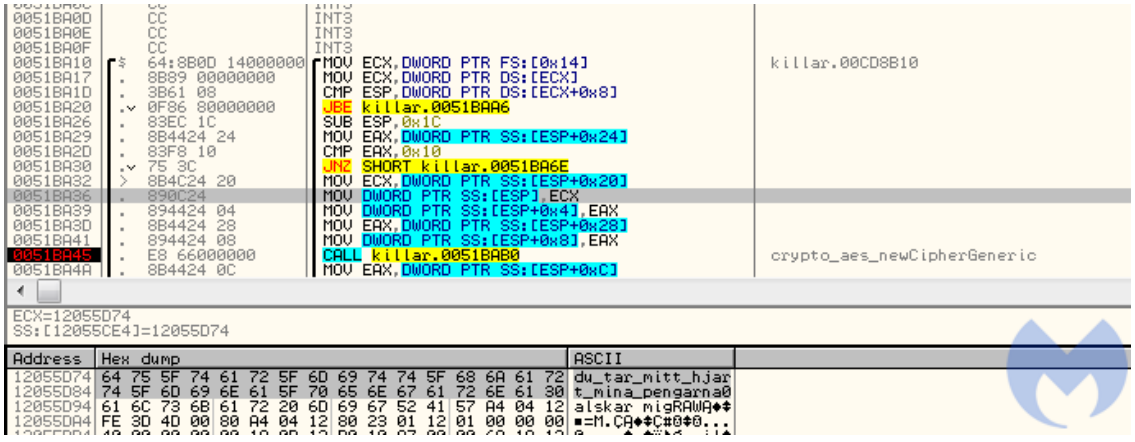


The content of the output file (with .VAGGEN extension) contains:

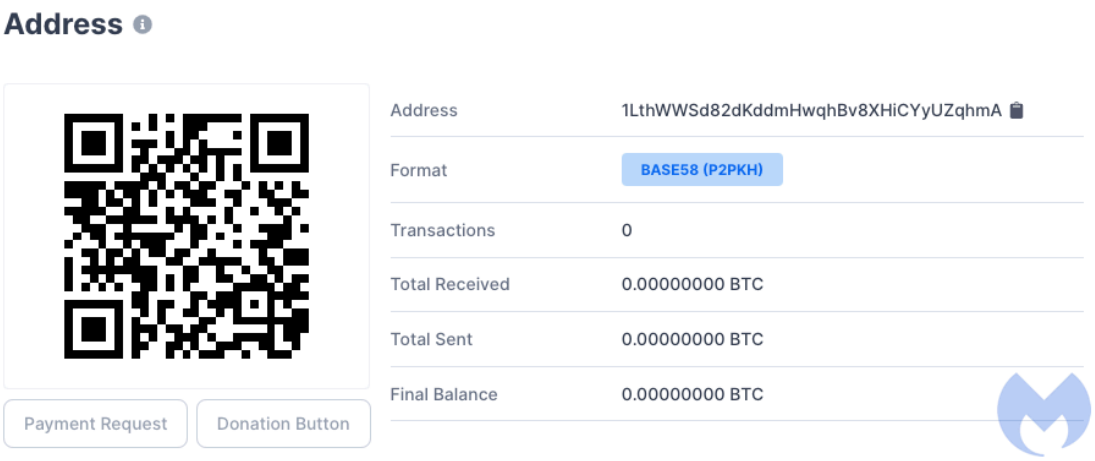
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

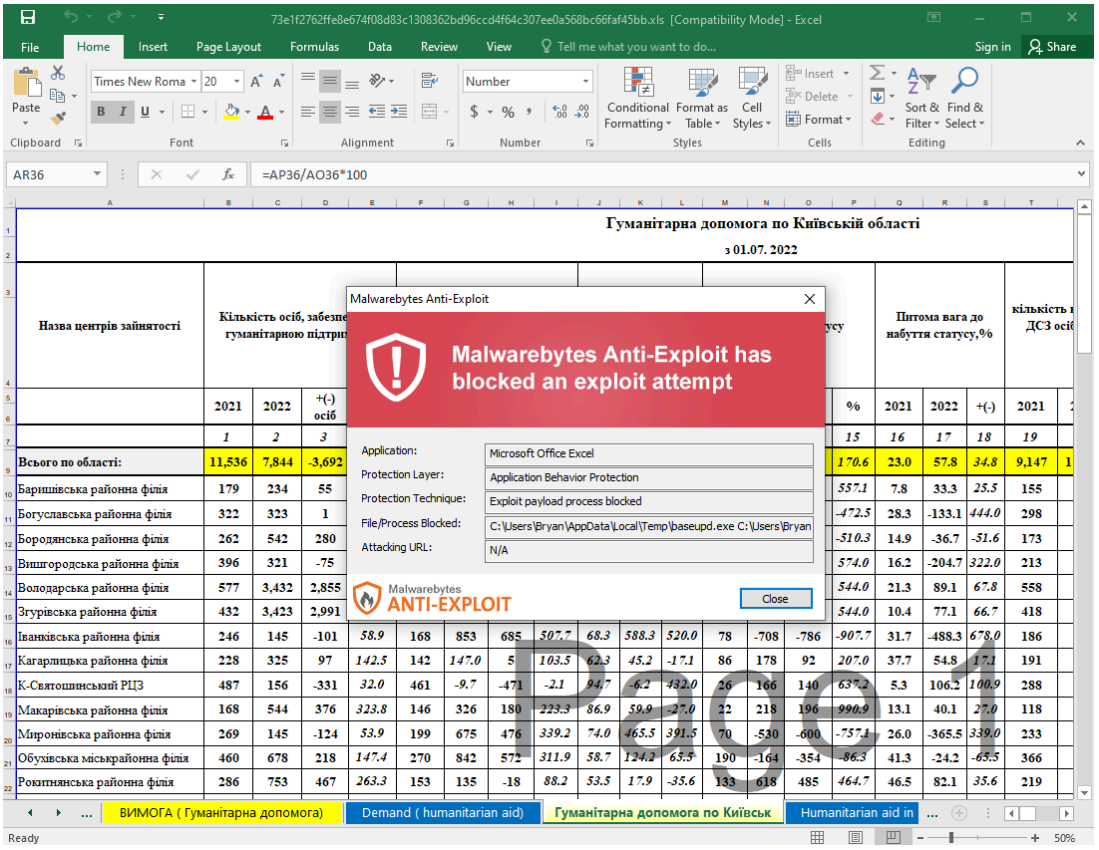
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

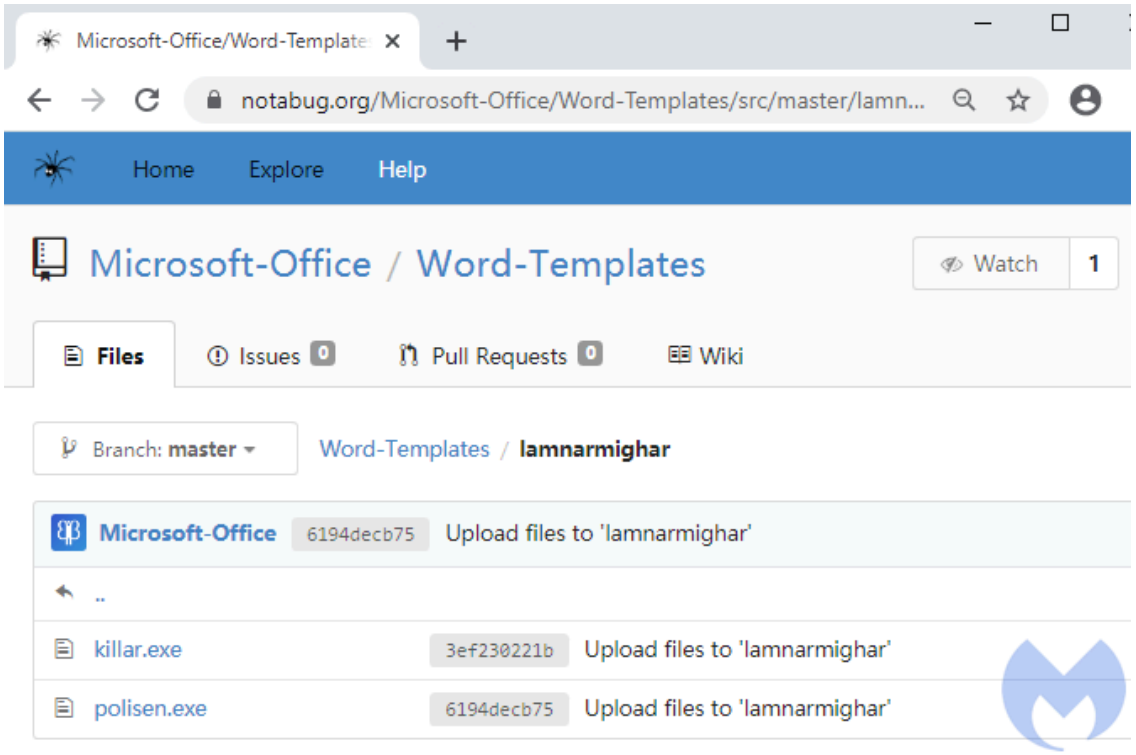
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



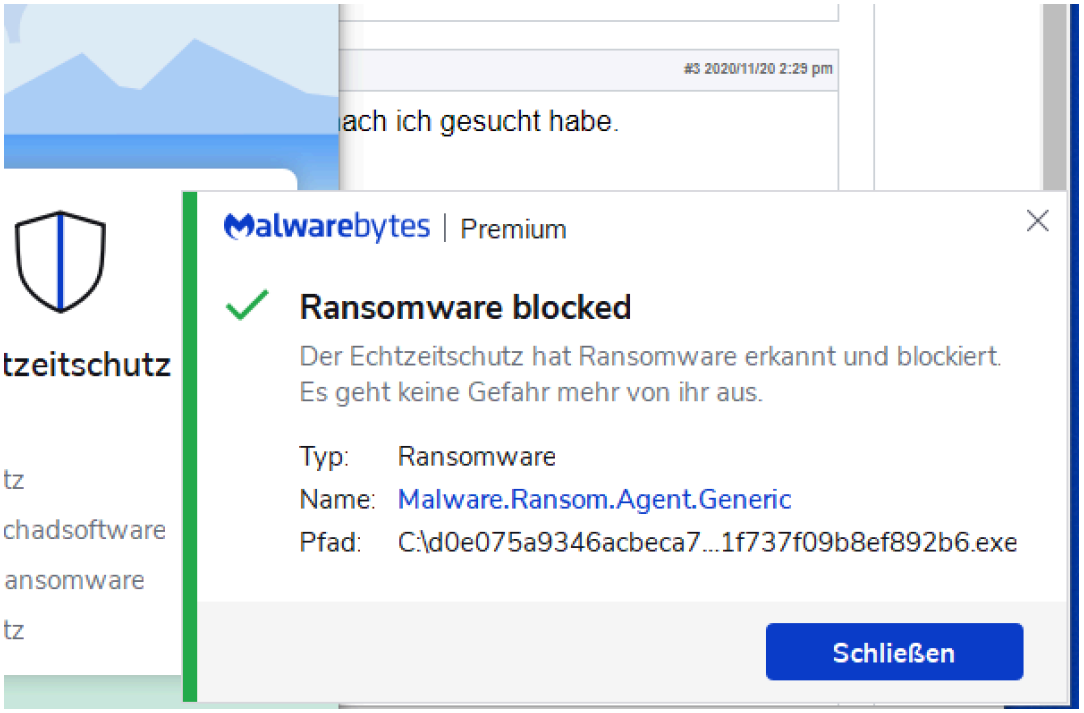
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

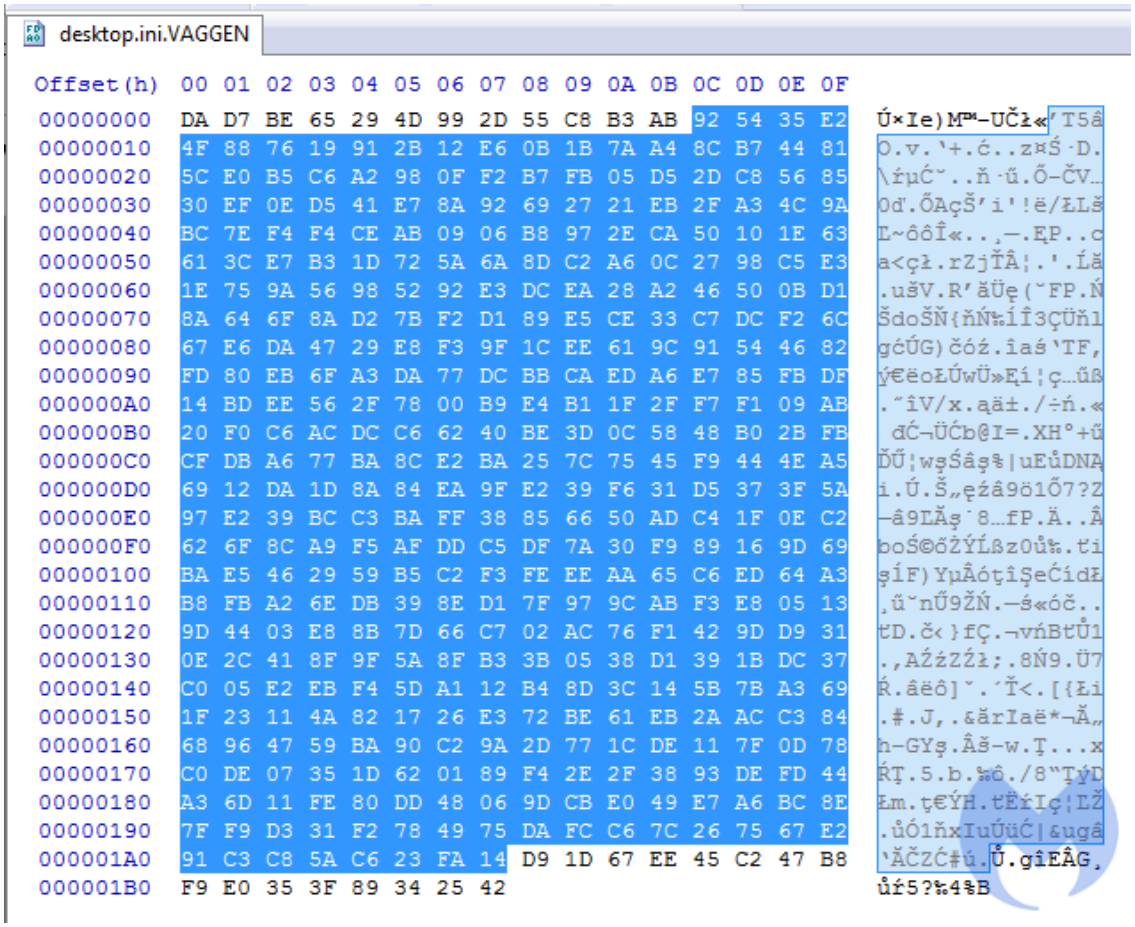
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

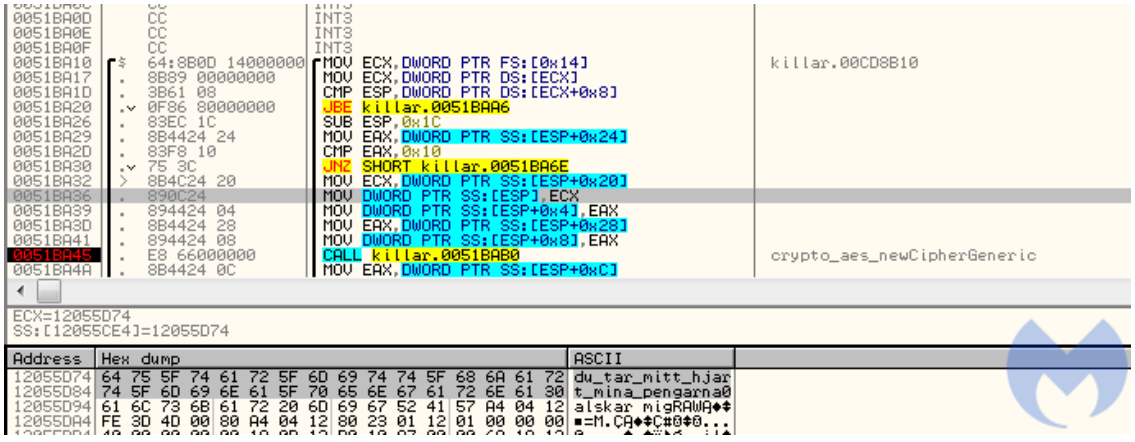


The content of the output file (with .VAGGEN extension) contains:

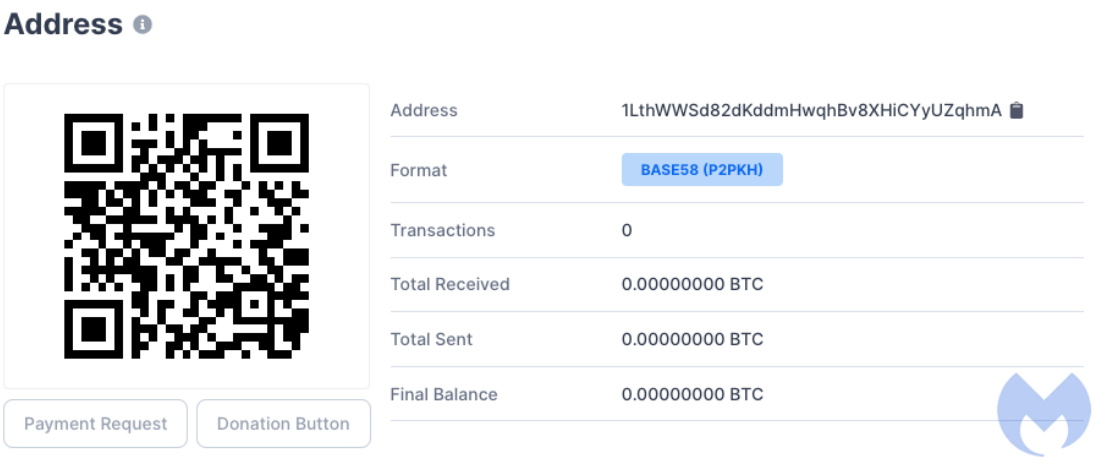
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

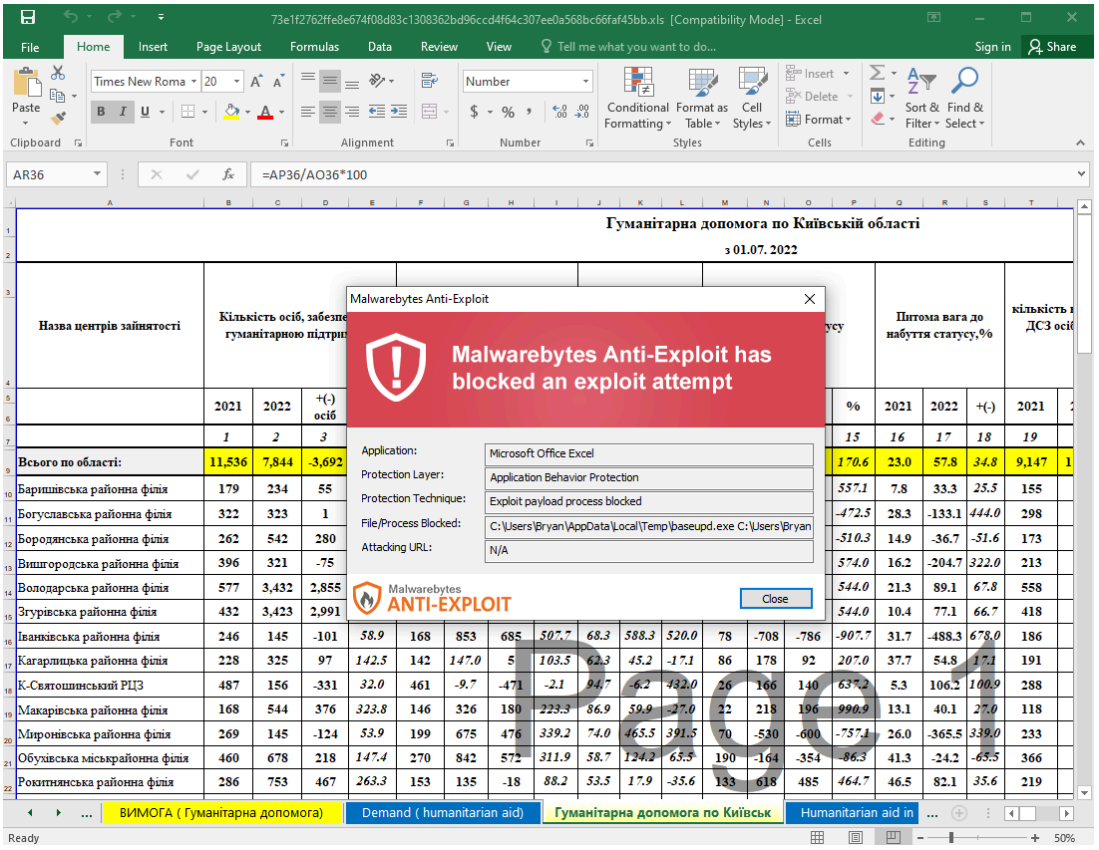
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

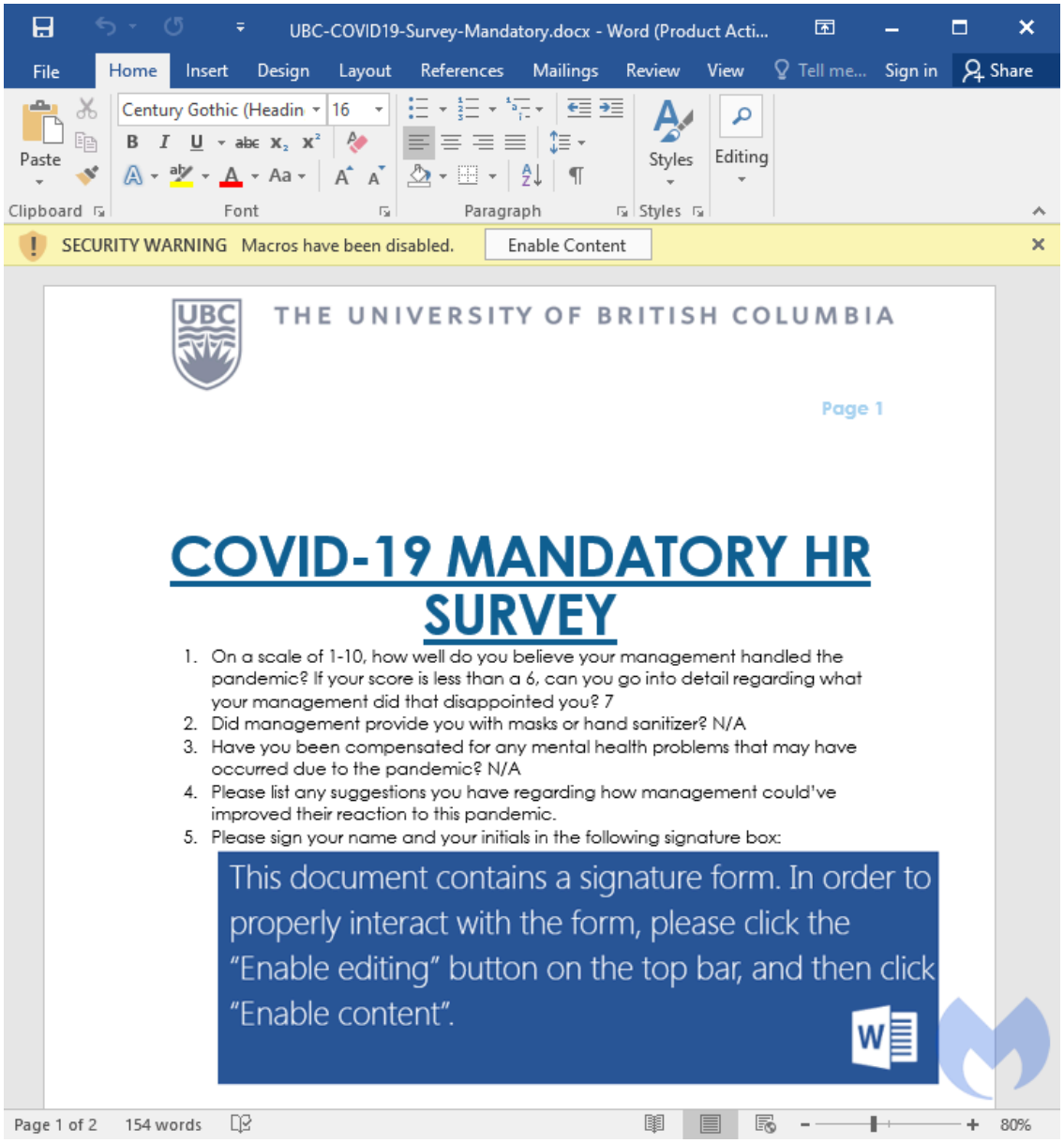
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

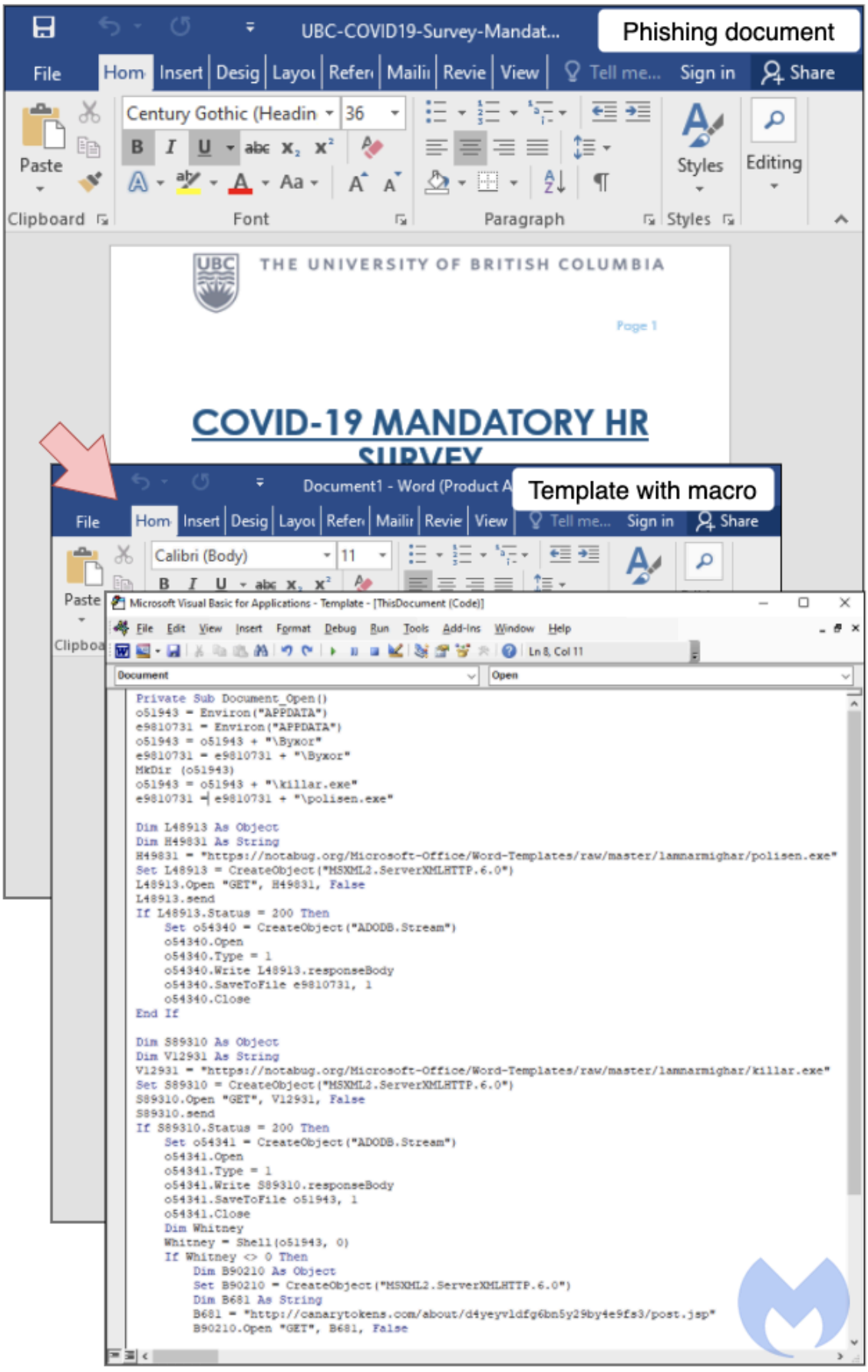
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

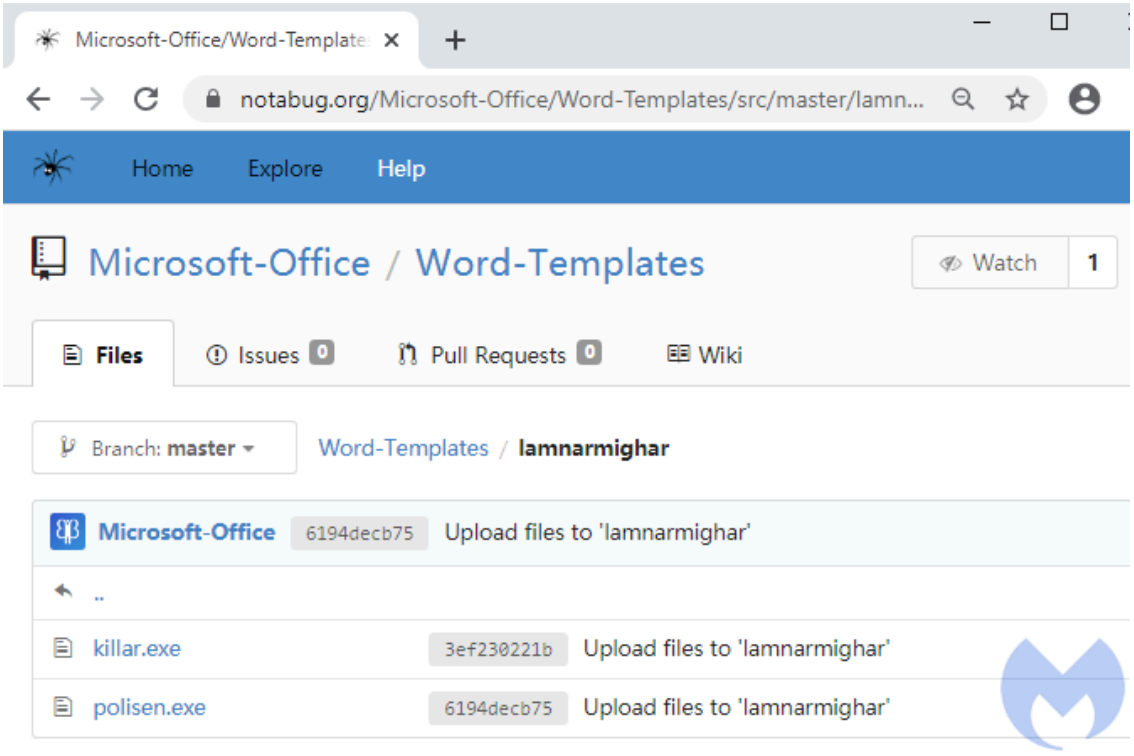
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



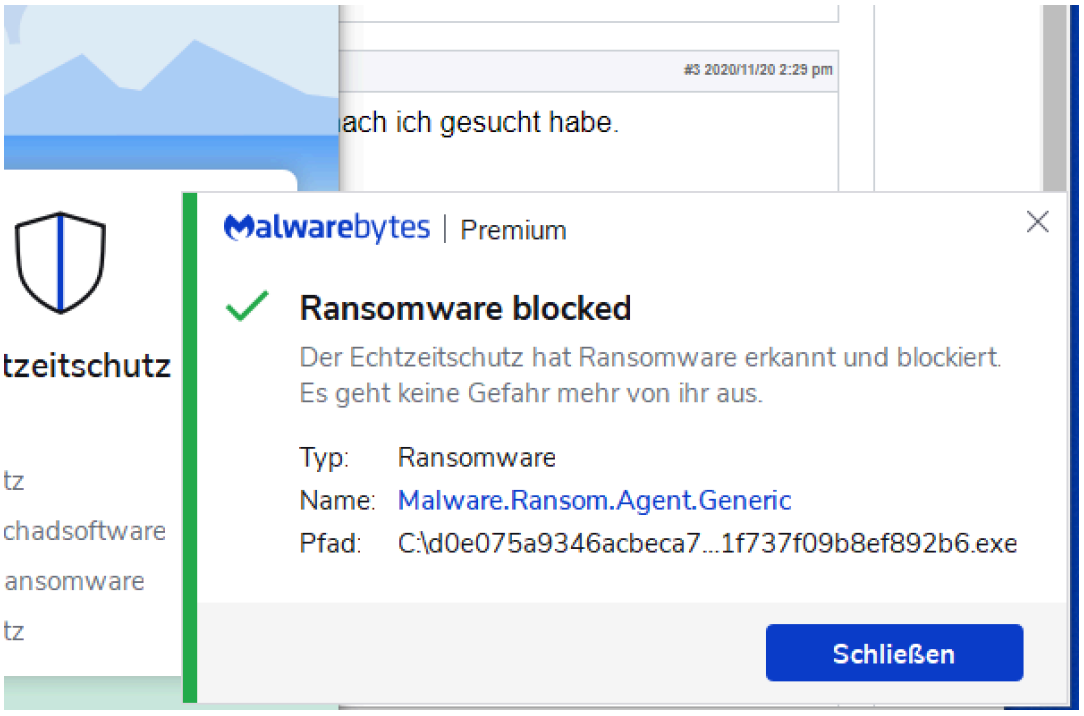
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

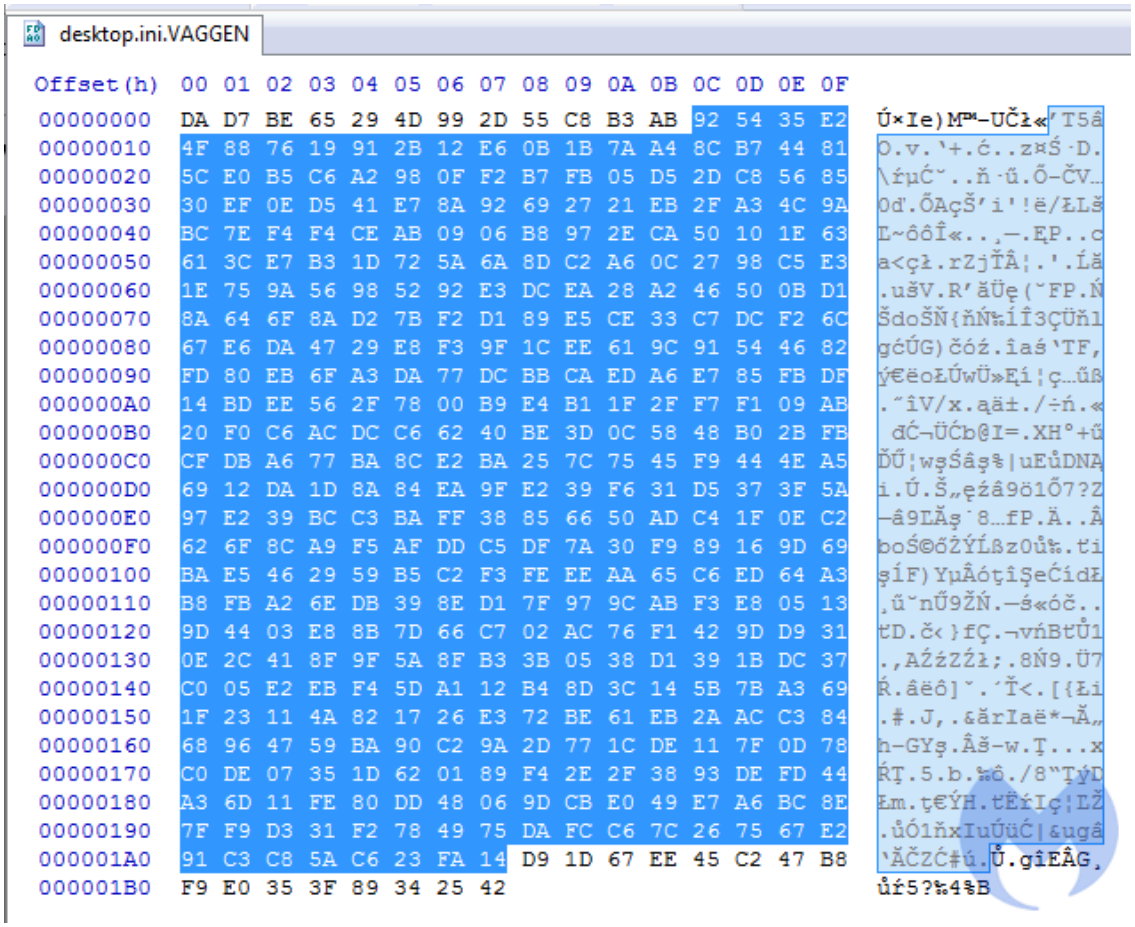
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

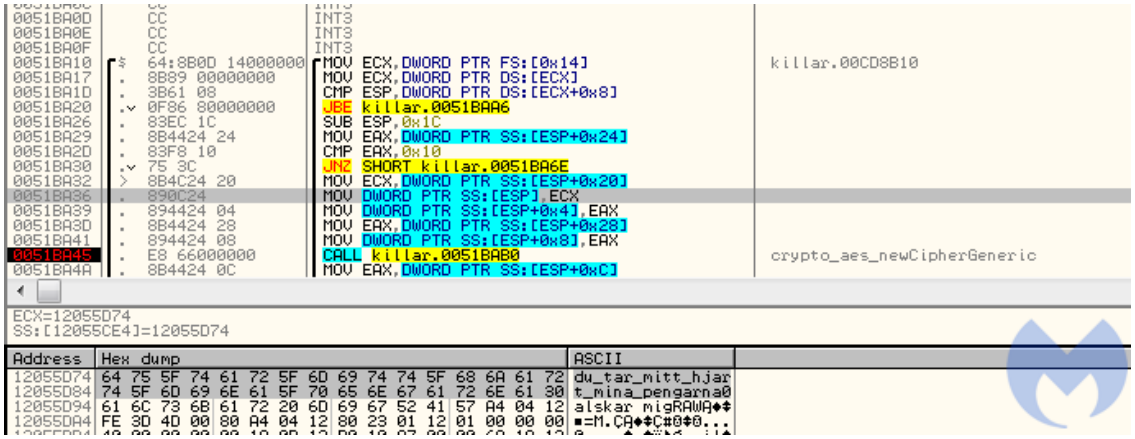


The content of the output file (with .VAGGEN extension) contains:

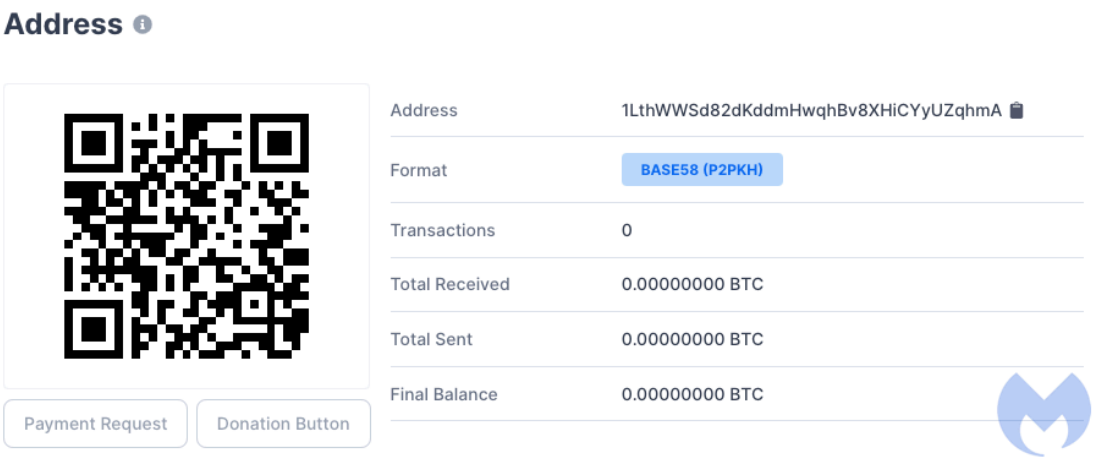
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

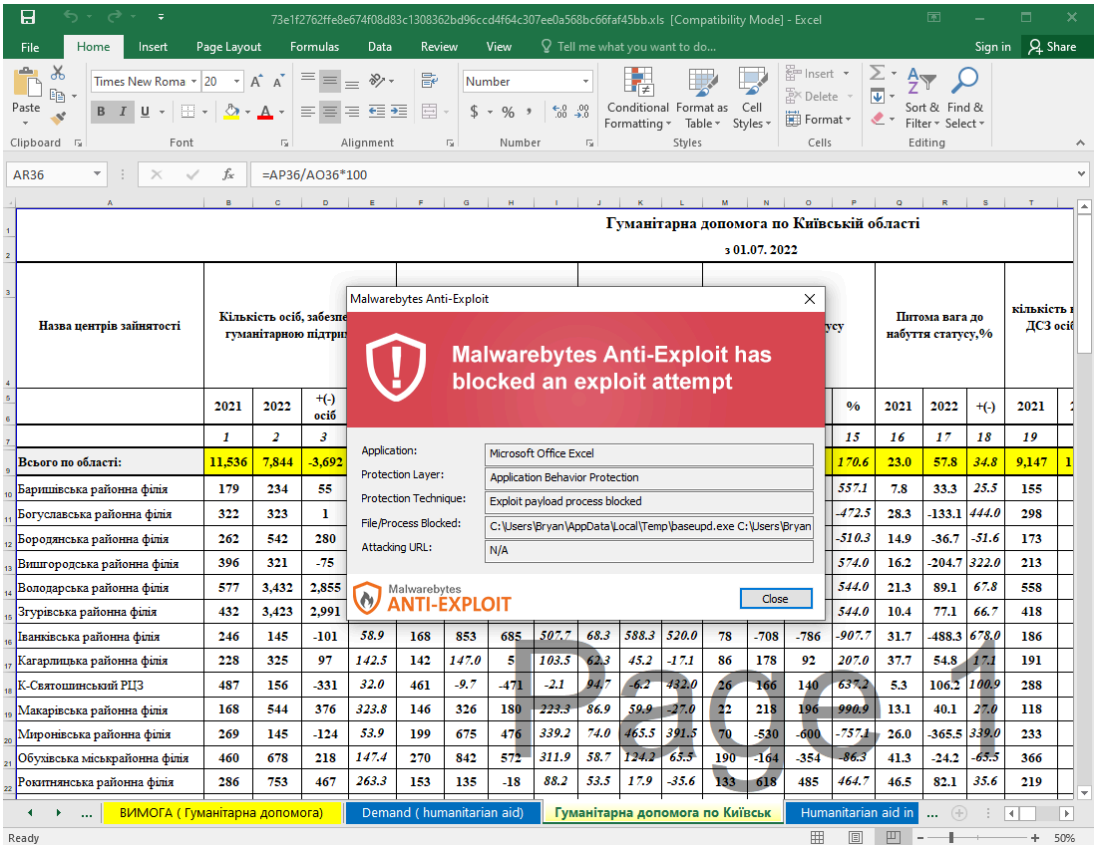
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

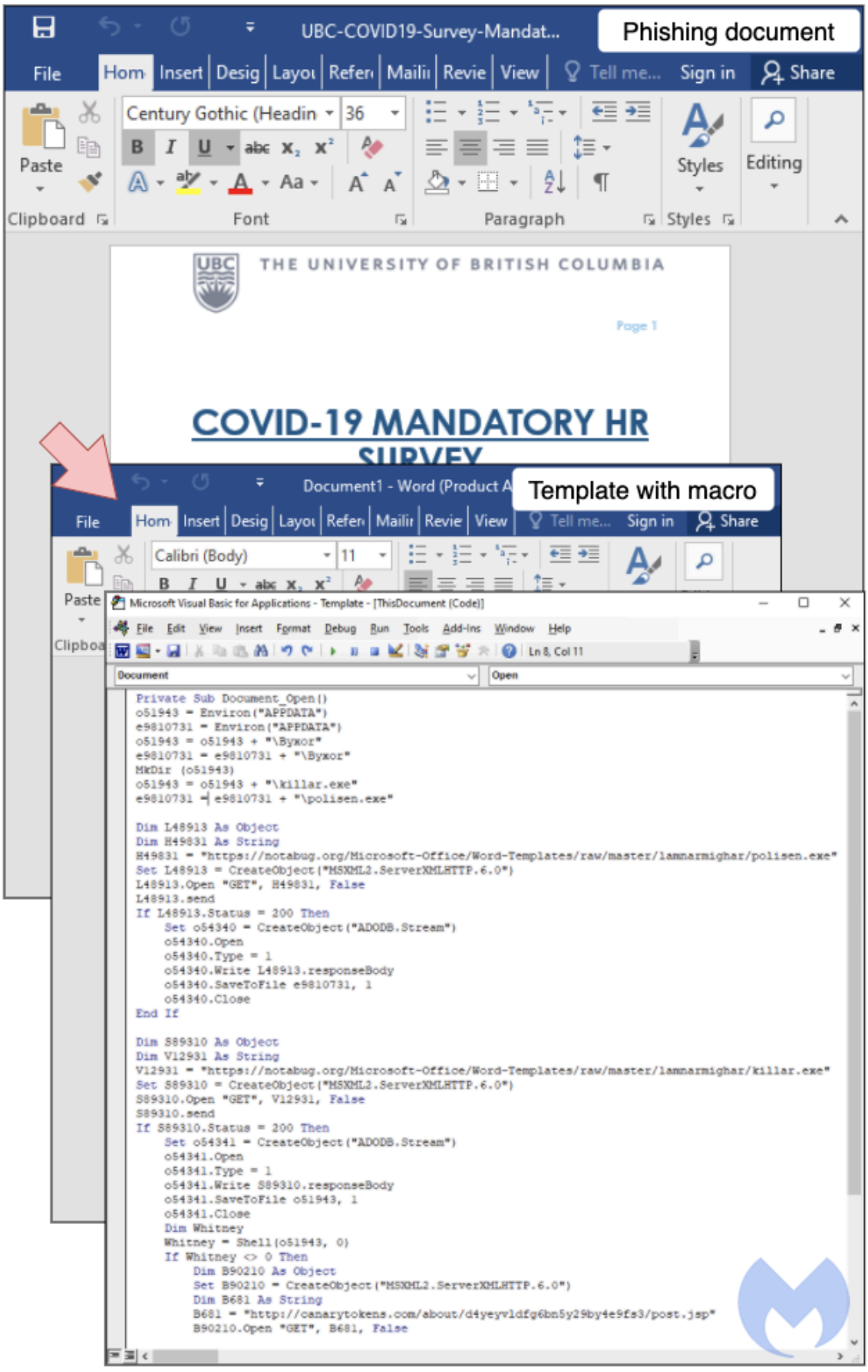
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

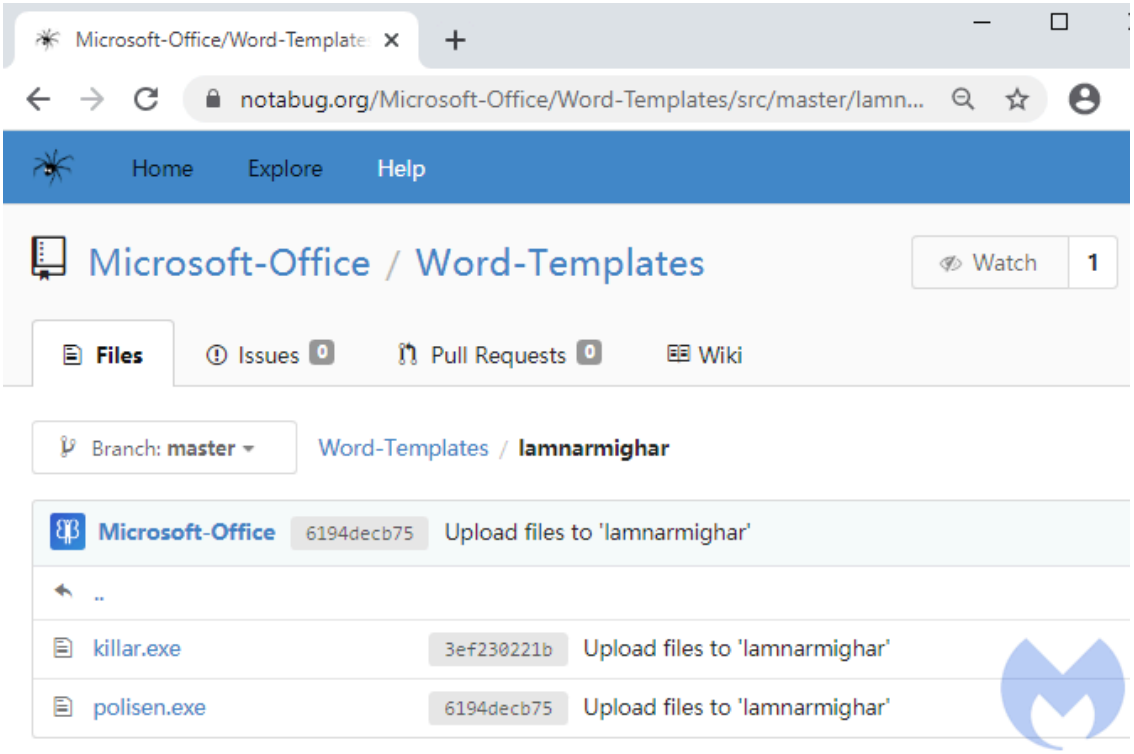
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



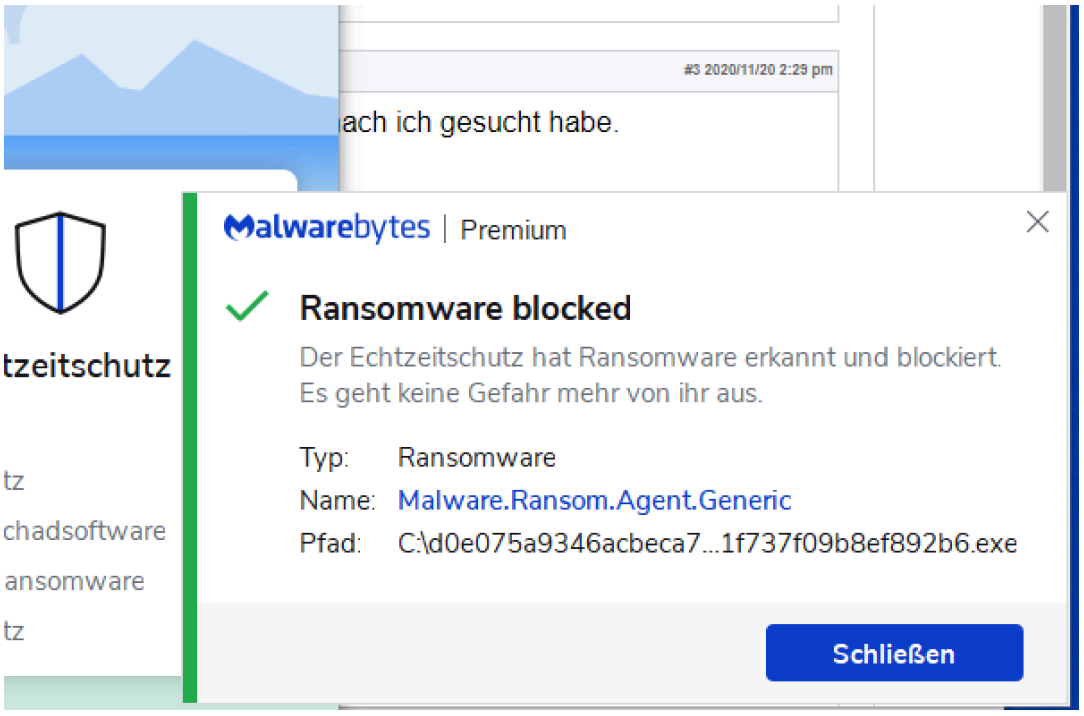
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

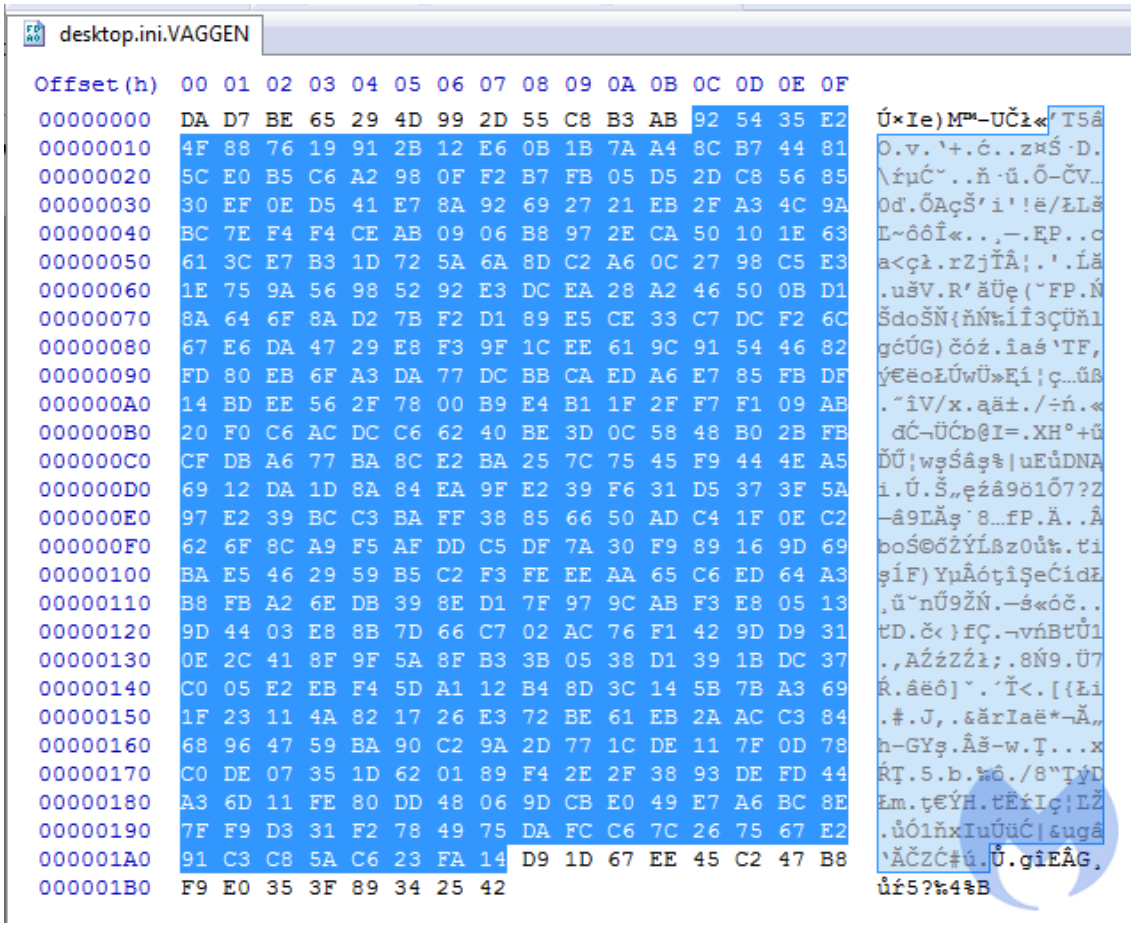
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

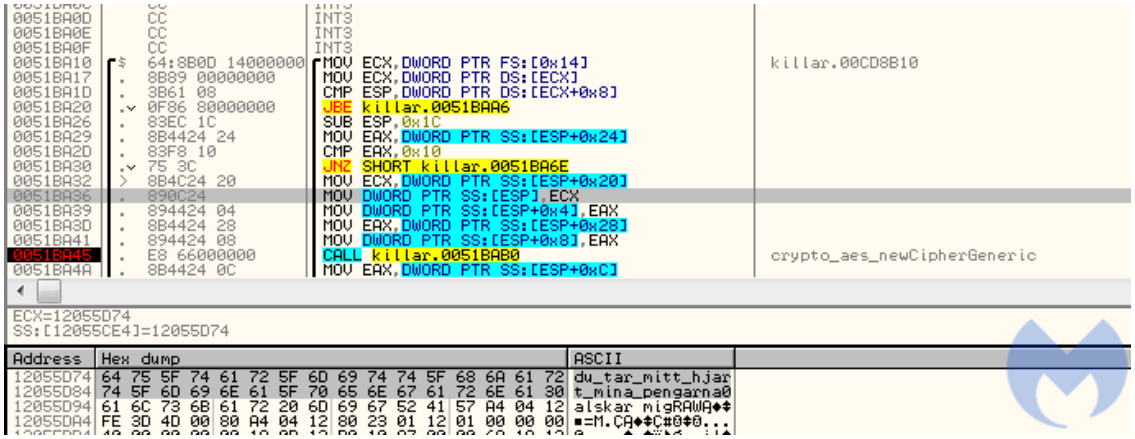


The content of the output file (with .VAGGEN extension) contains:

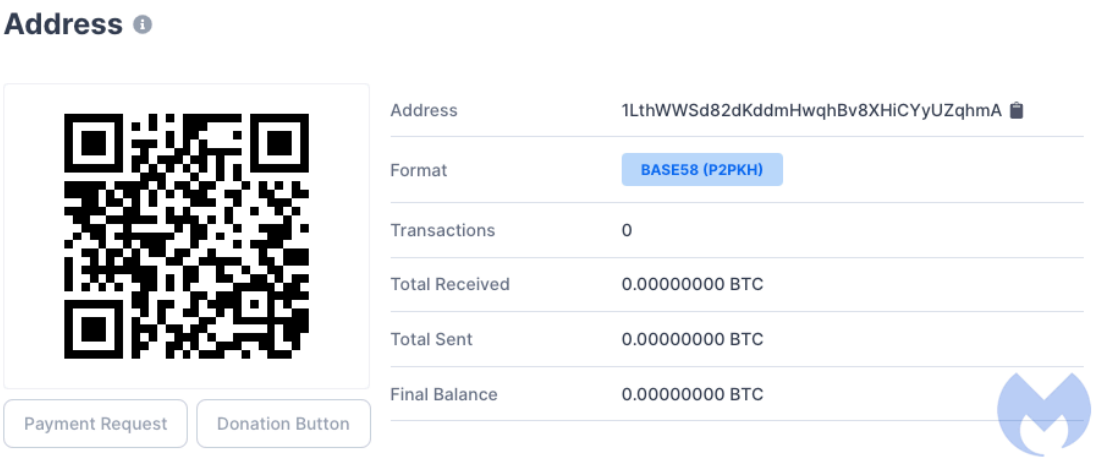
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

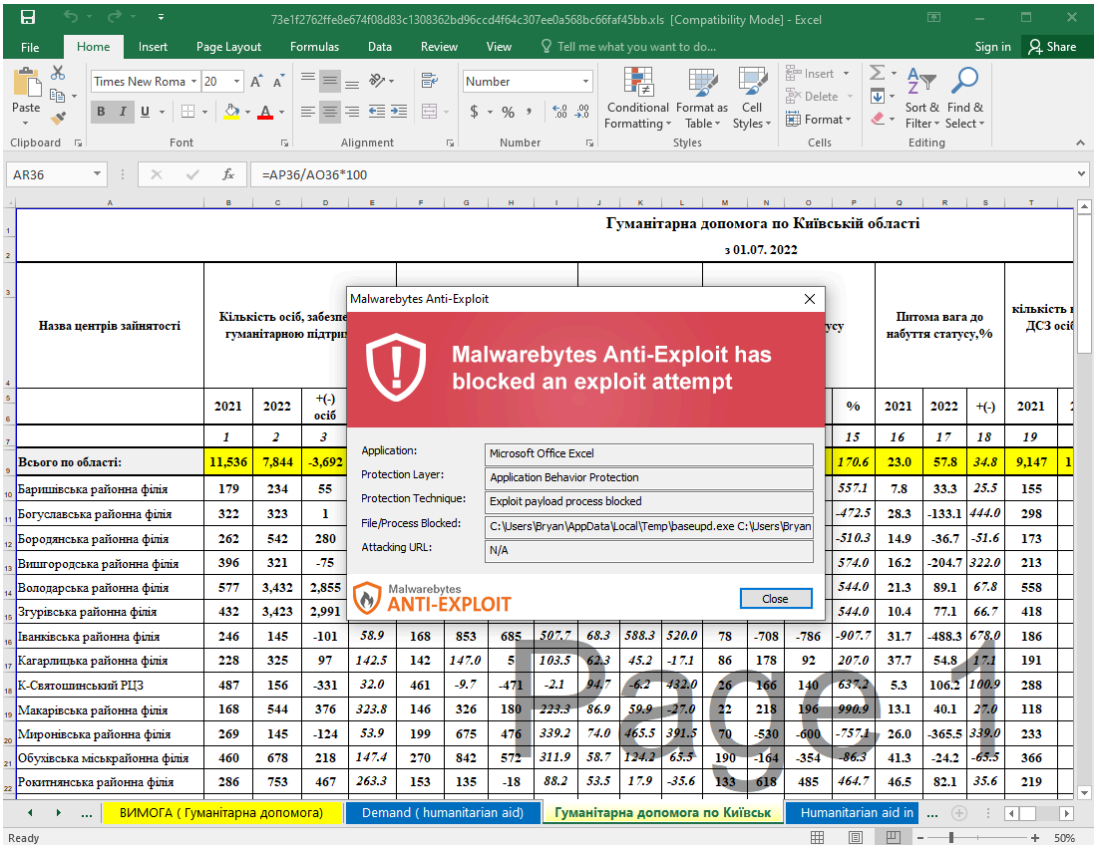
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

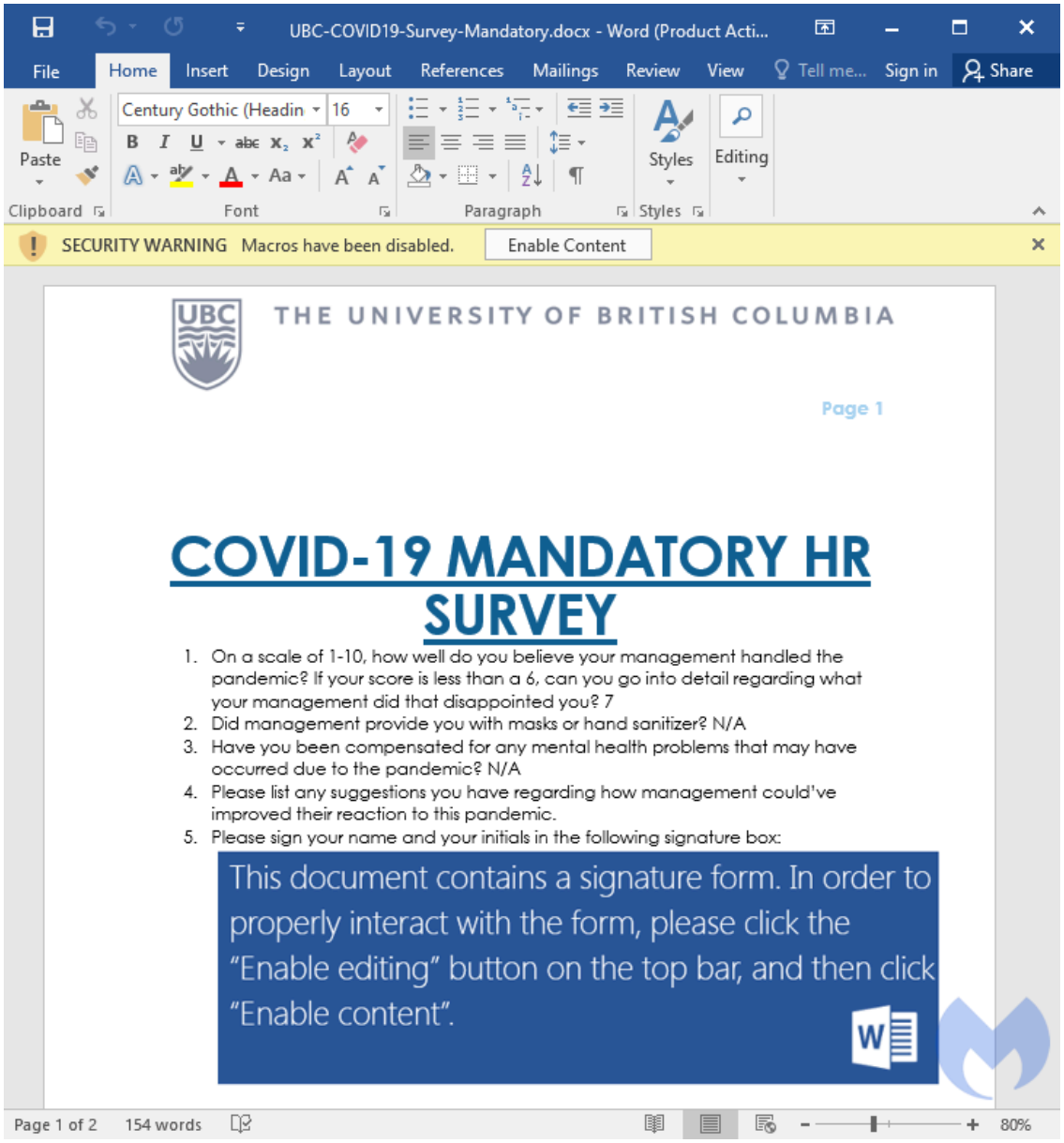
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

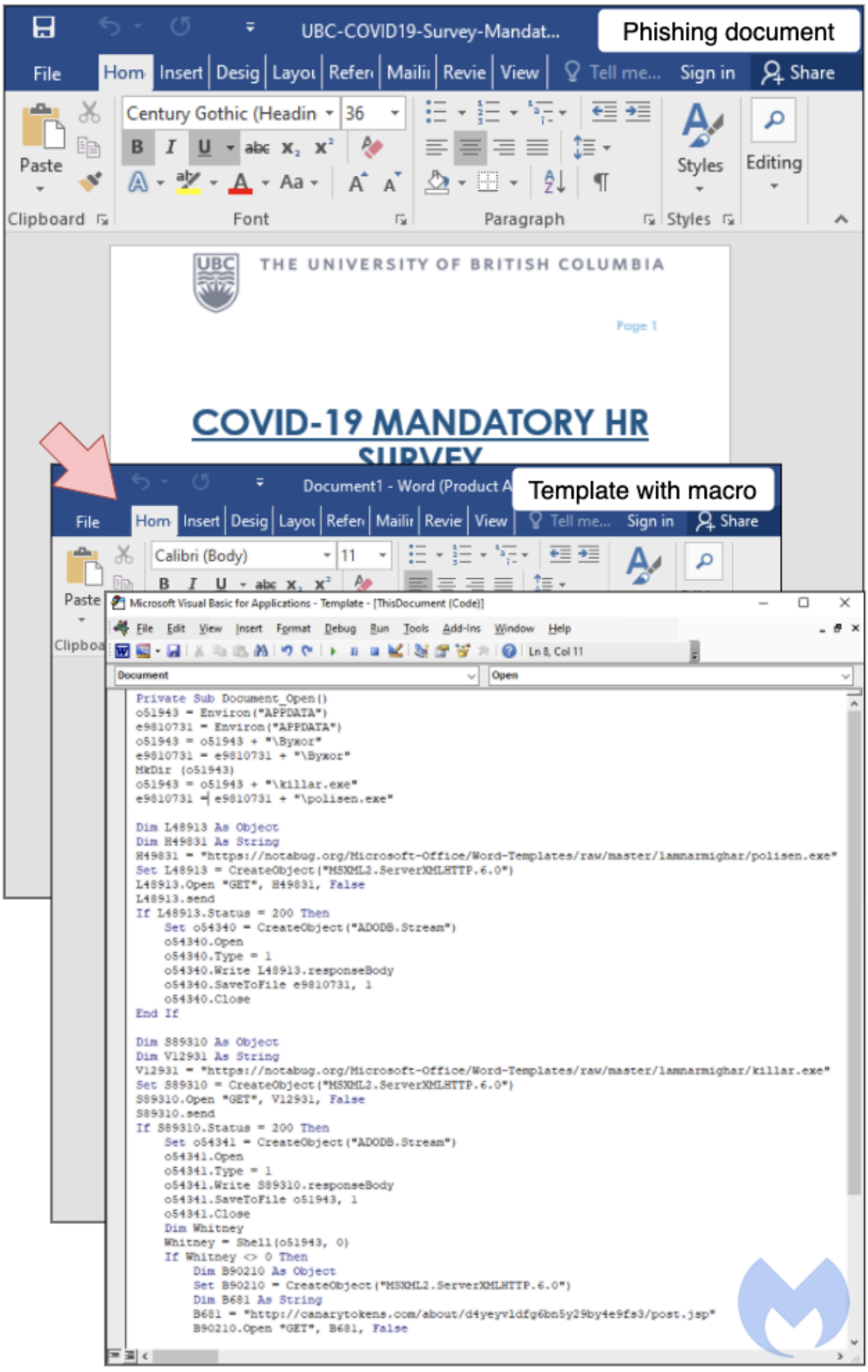
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

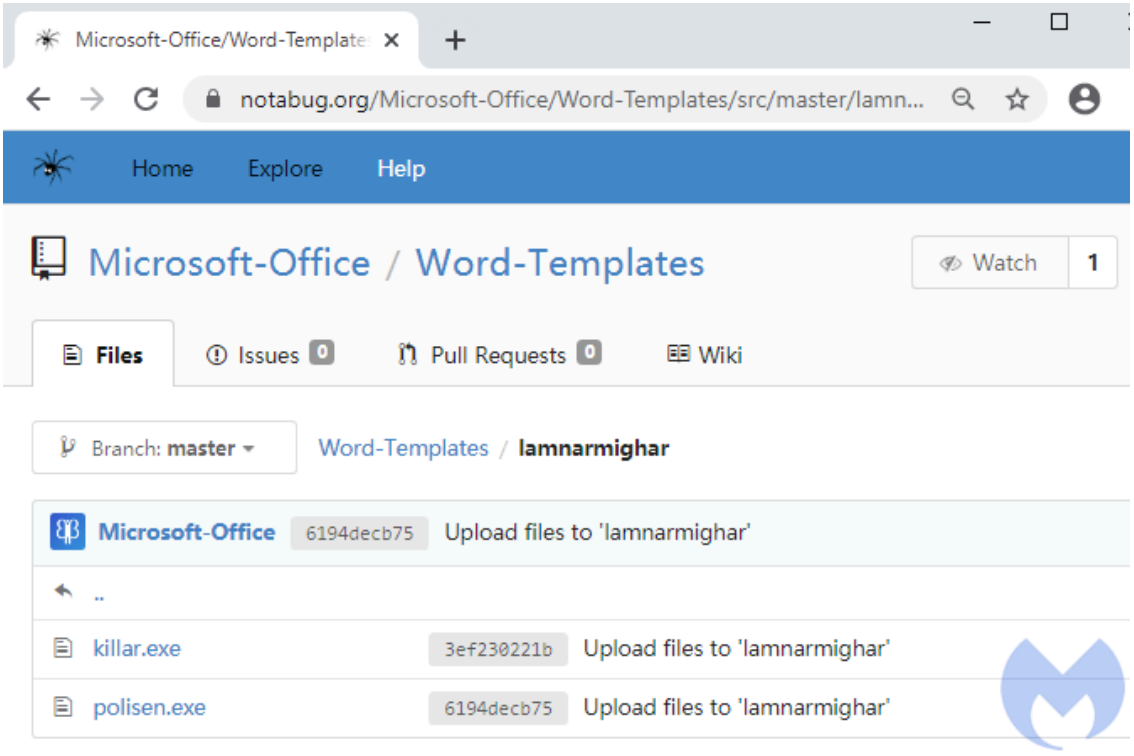
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



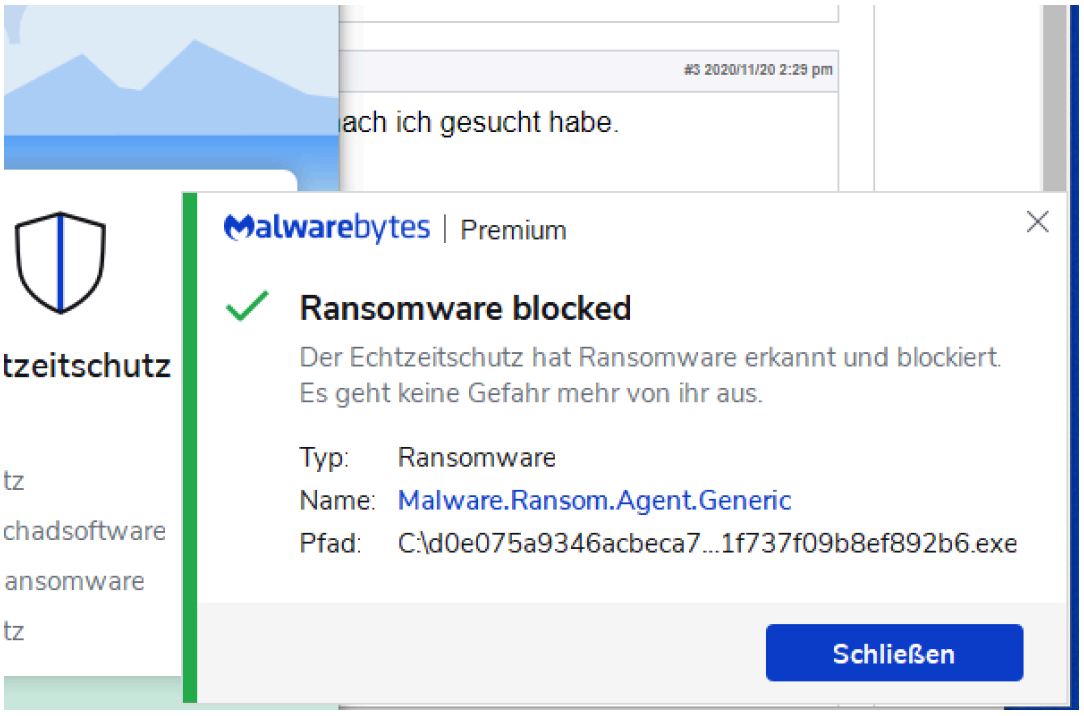
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

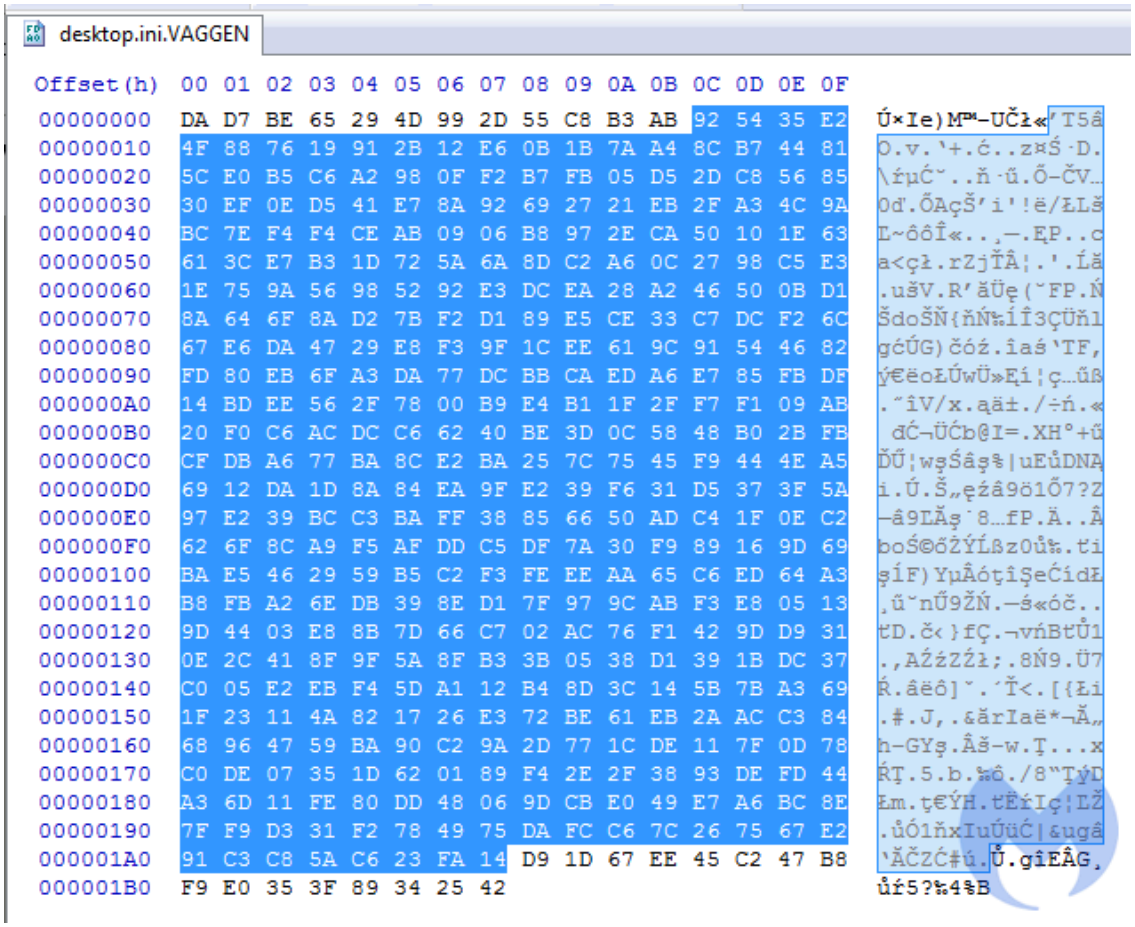
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

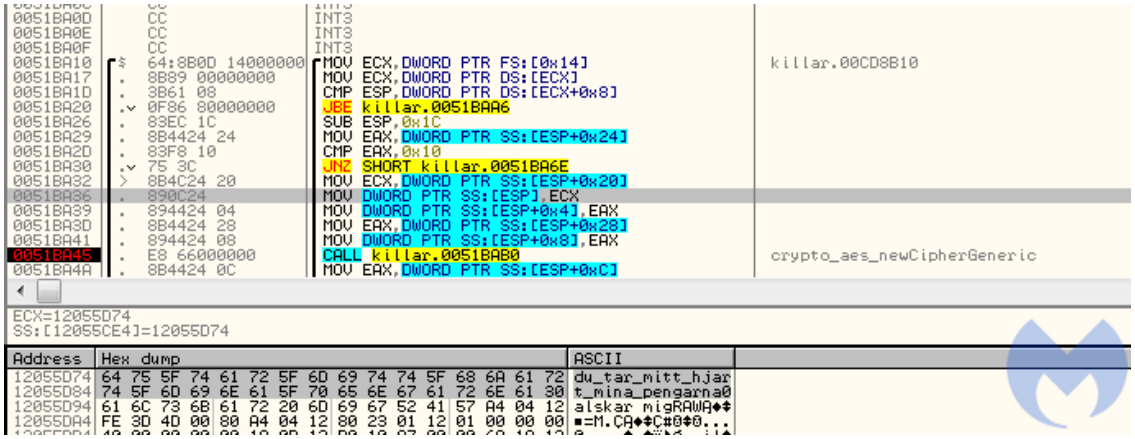


The content of the output file (with .VAGGEN extension) contains:

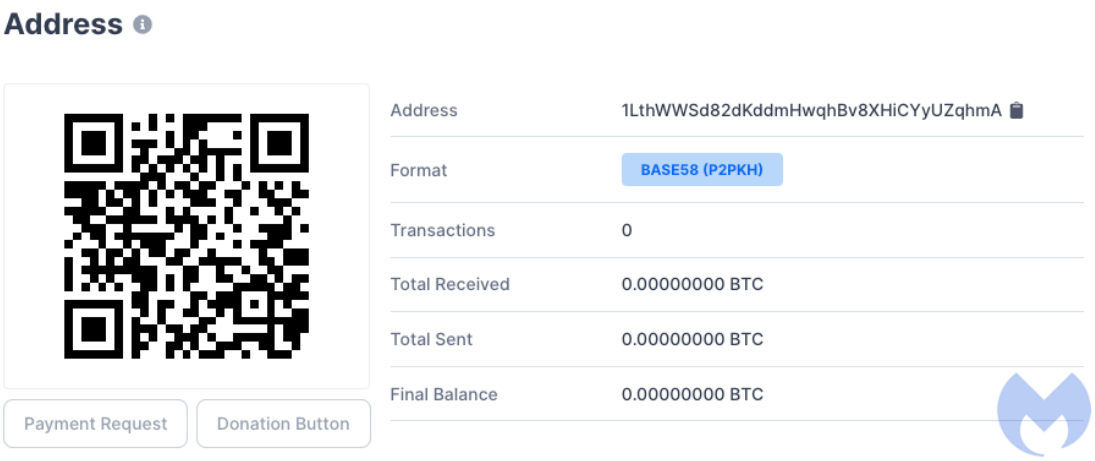
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

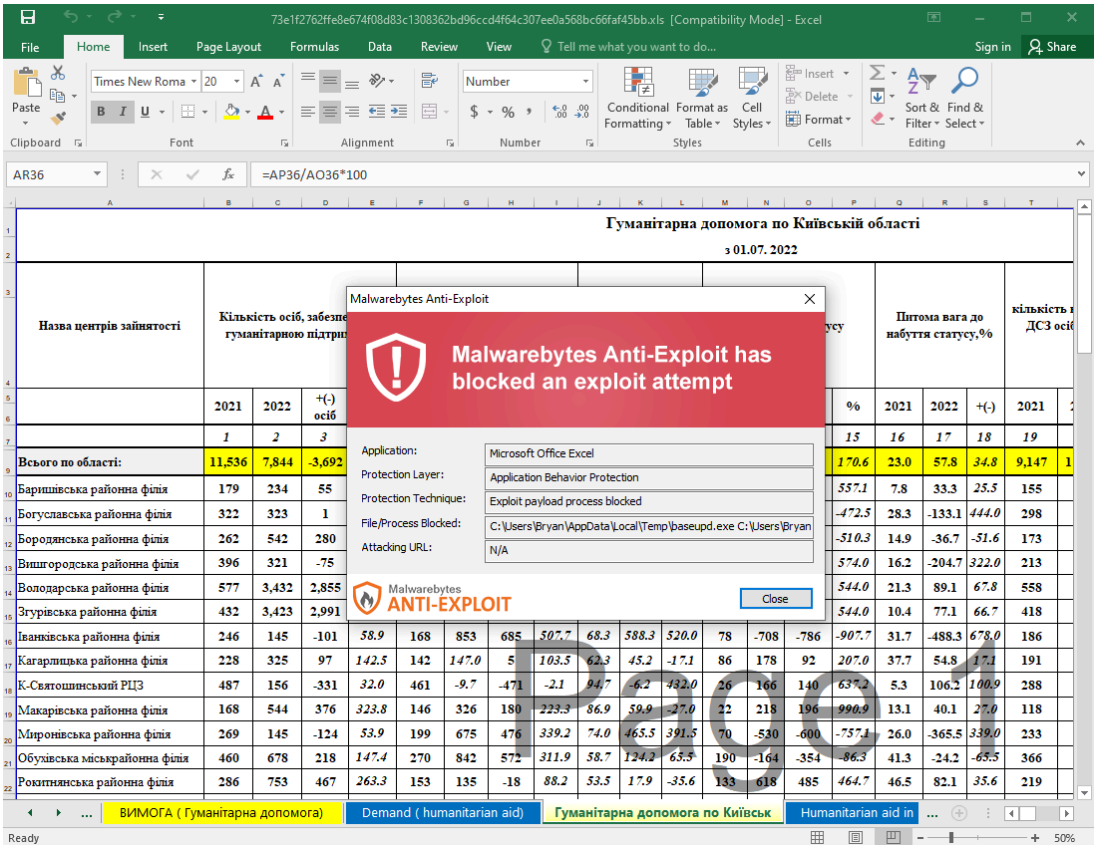
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

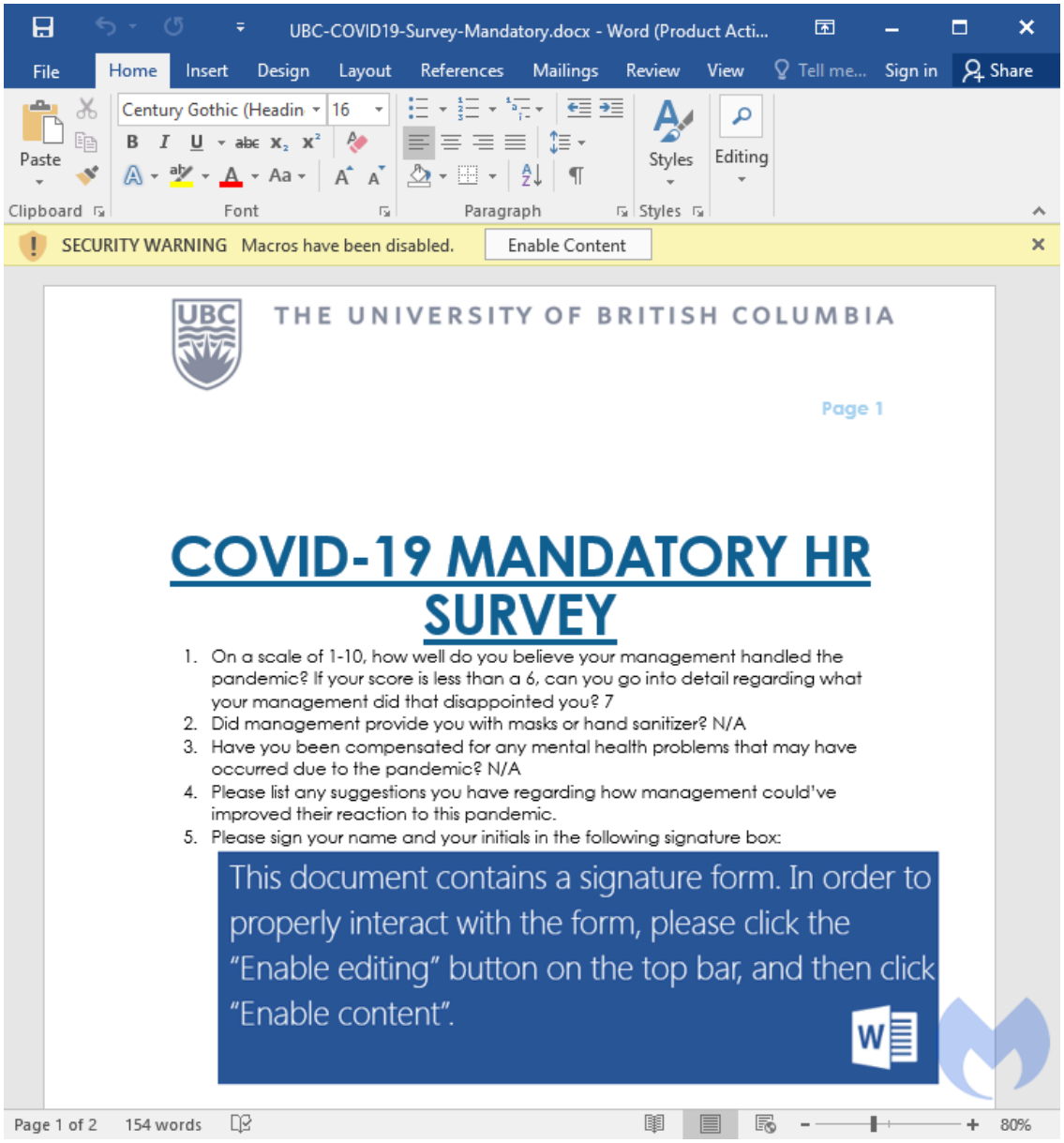
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

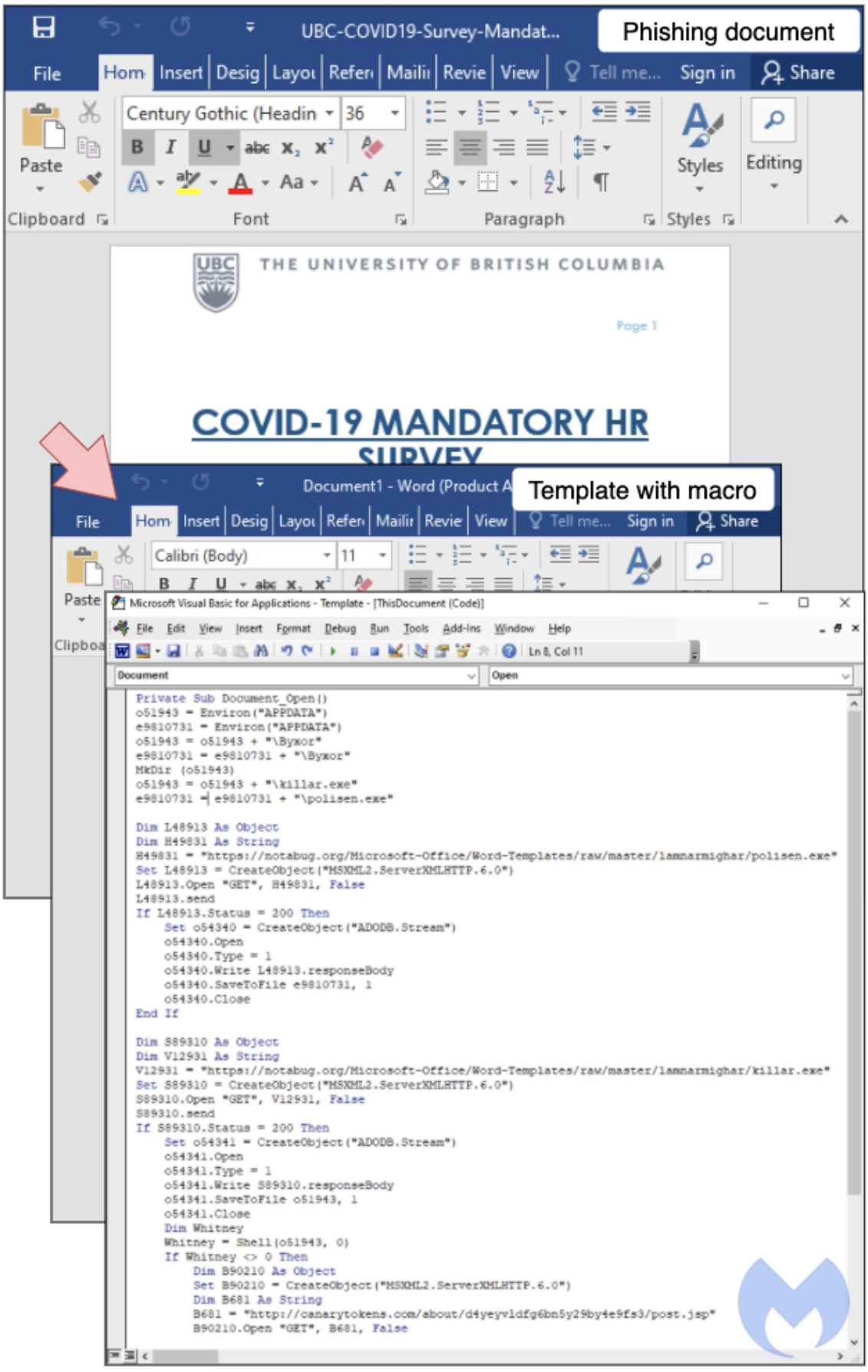
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

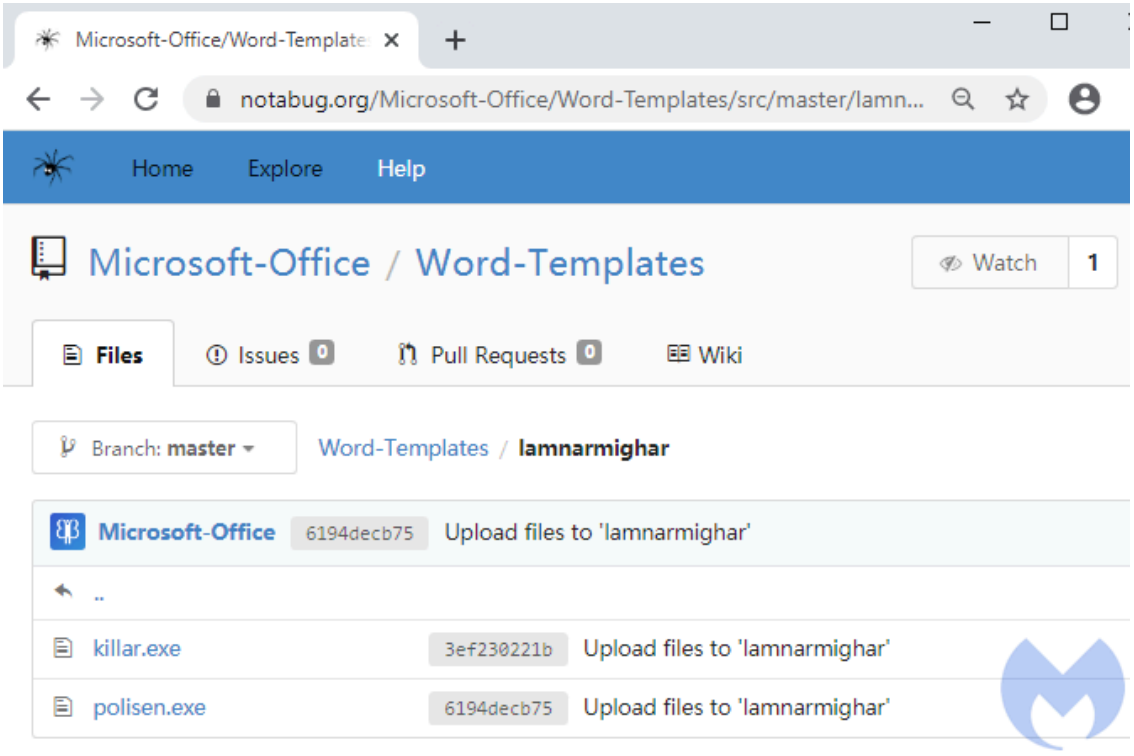
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



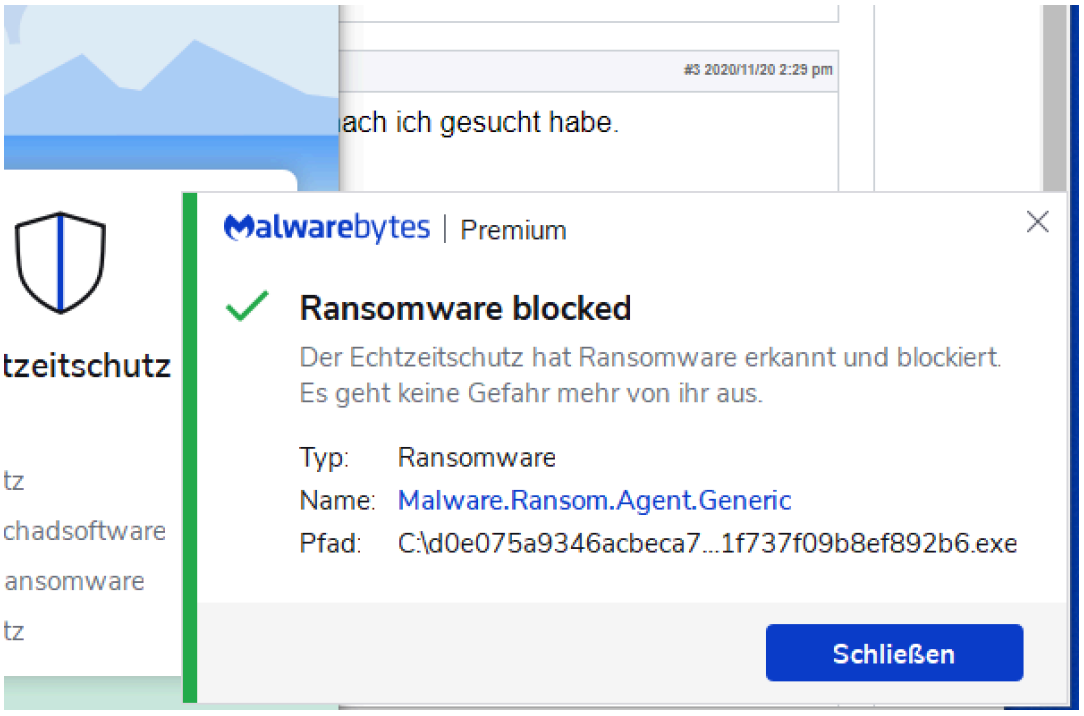
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

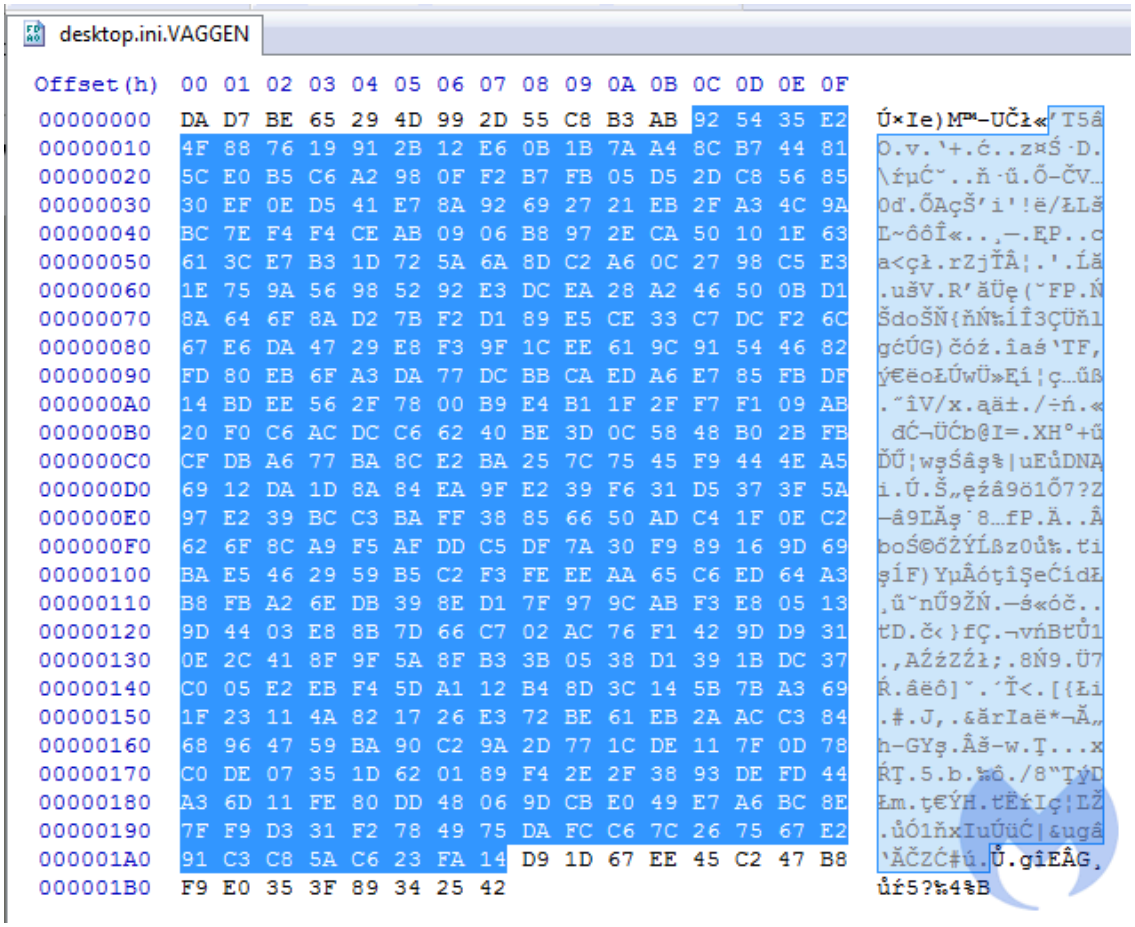
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

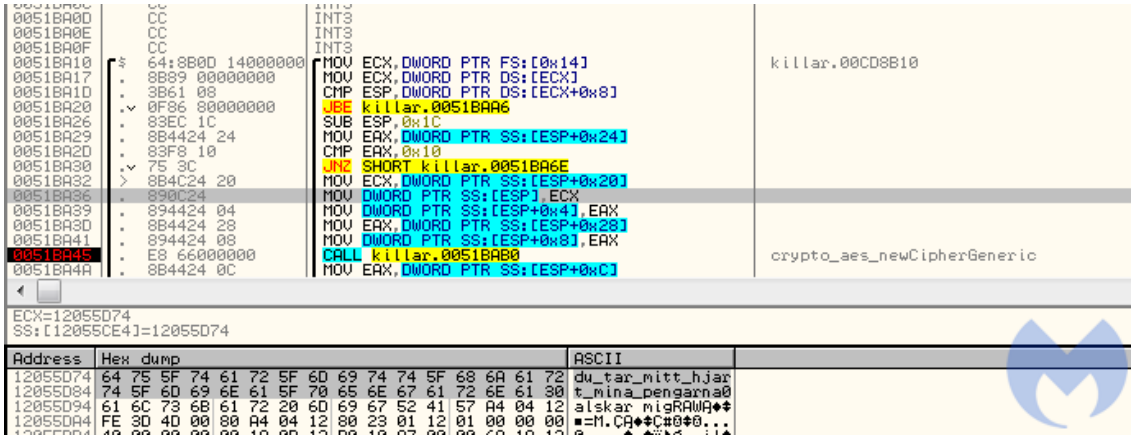


The content of the output file (with .VAGGEN extension) contains:

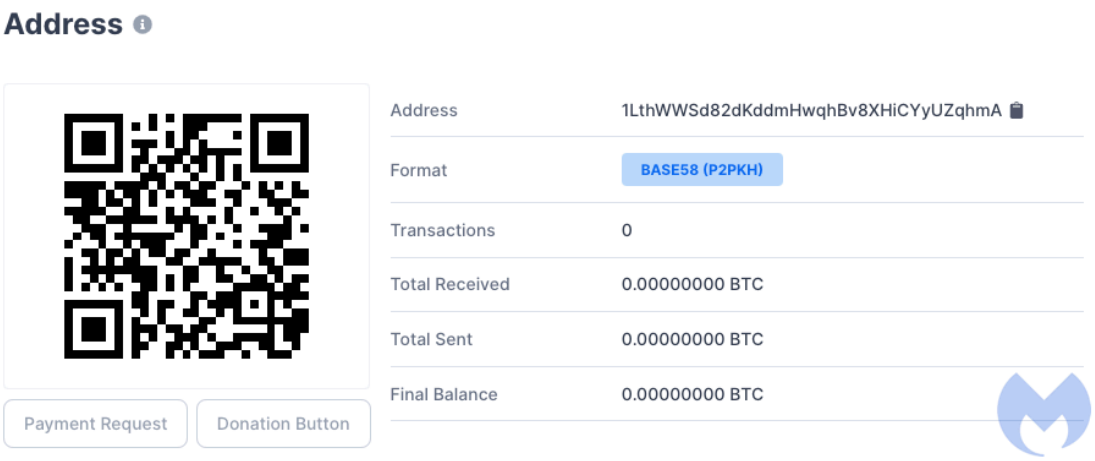
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

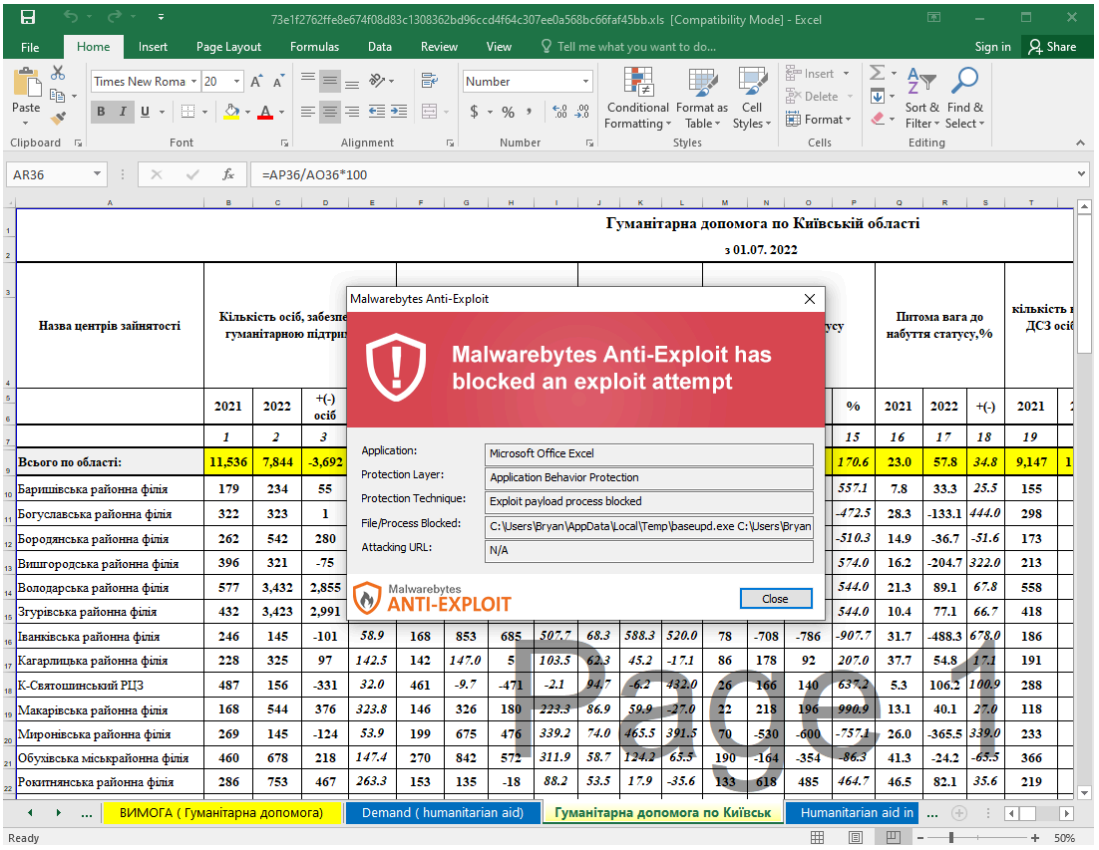
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

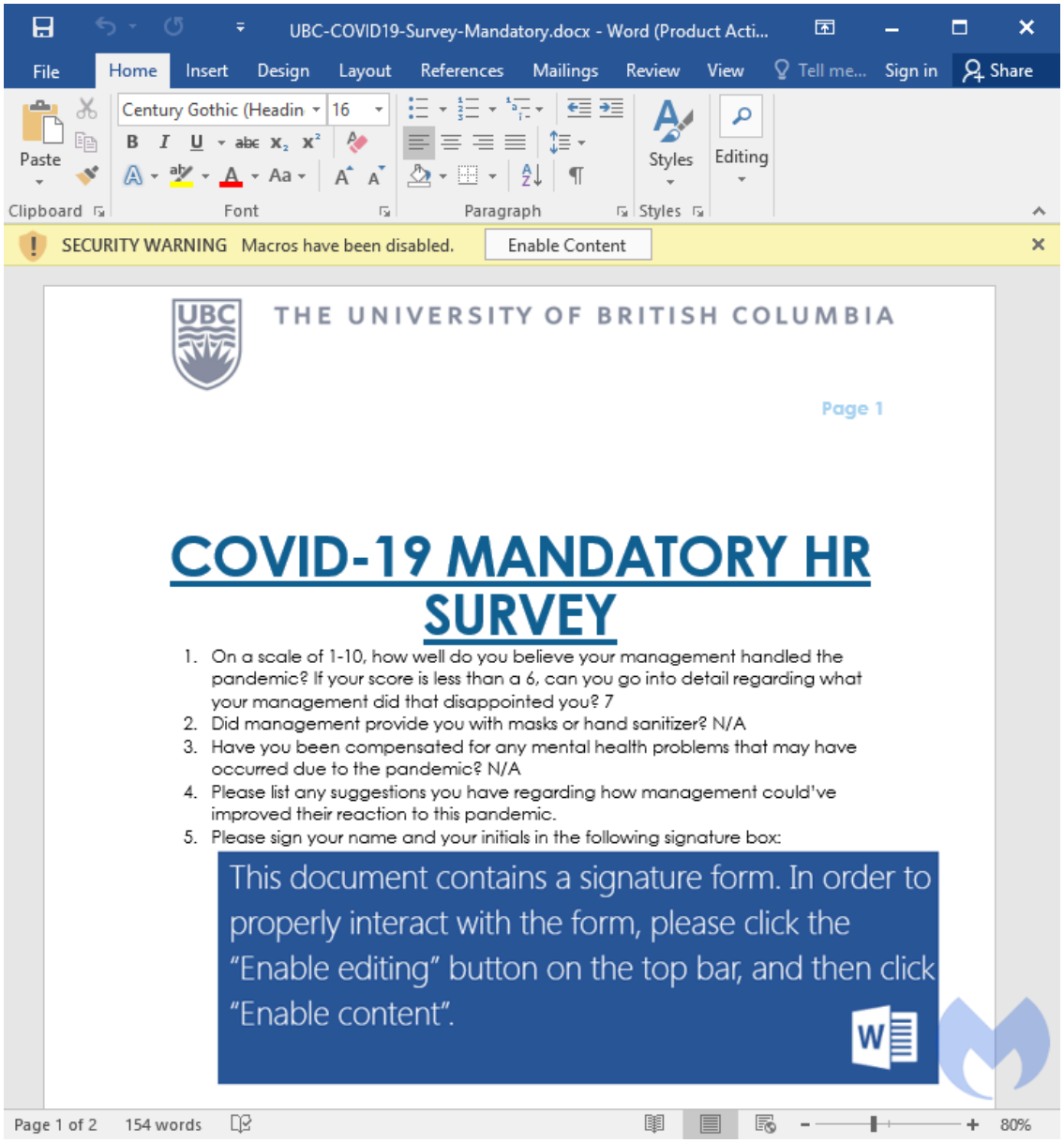
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

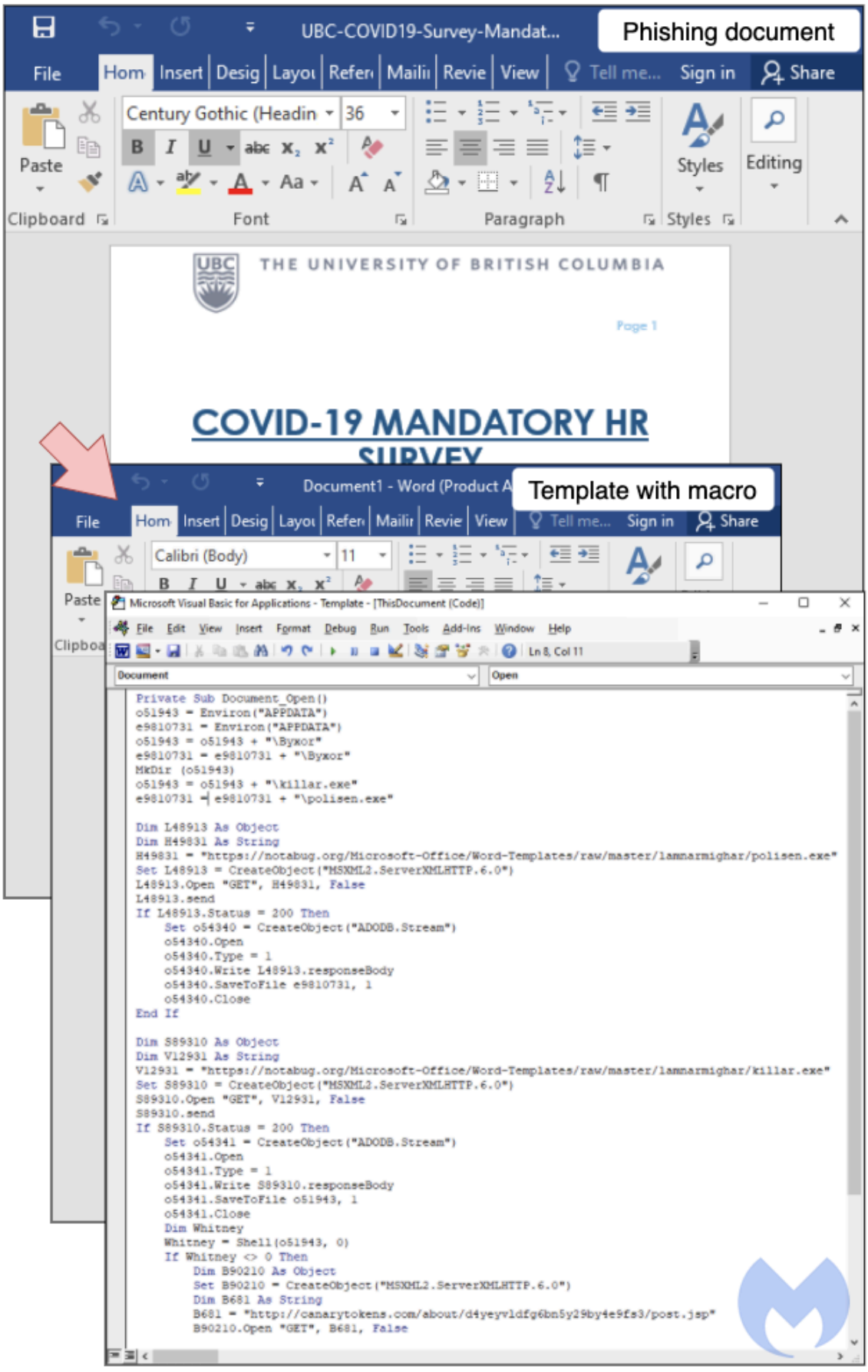
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

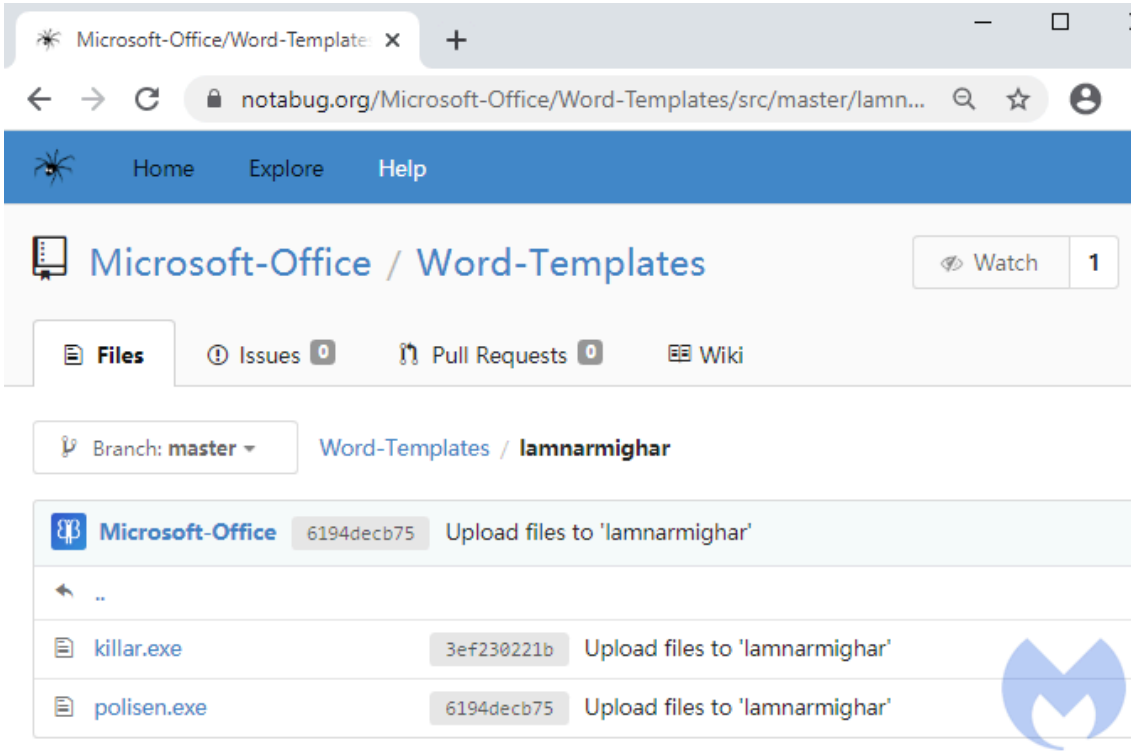
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



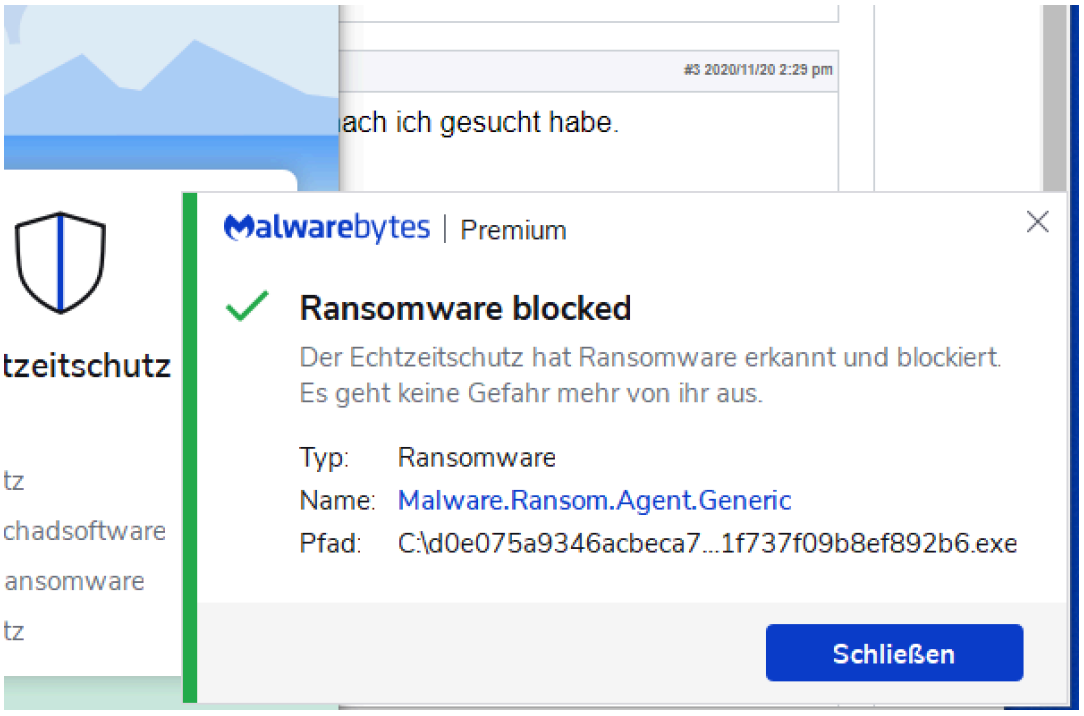
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

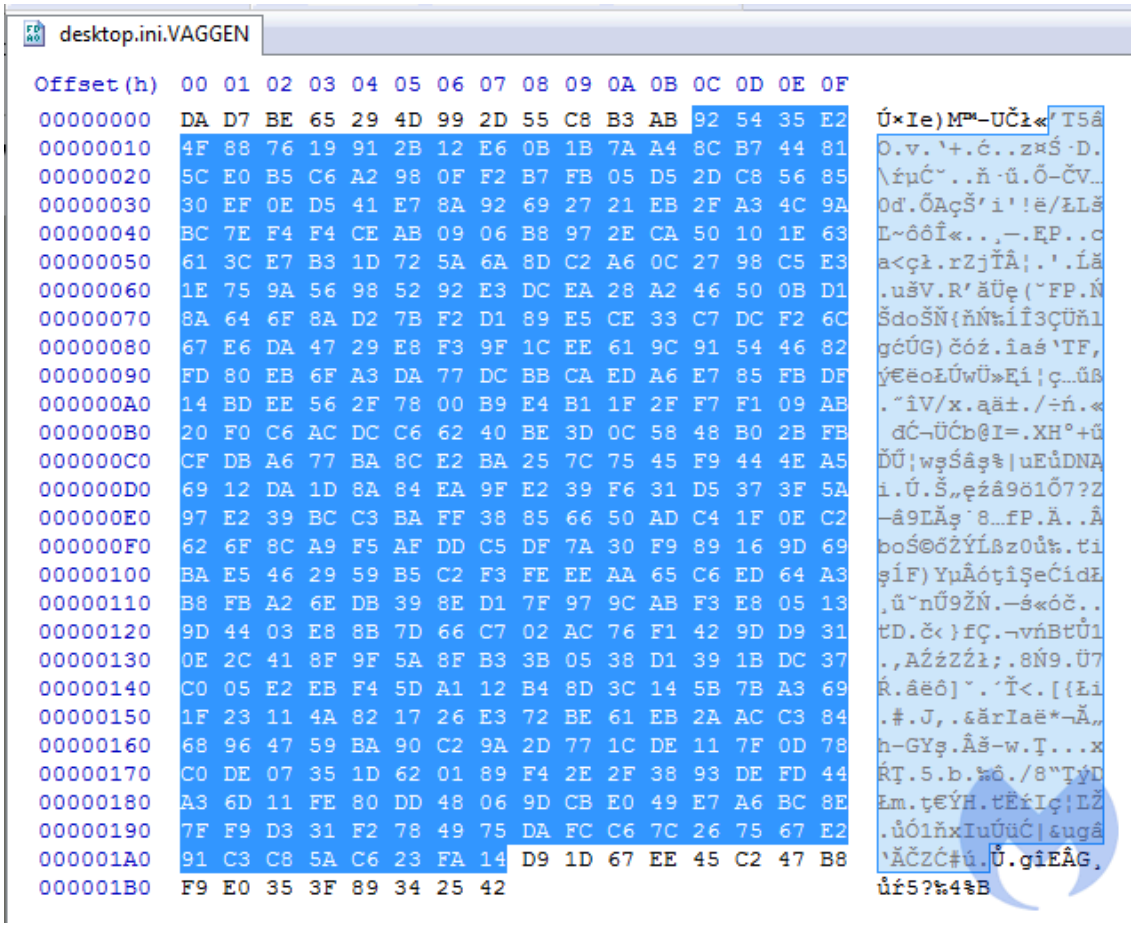
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

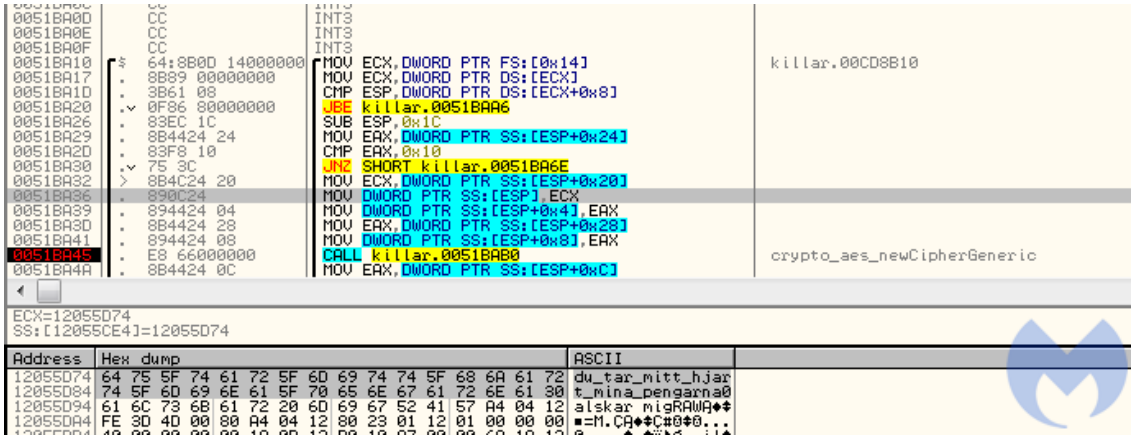


The content of the output file (with .VAGGEN extension) contains:

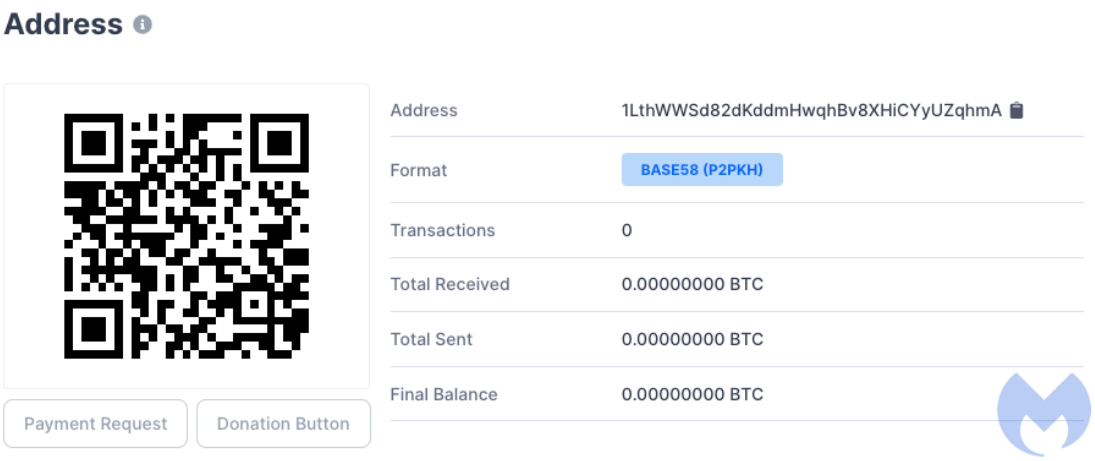
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

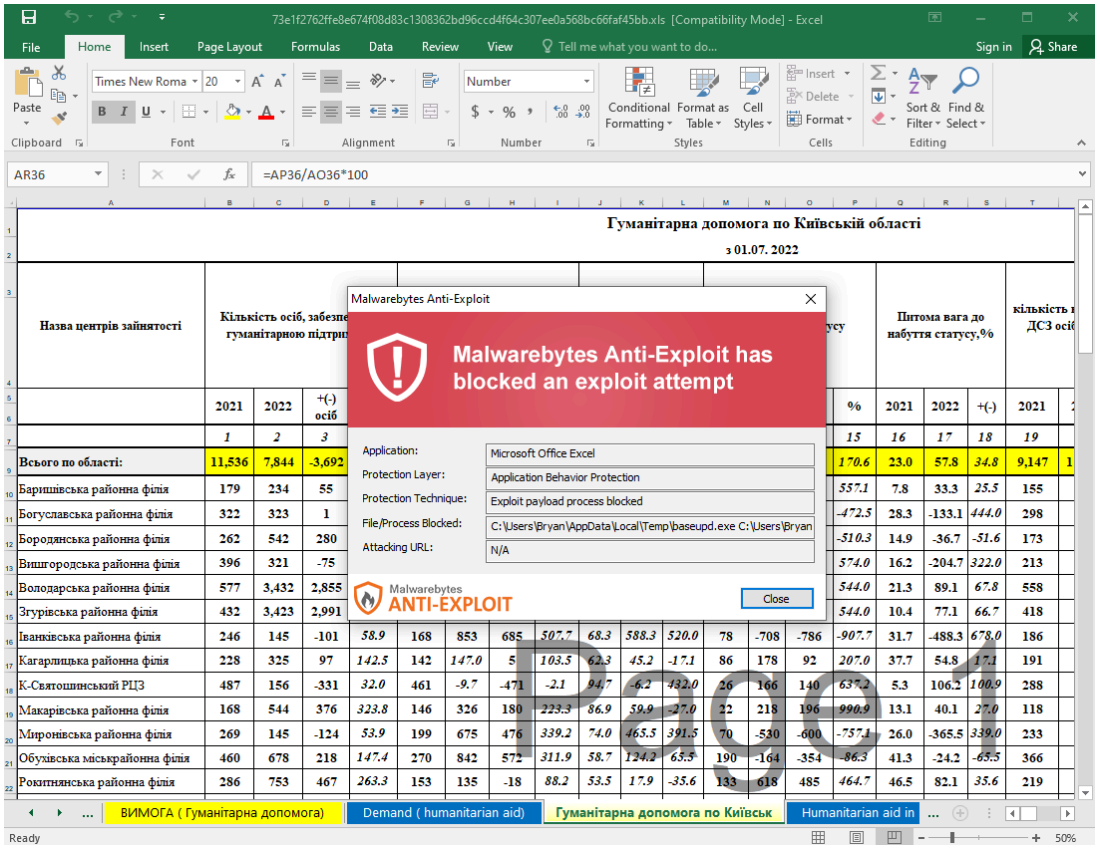
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

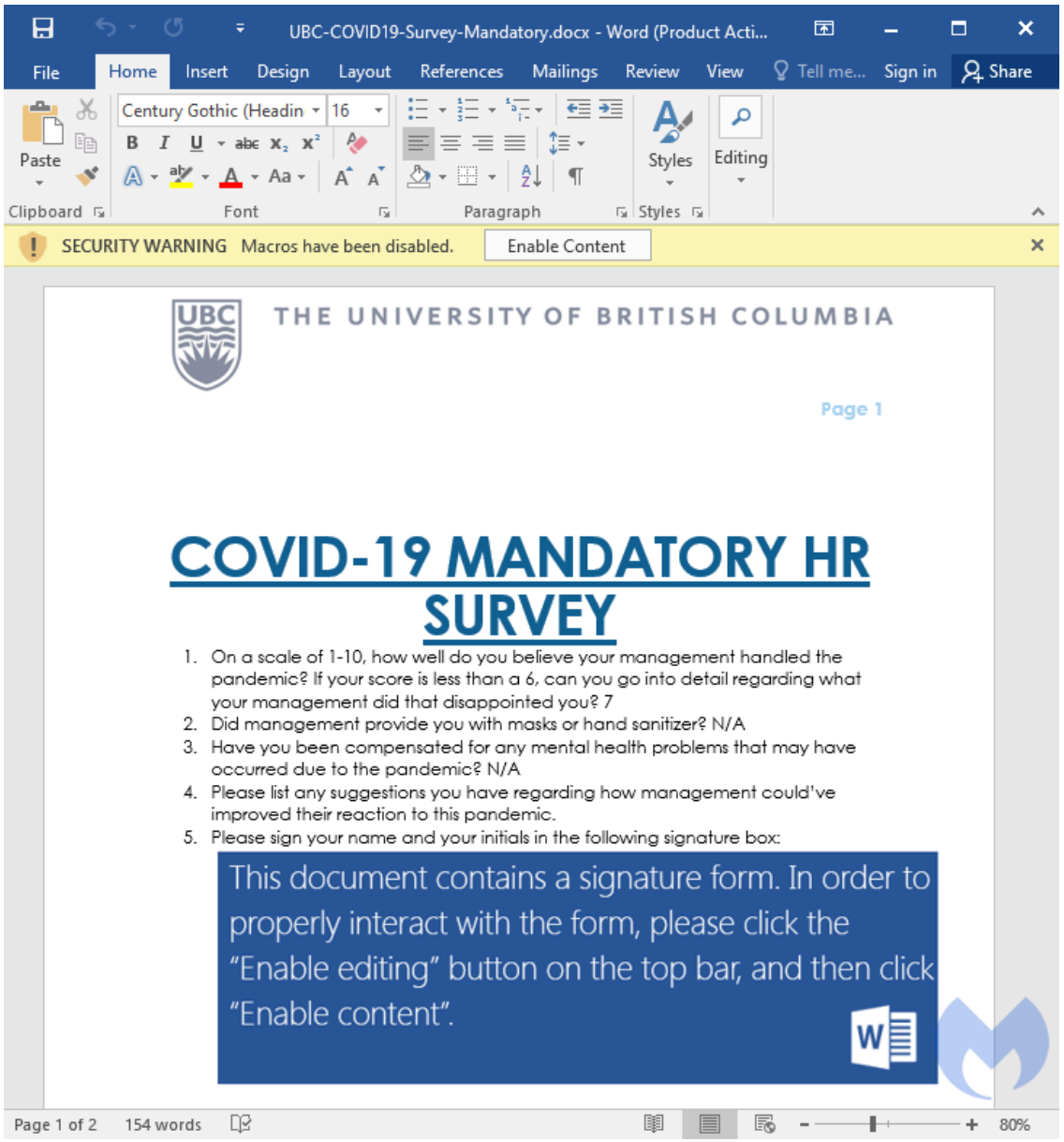
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

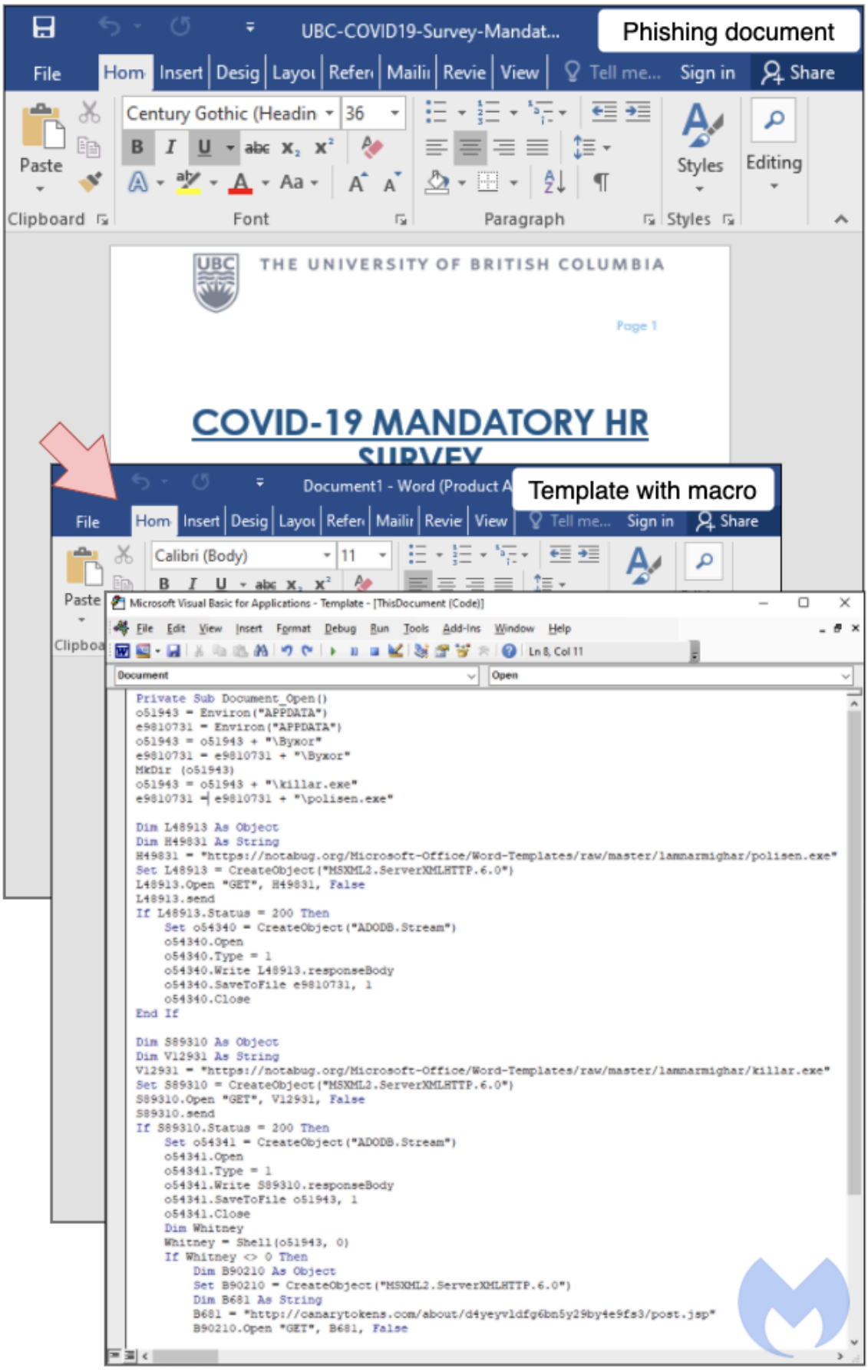
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

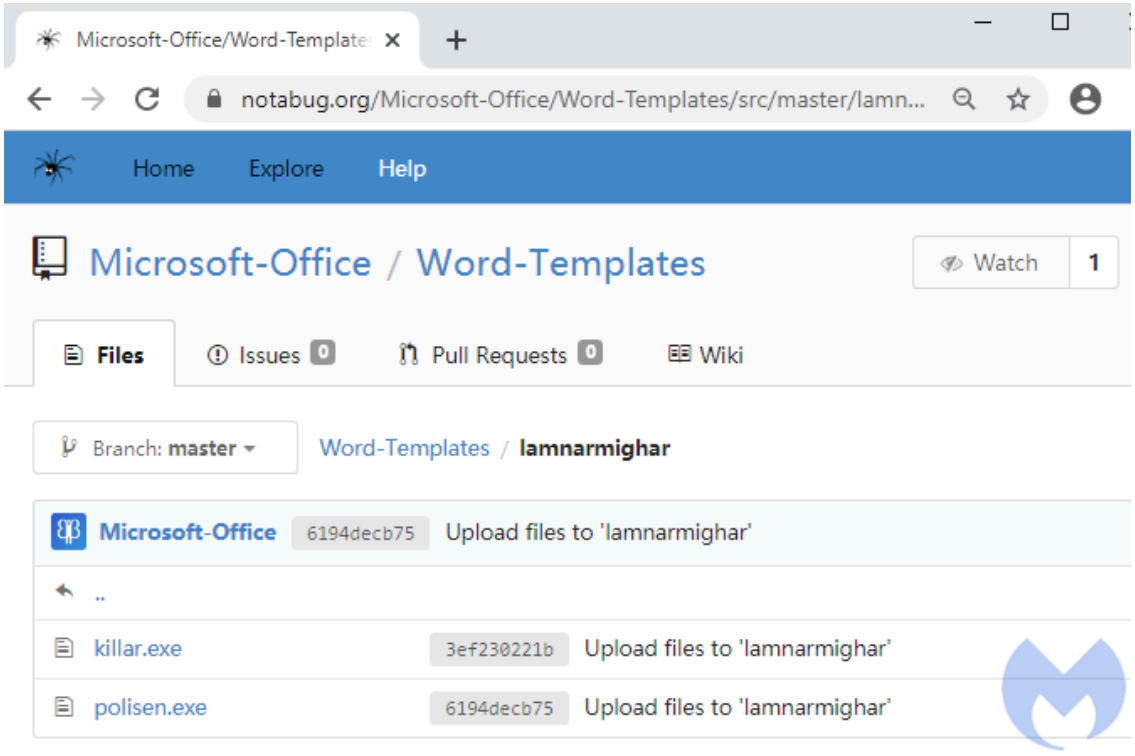
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



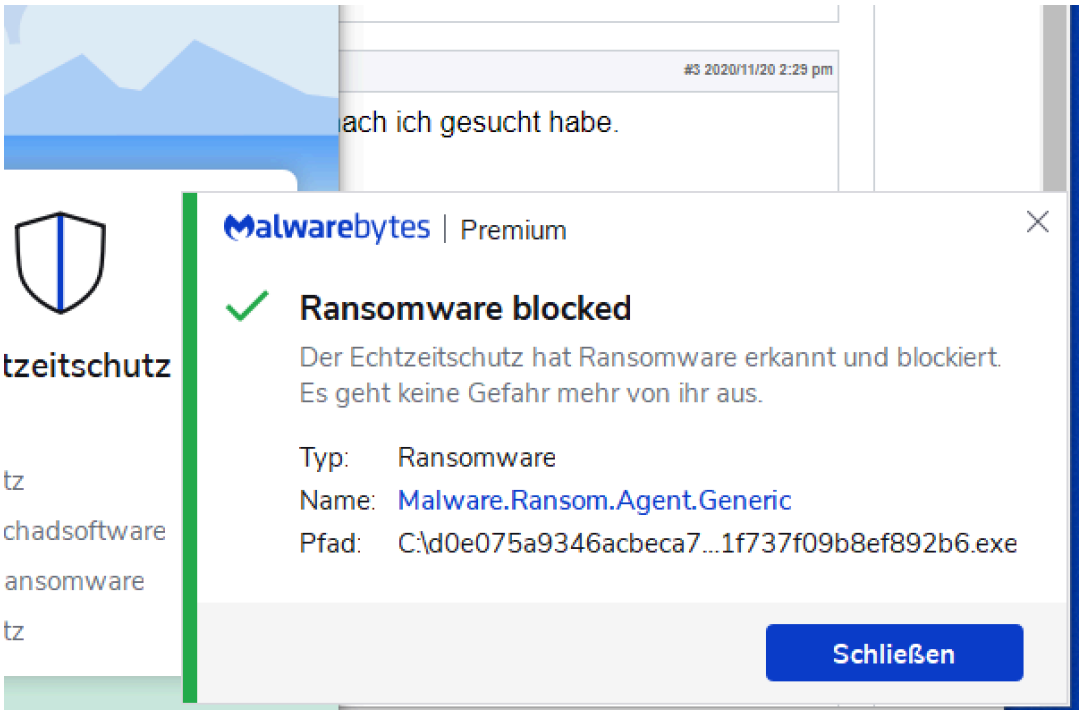
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

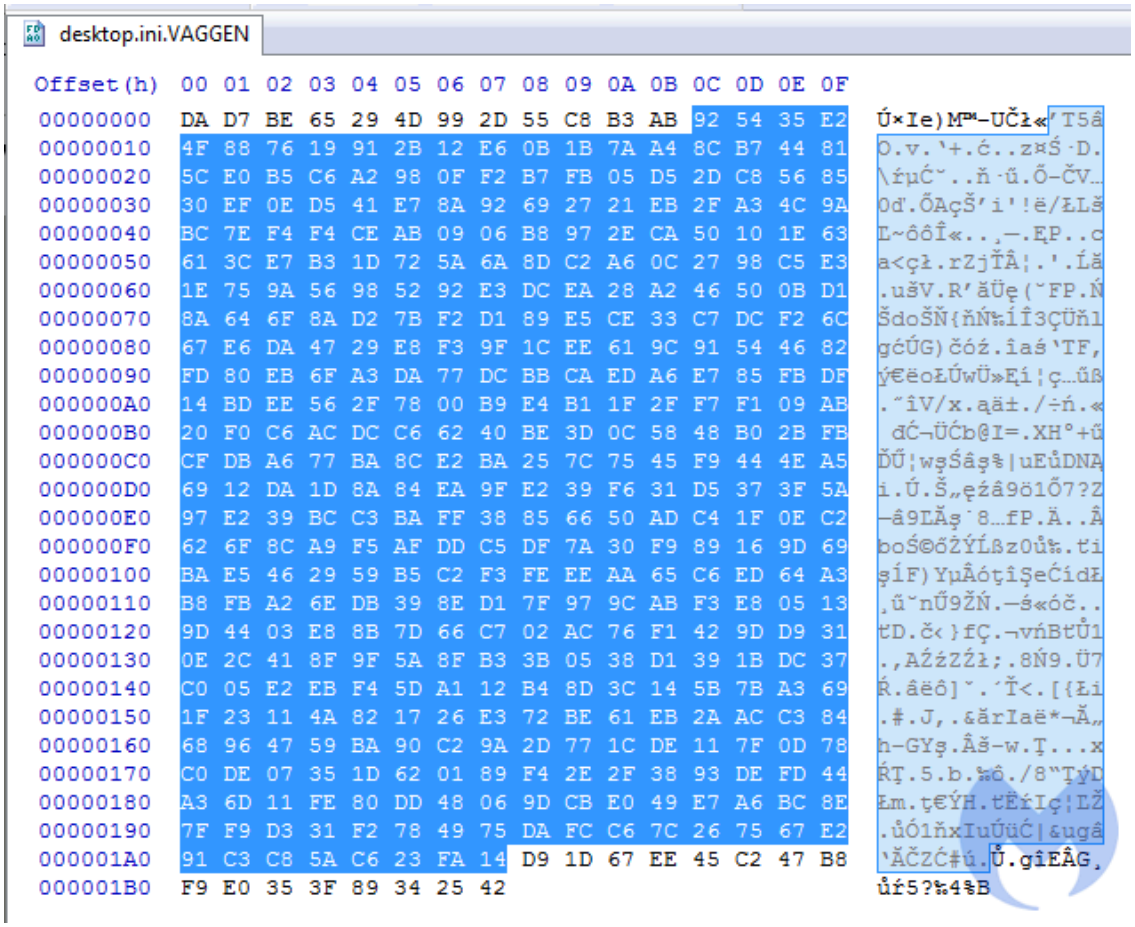
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

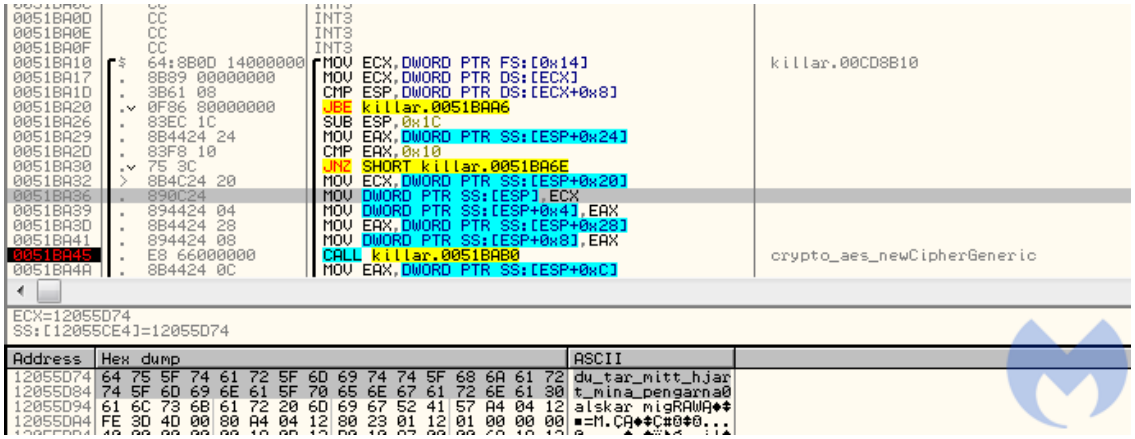


The content of the output file (with .VAGGEN extension) contains:

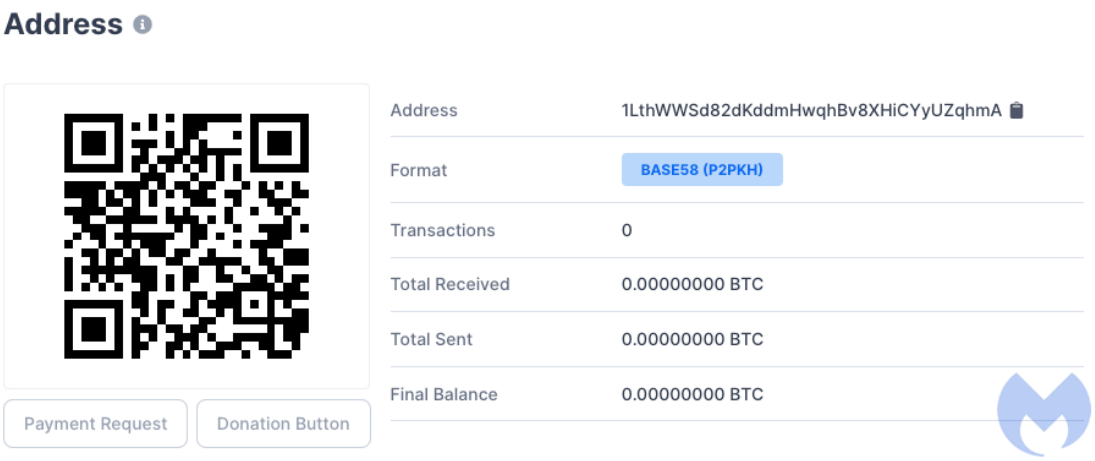
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

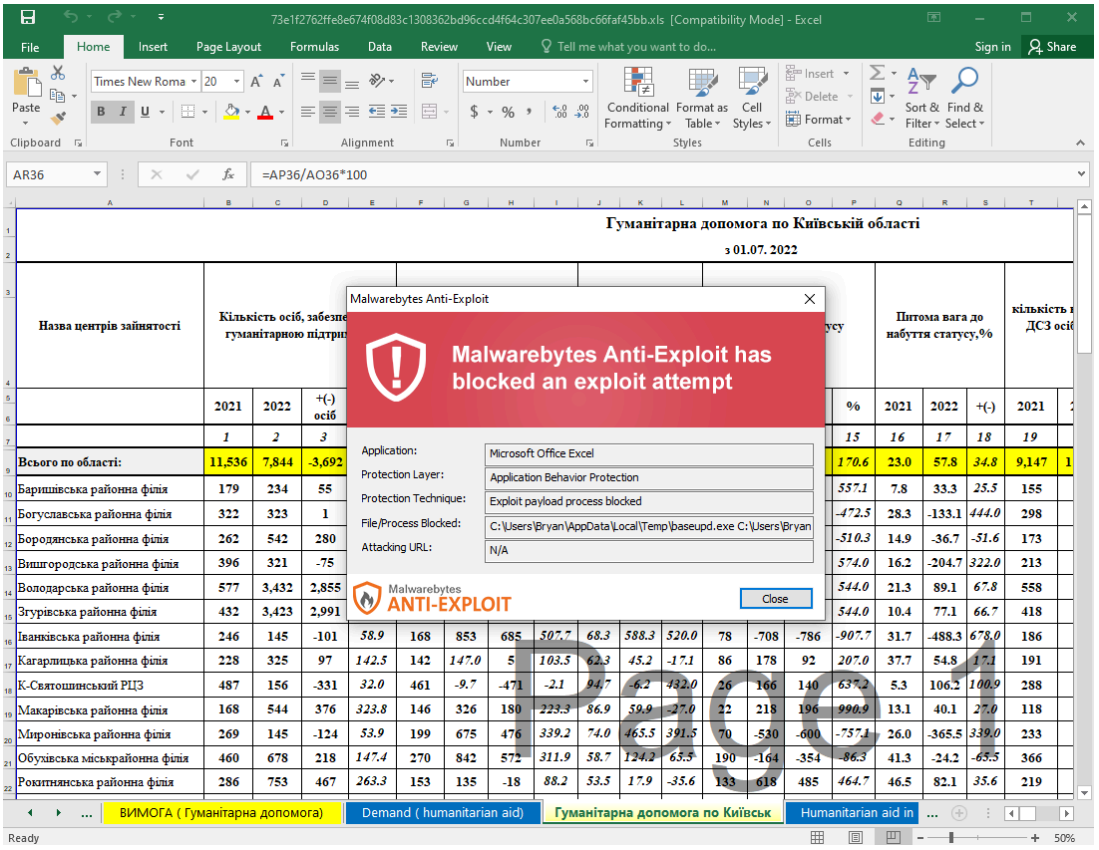
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

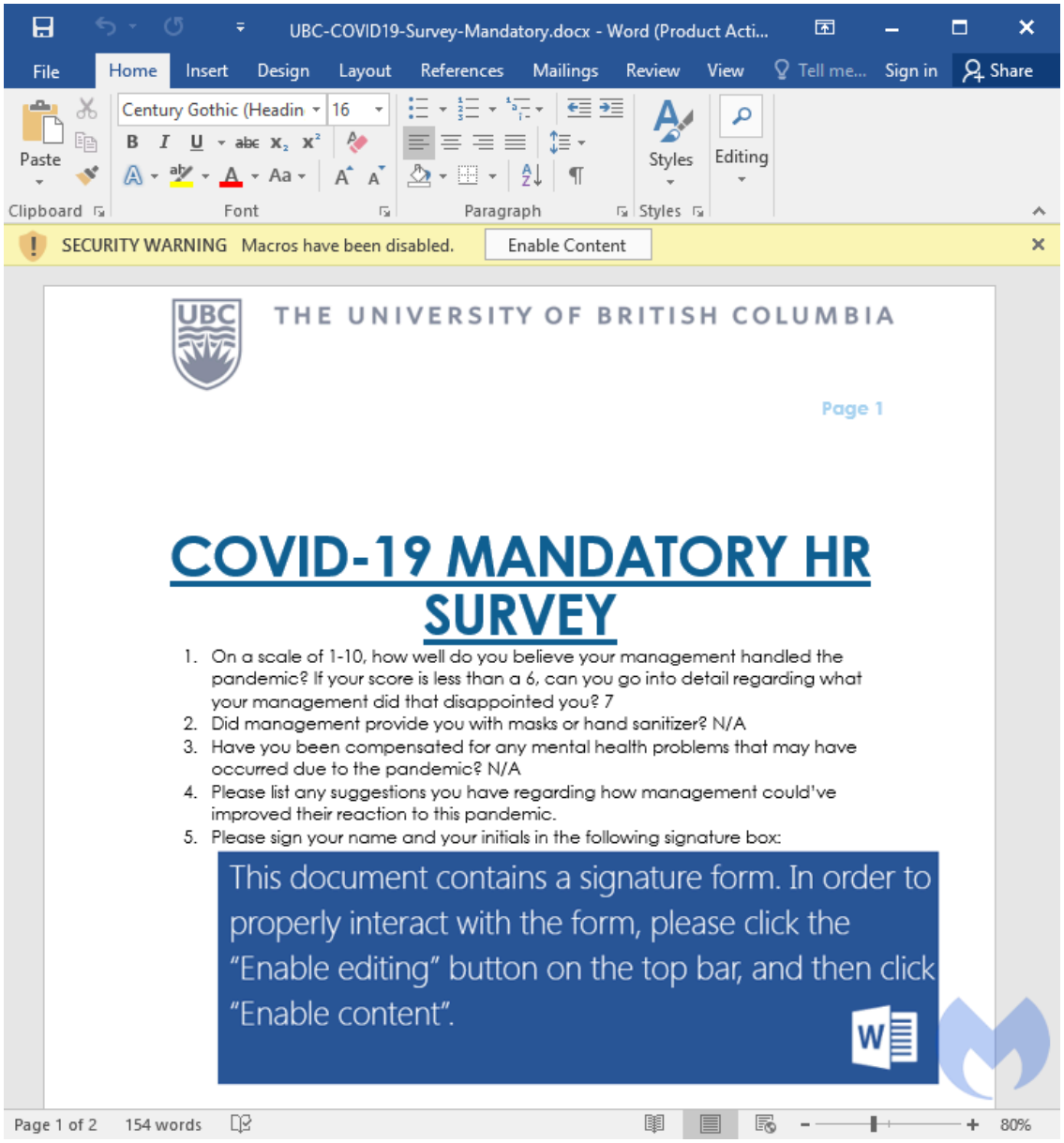
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

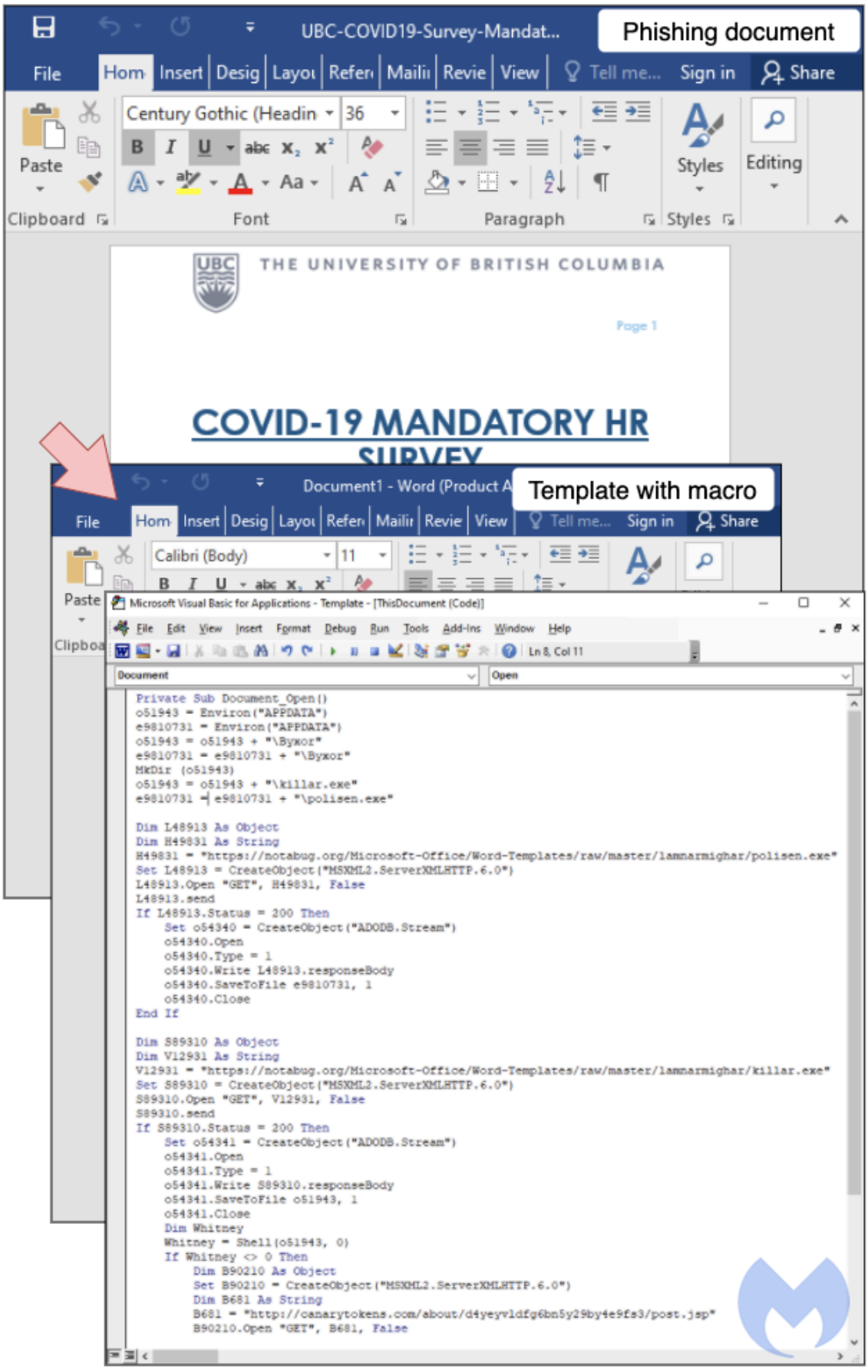
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

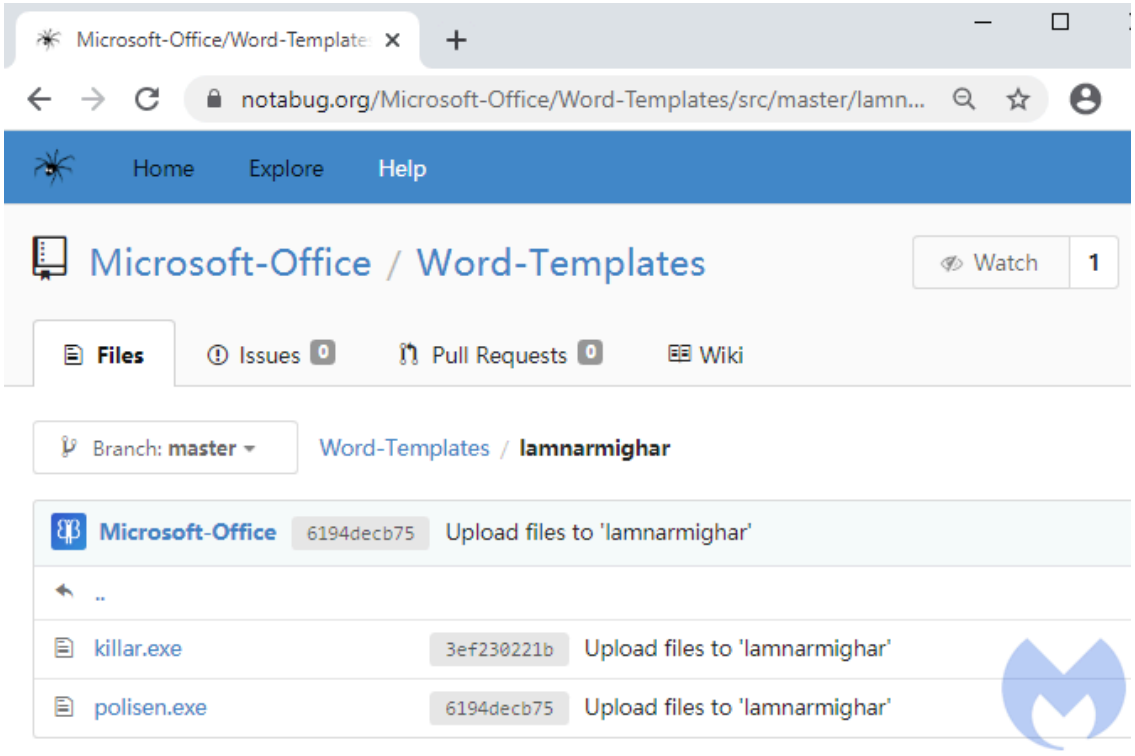
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



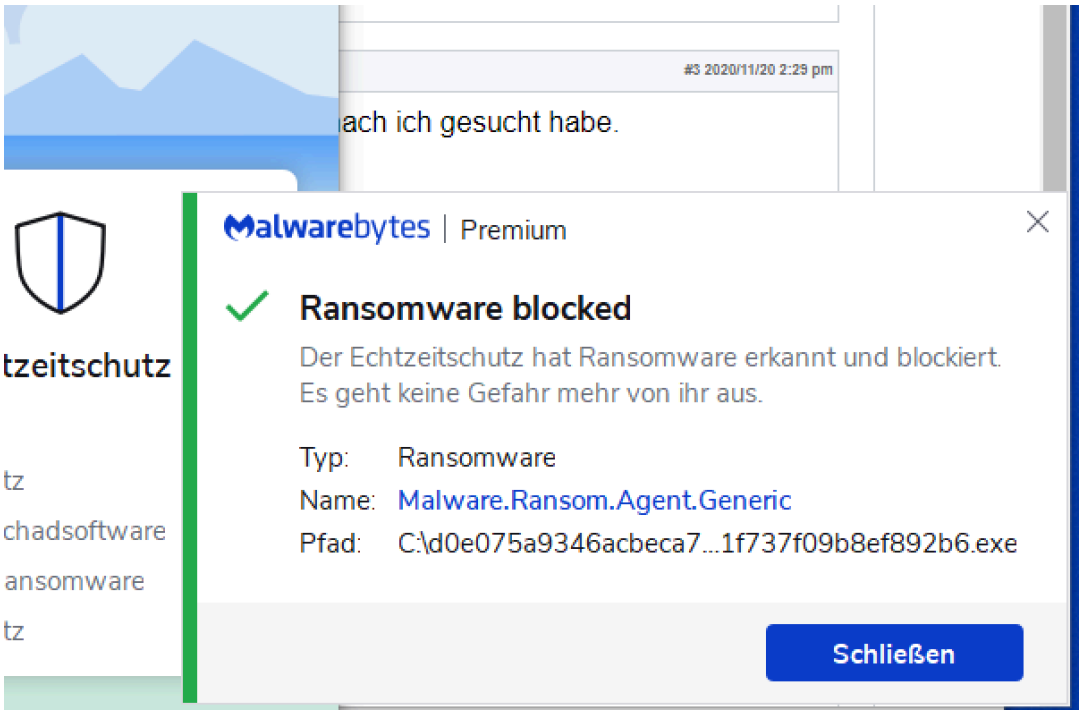
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

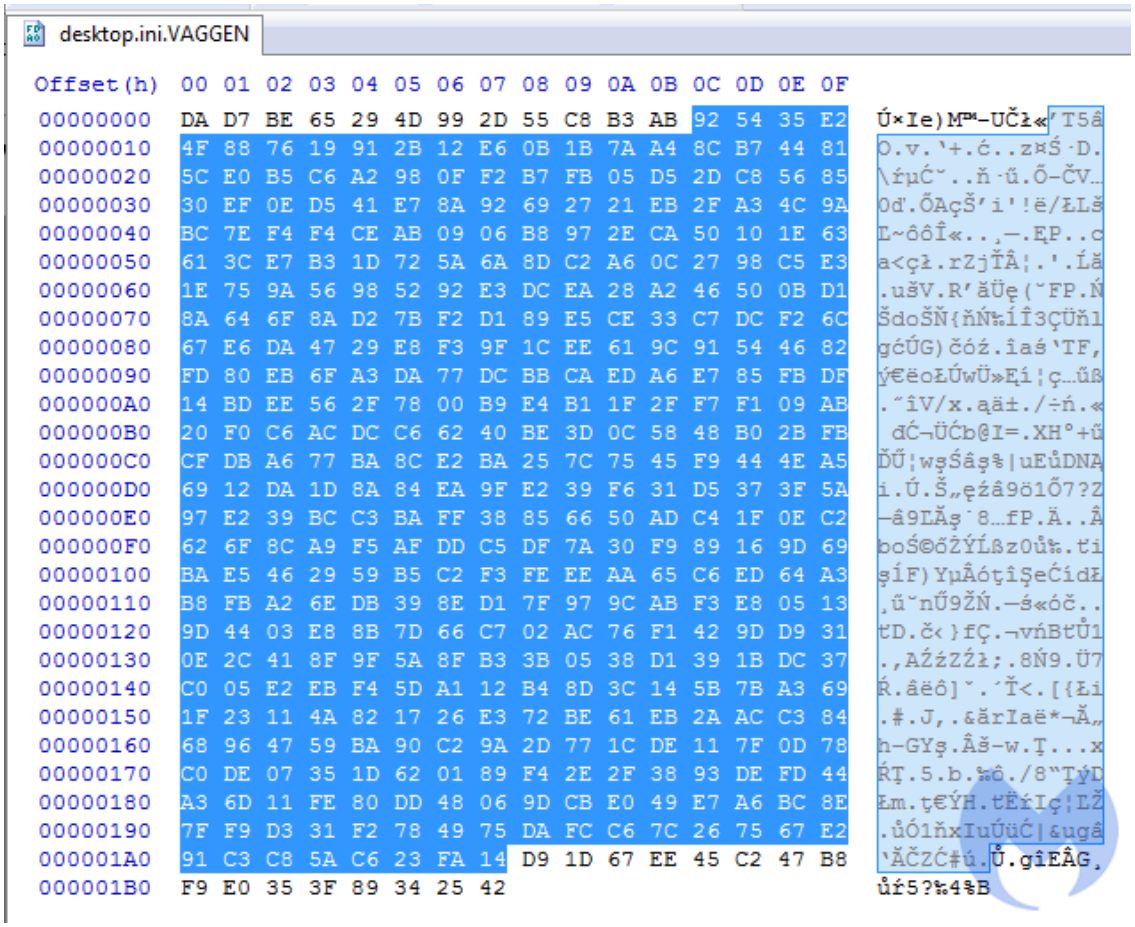
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

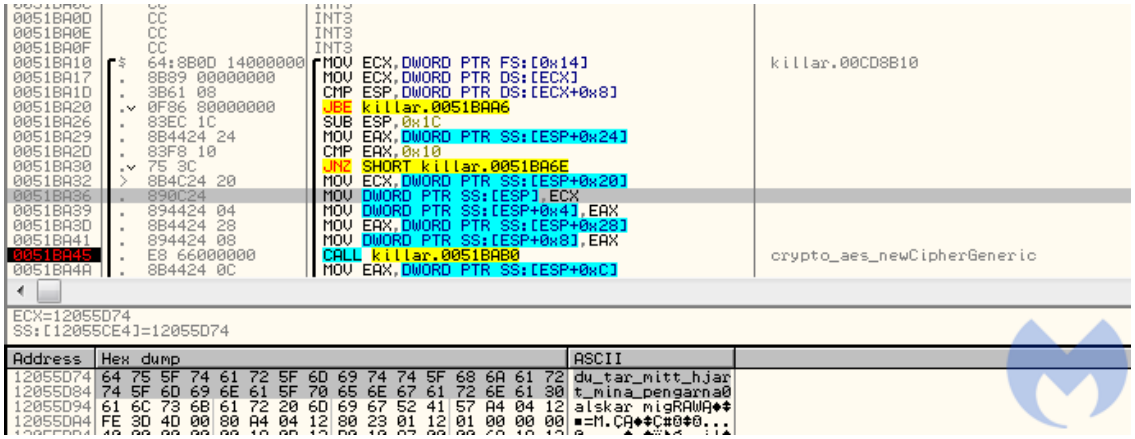


The content of the output file (with .VAGGEN extension) contains:

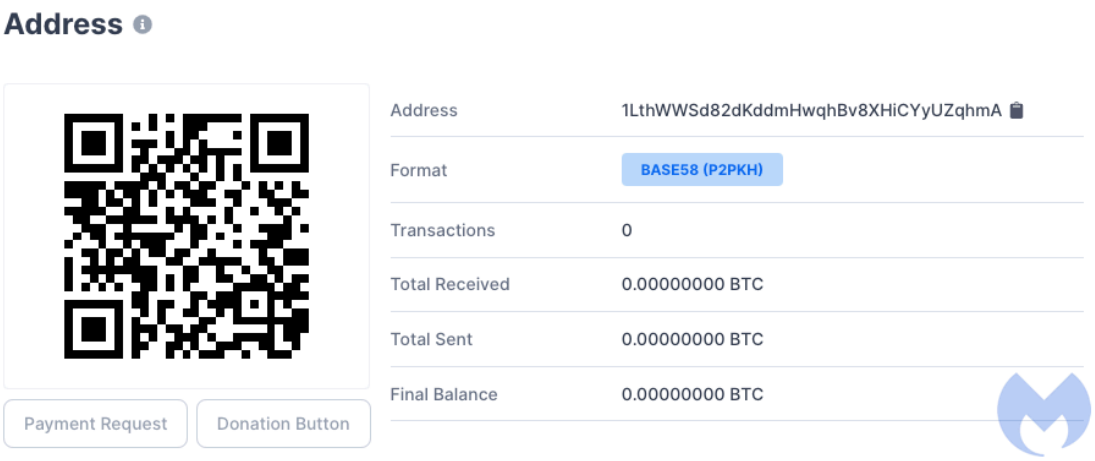
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

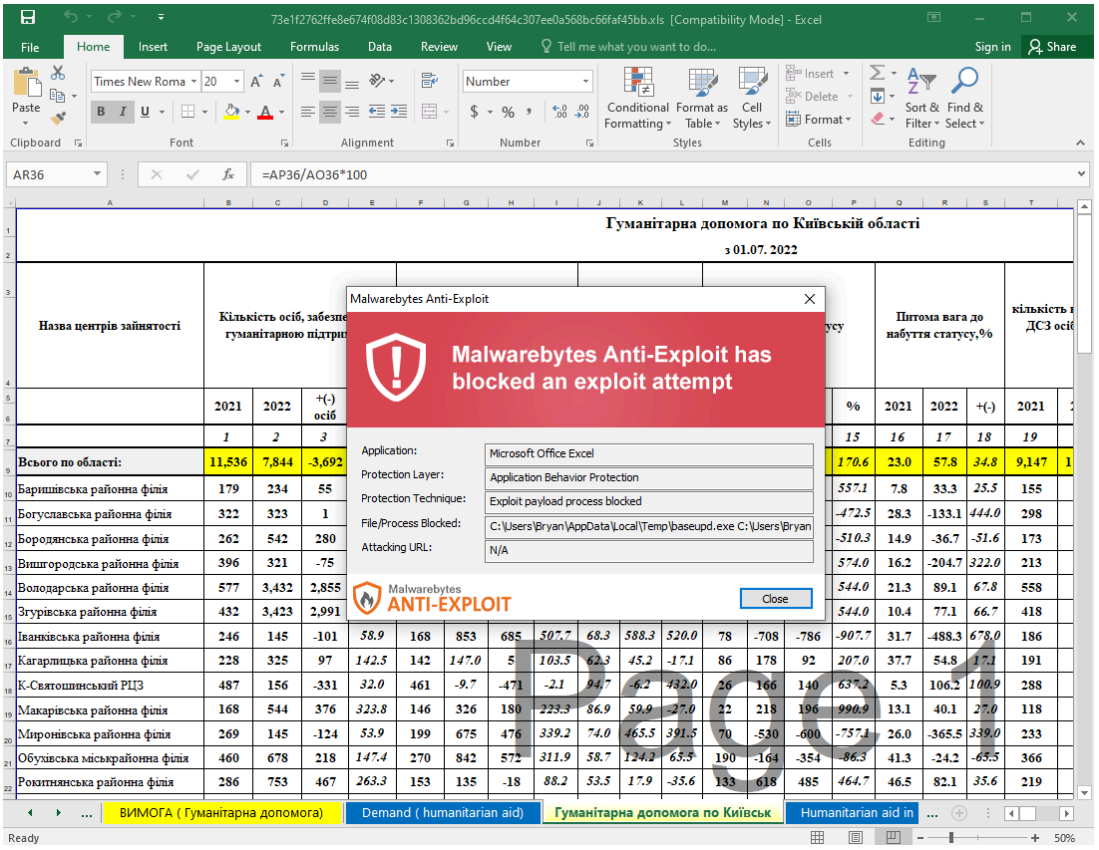
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

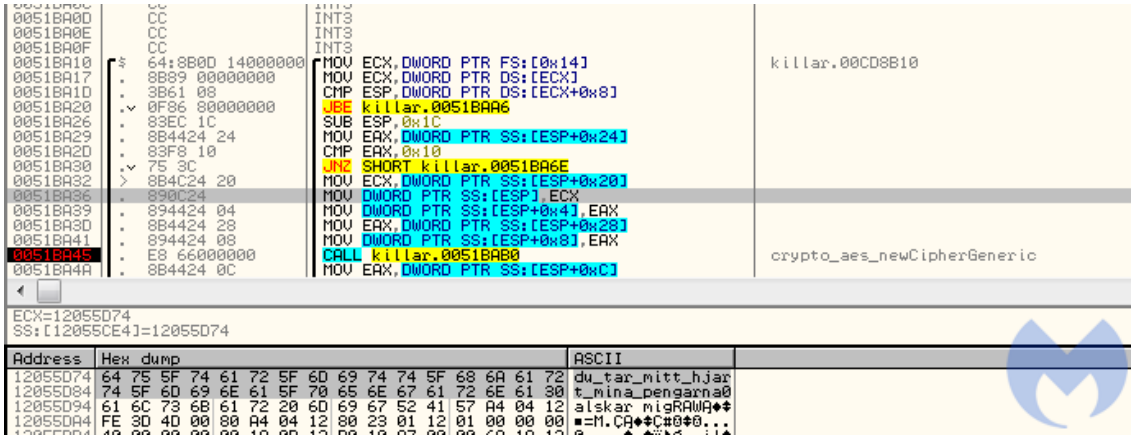
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

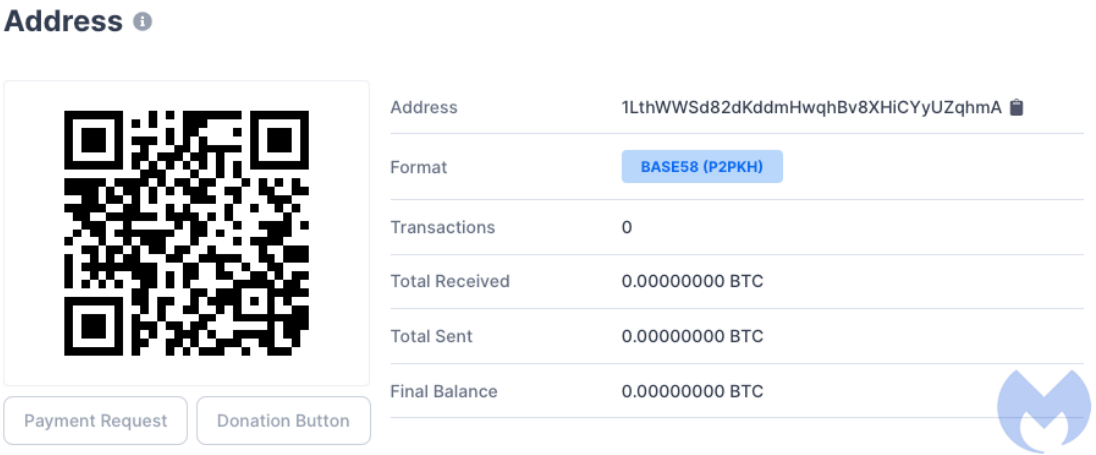
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

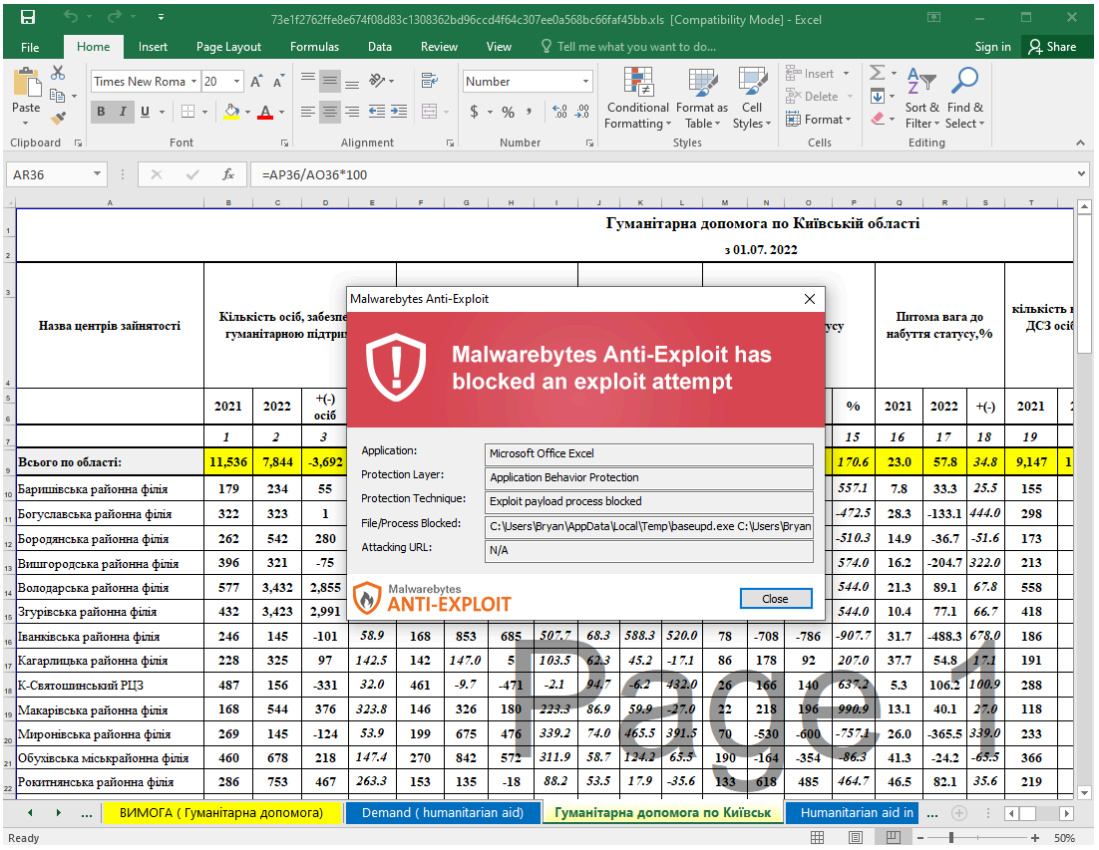
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

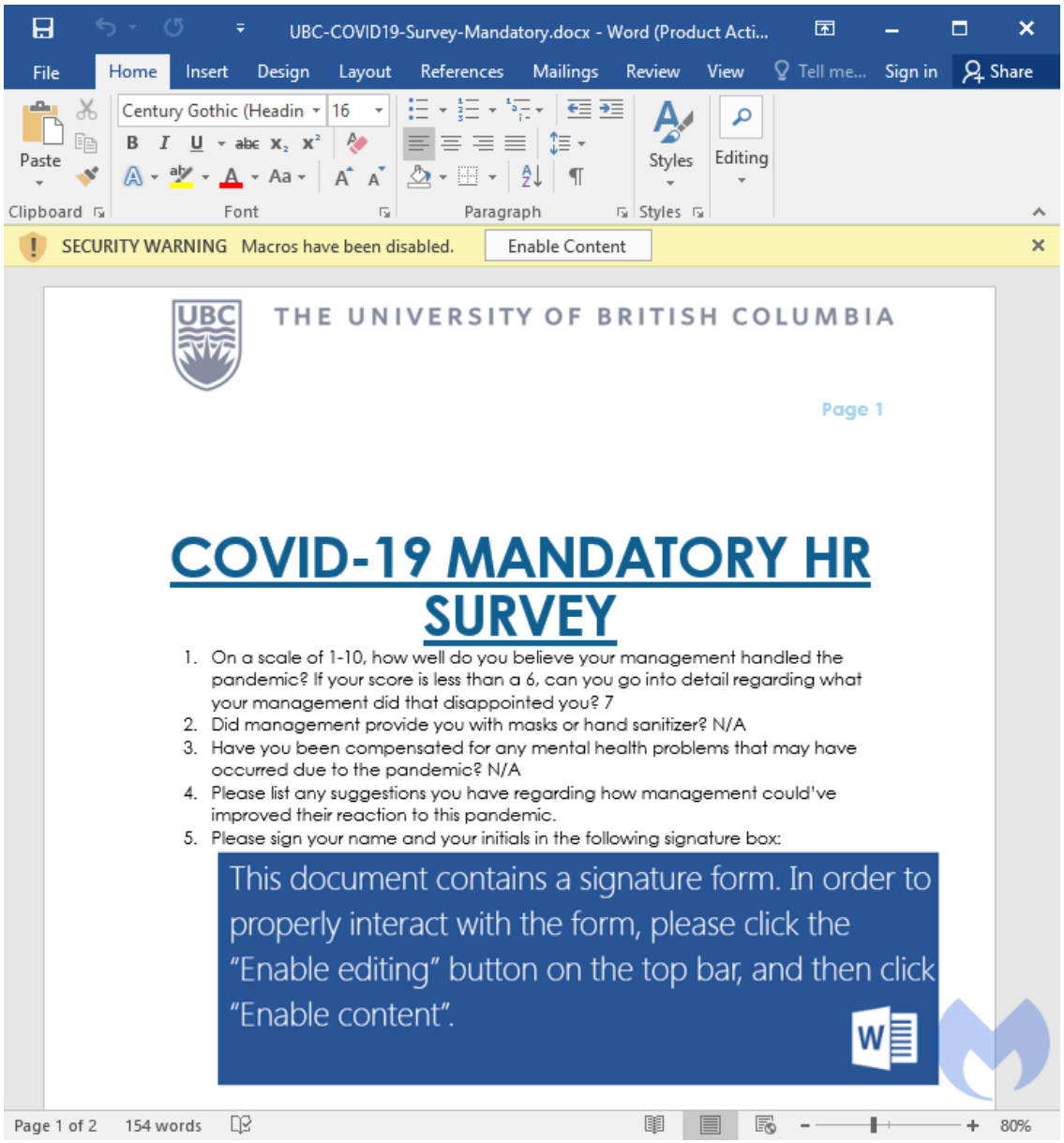
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

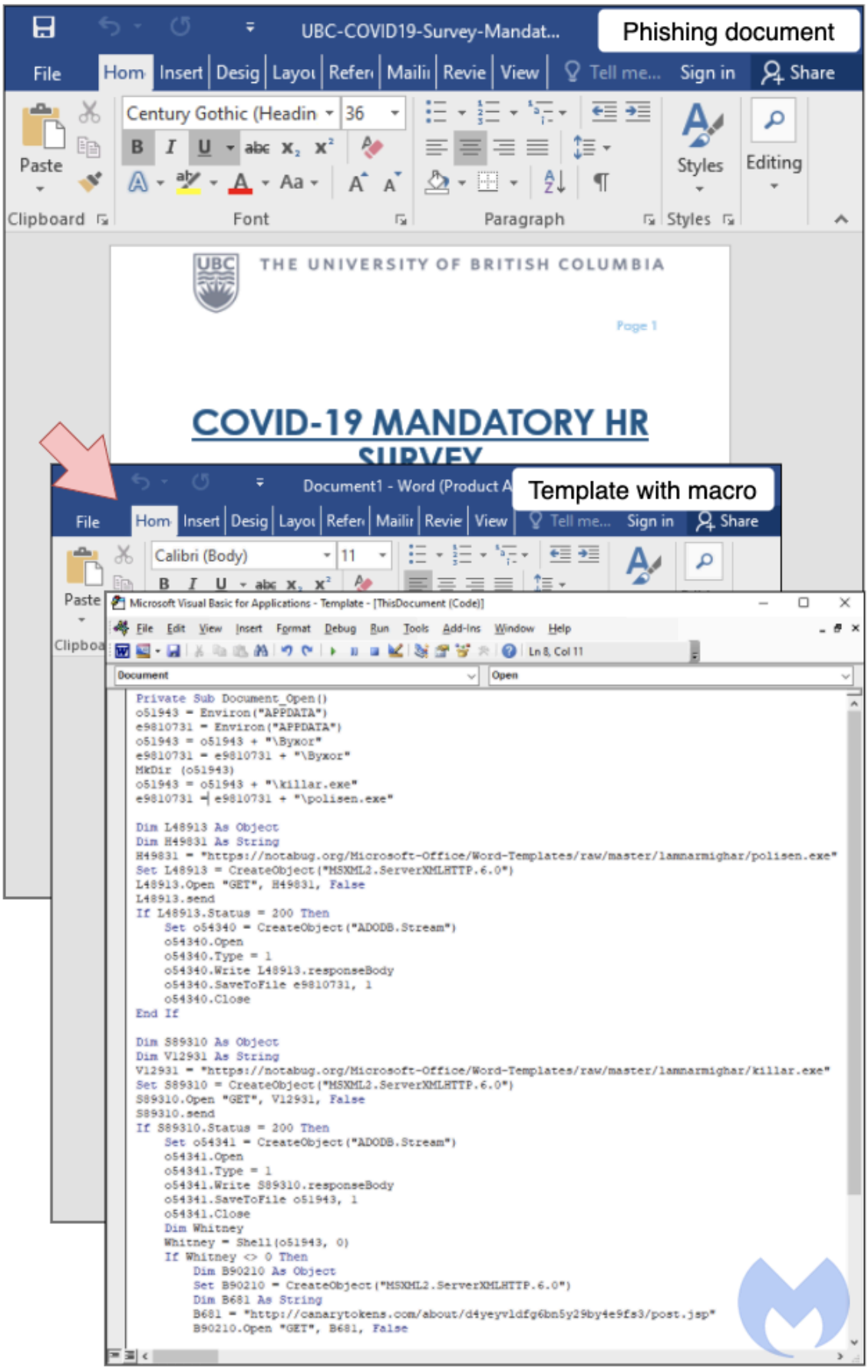
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

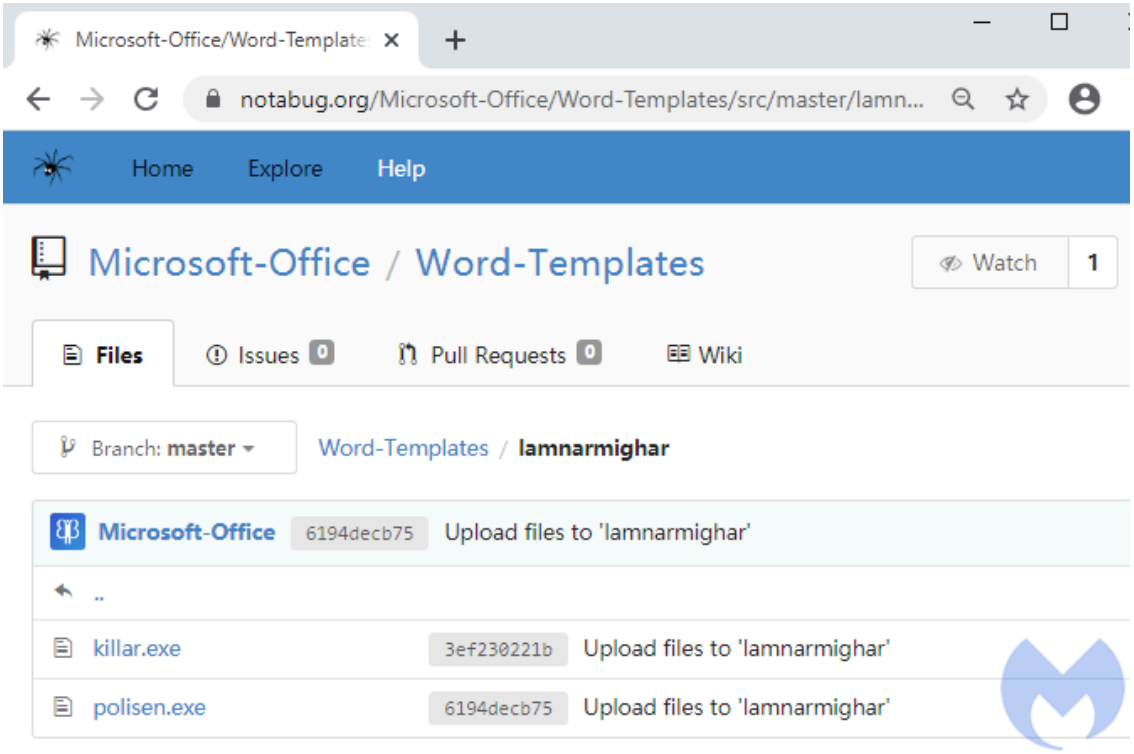
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



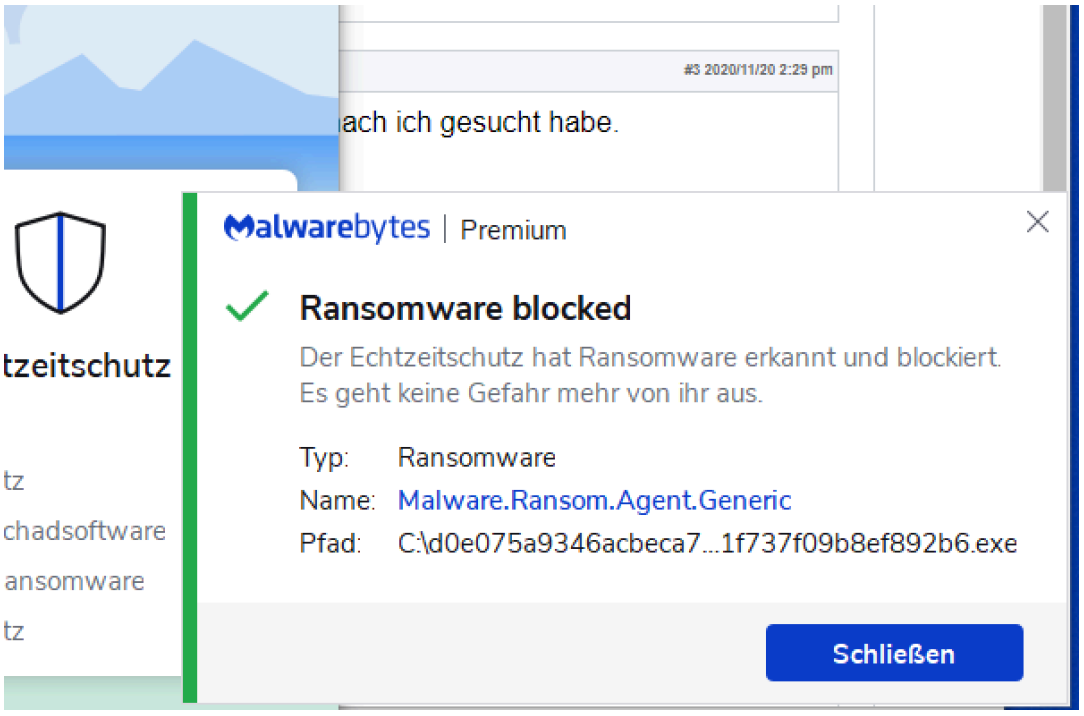
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

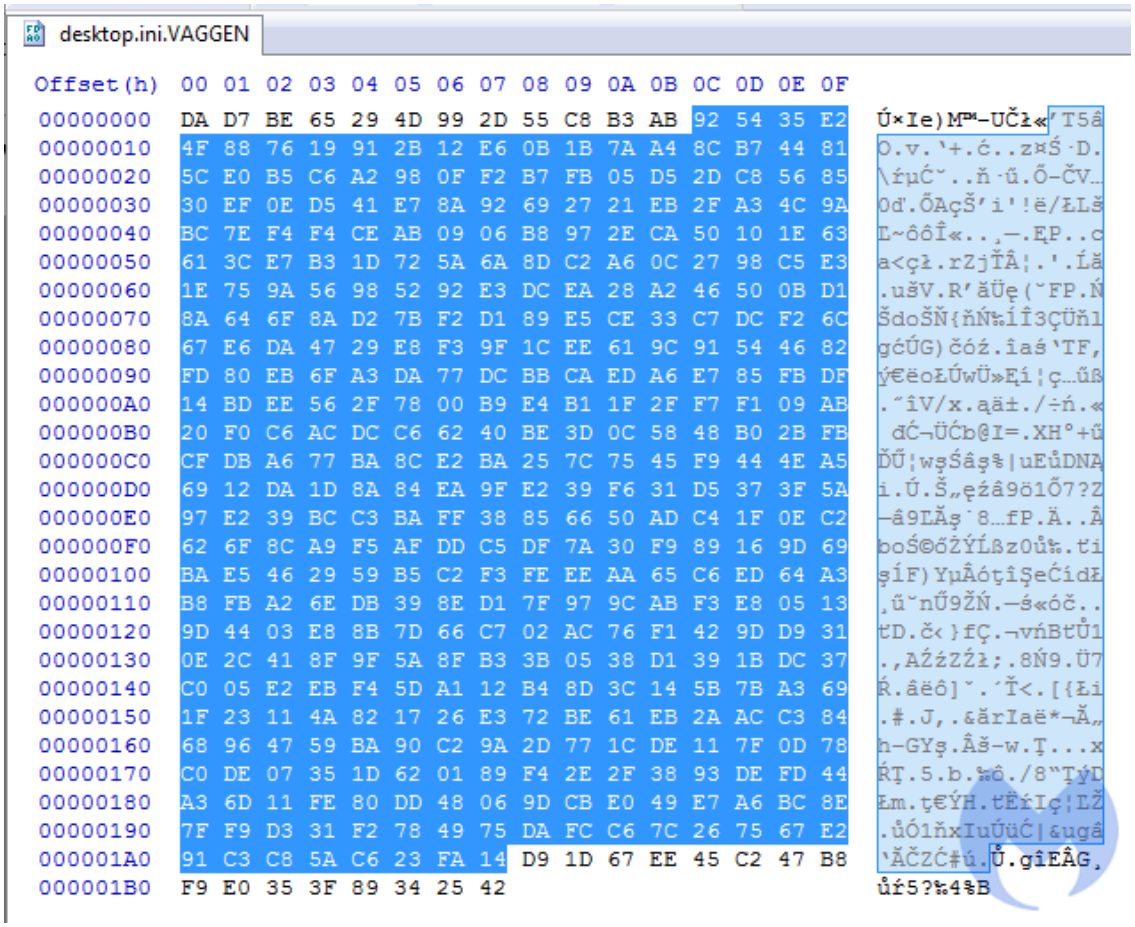
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

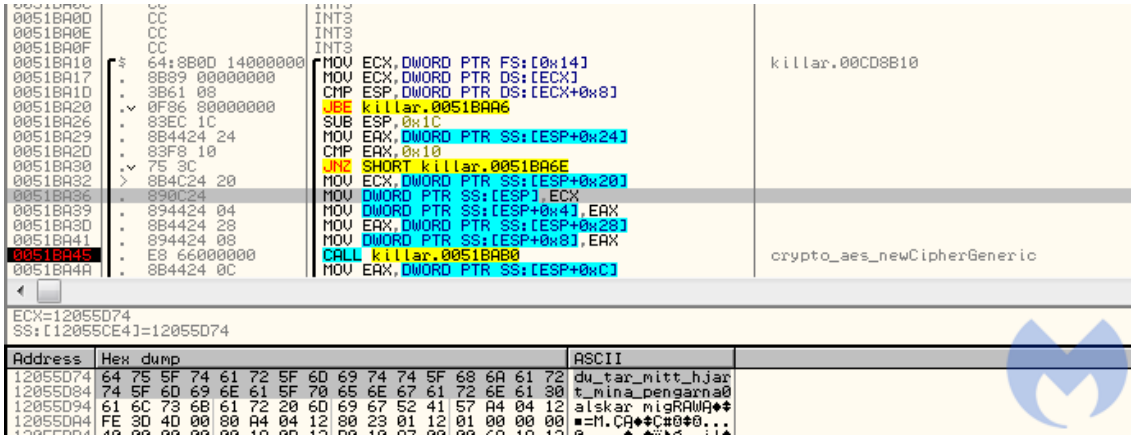


The content of the output file (with .VAGGEN extension) contains:

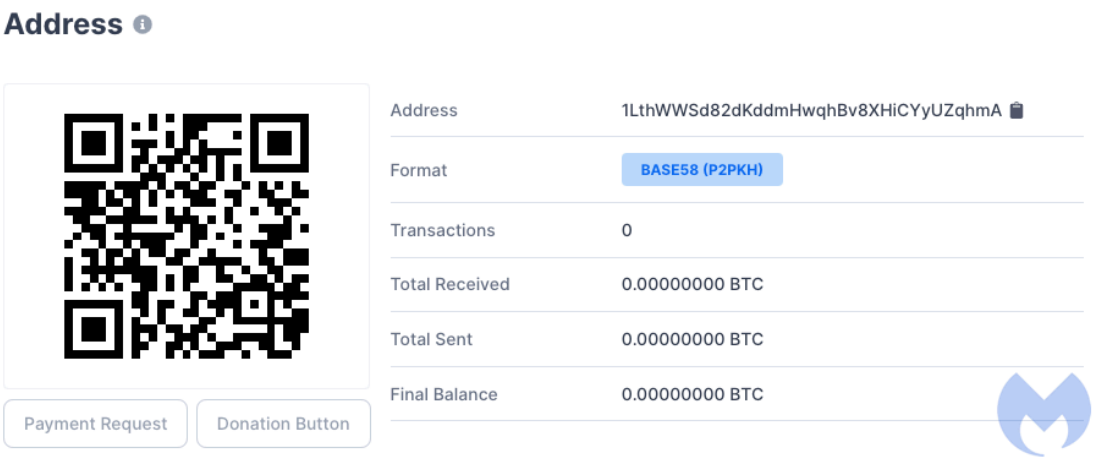
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

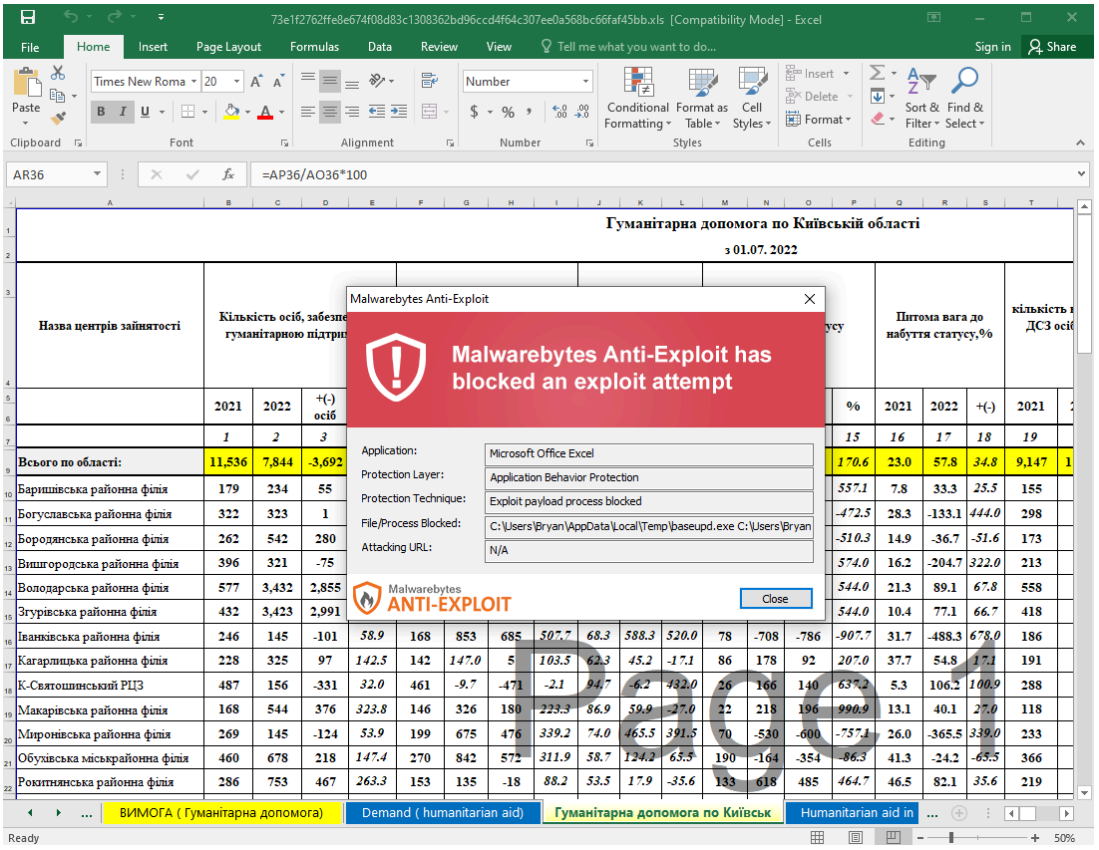
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

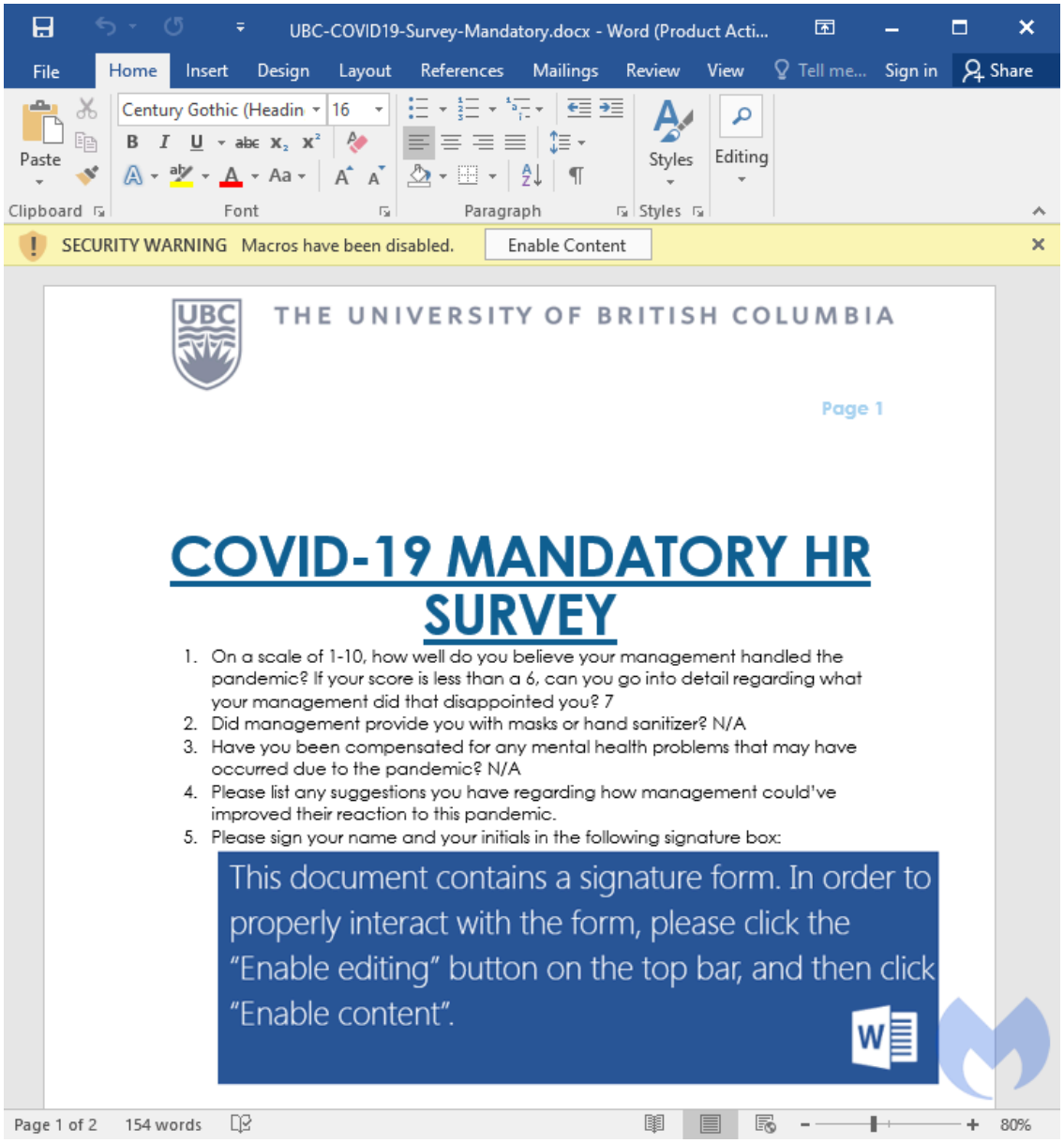
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

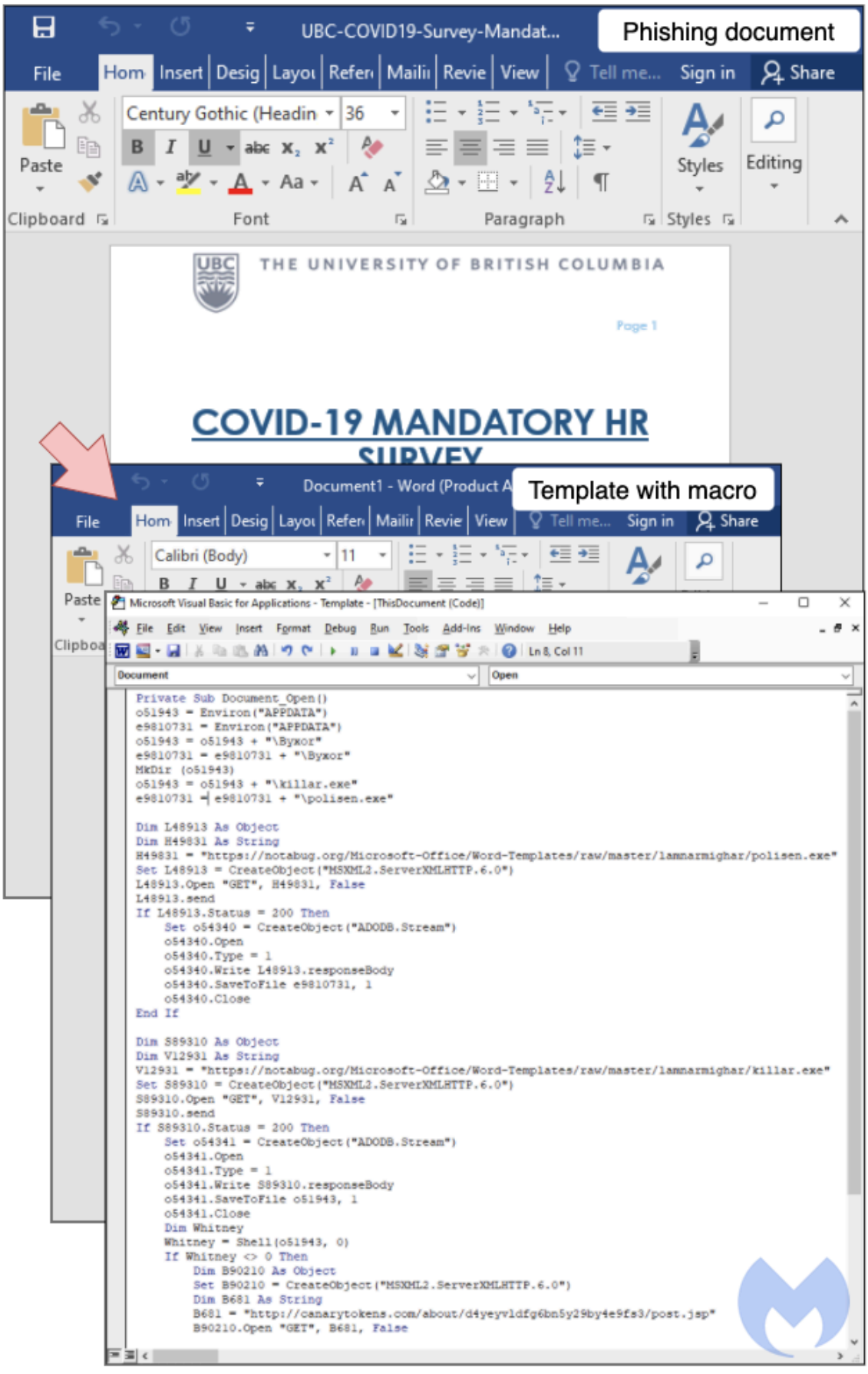
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

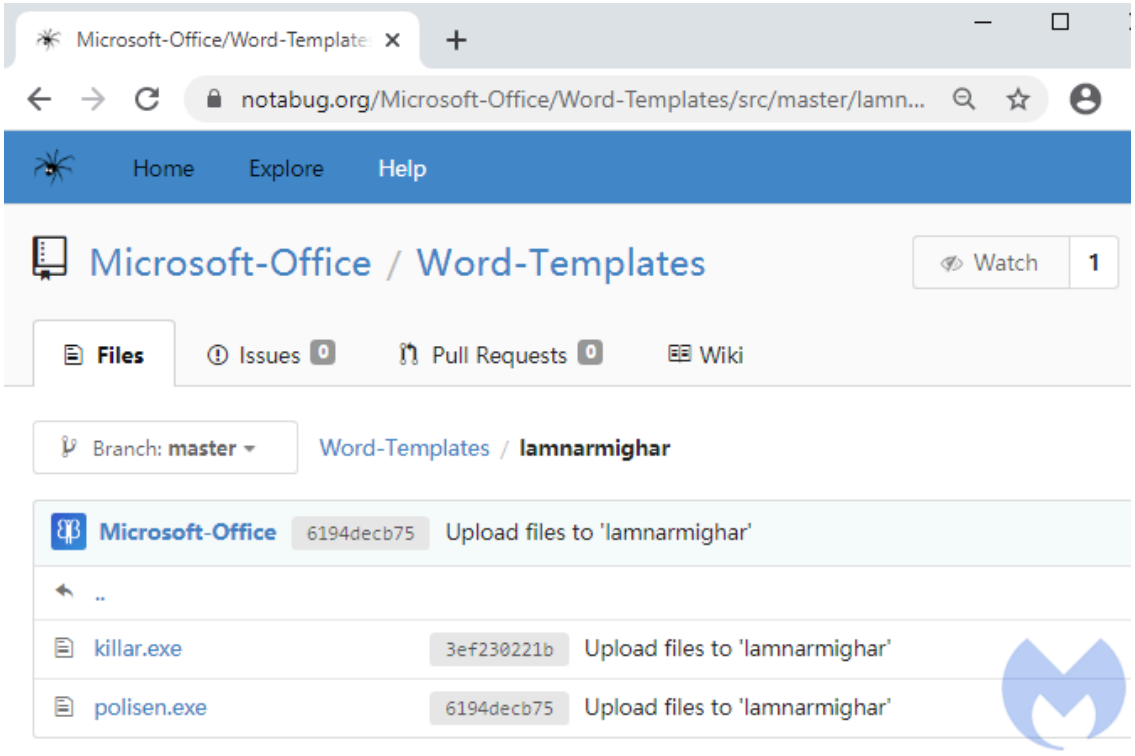
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



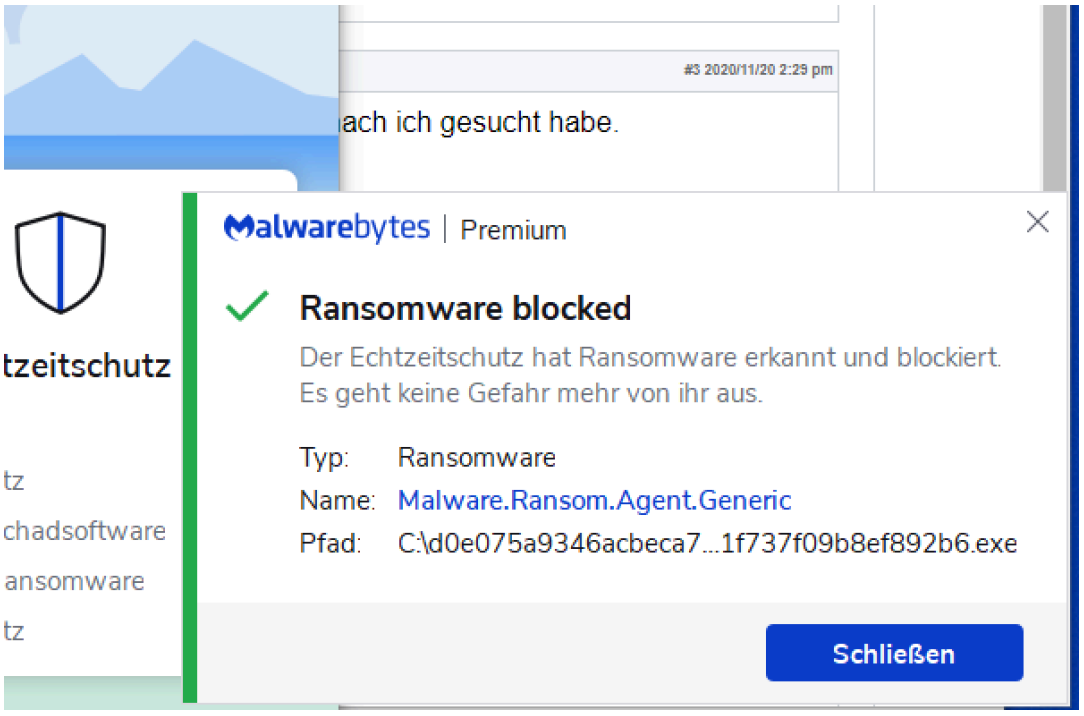
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

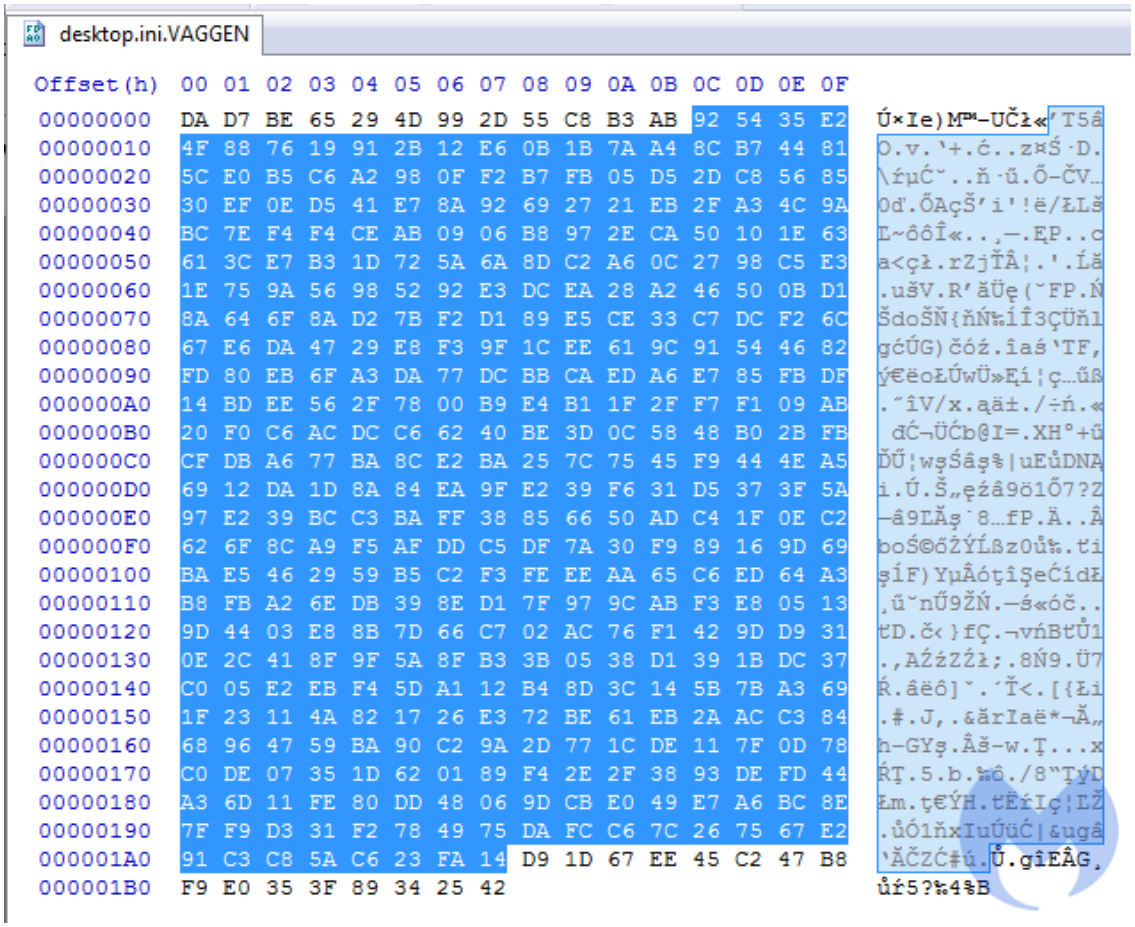
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

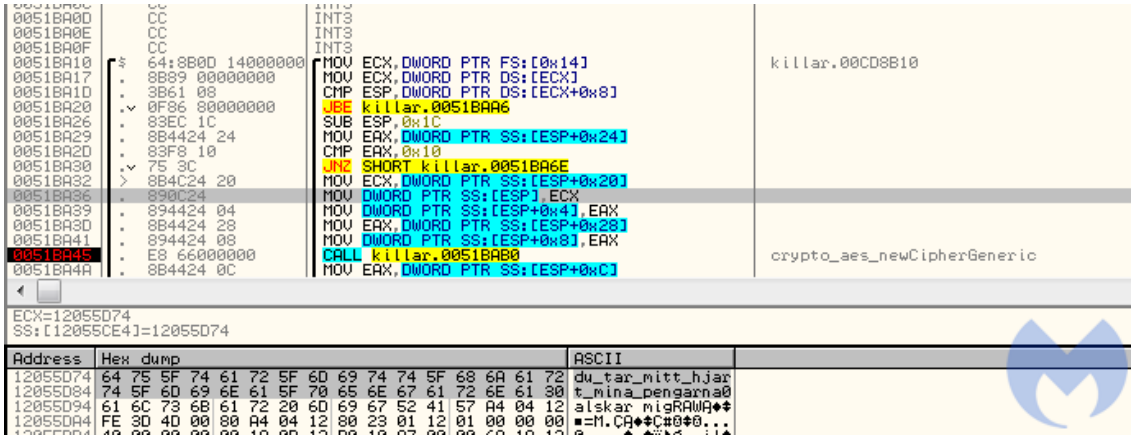


The content of the output file (with .VAGGEN extension) contains:

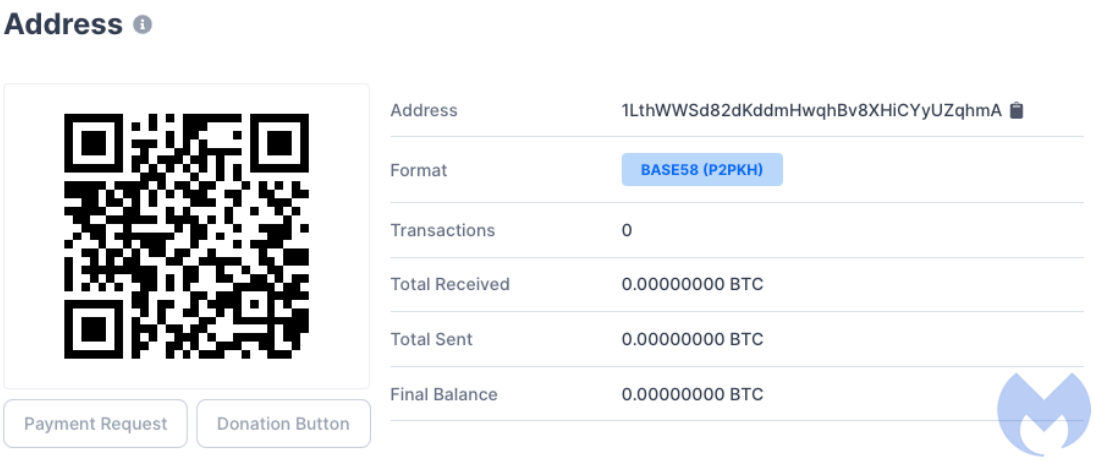
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

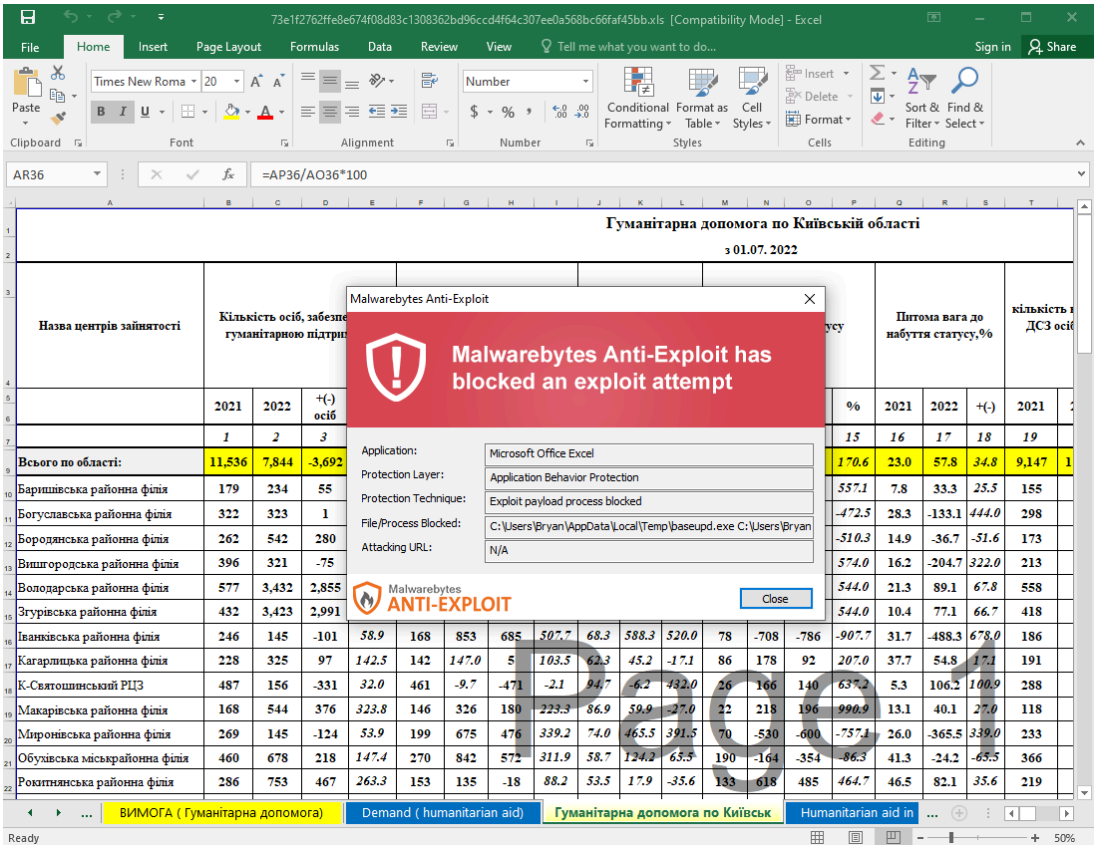
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

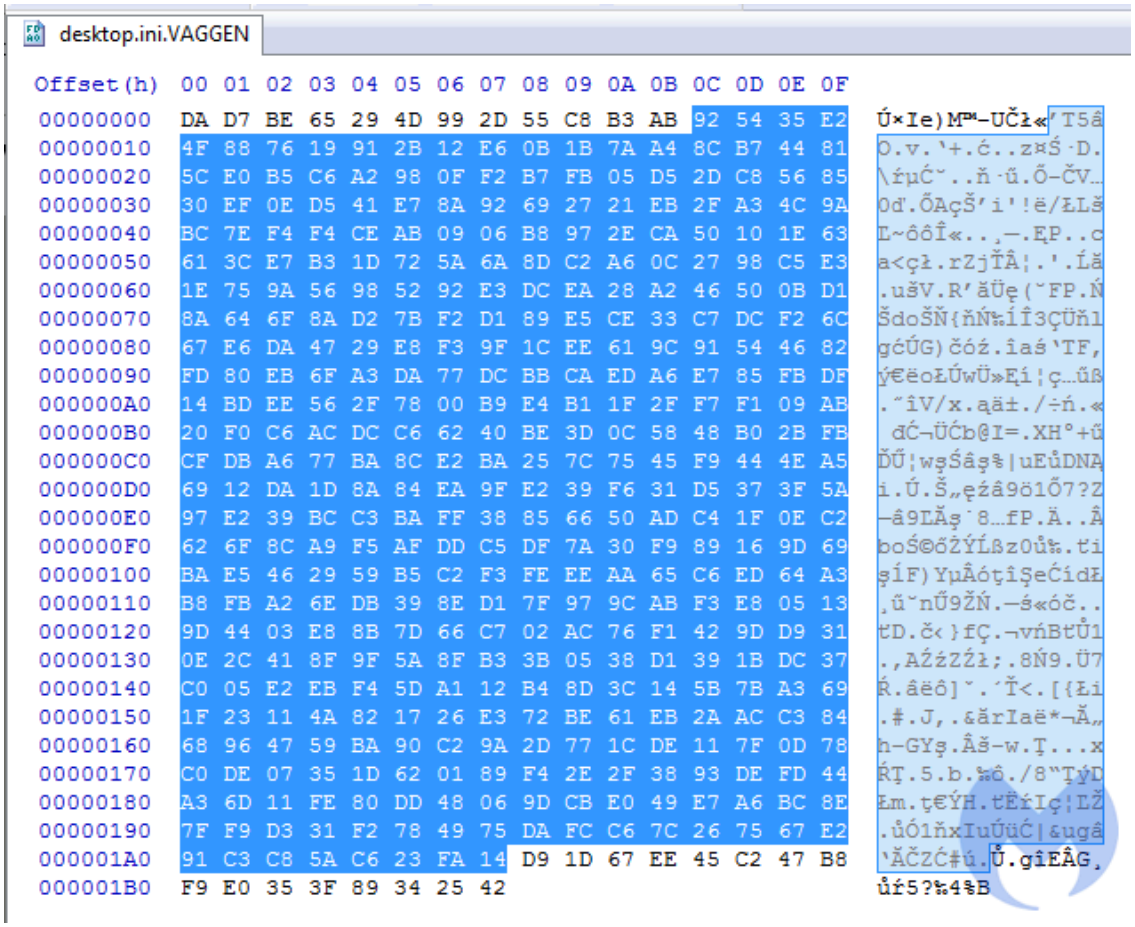
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

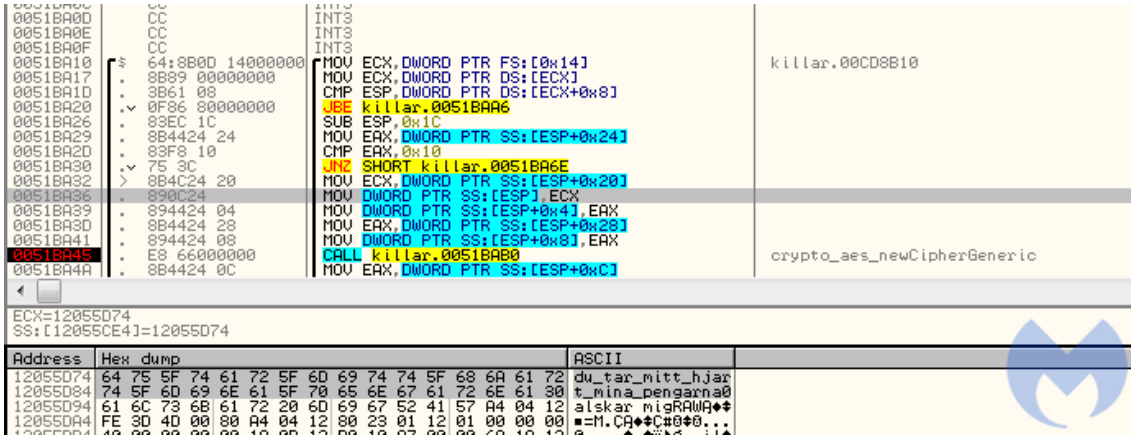
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

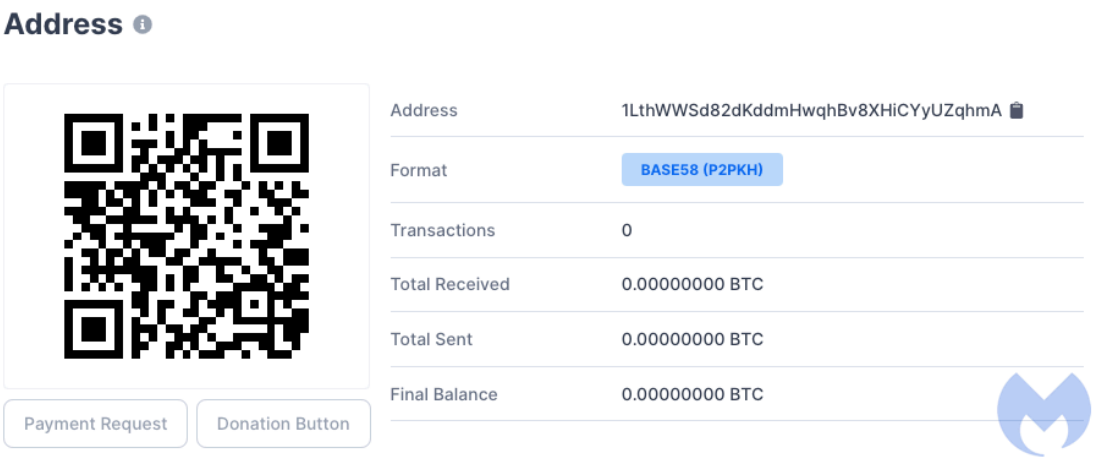
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

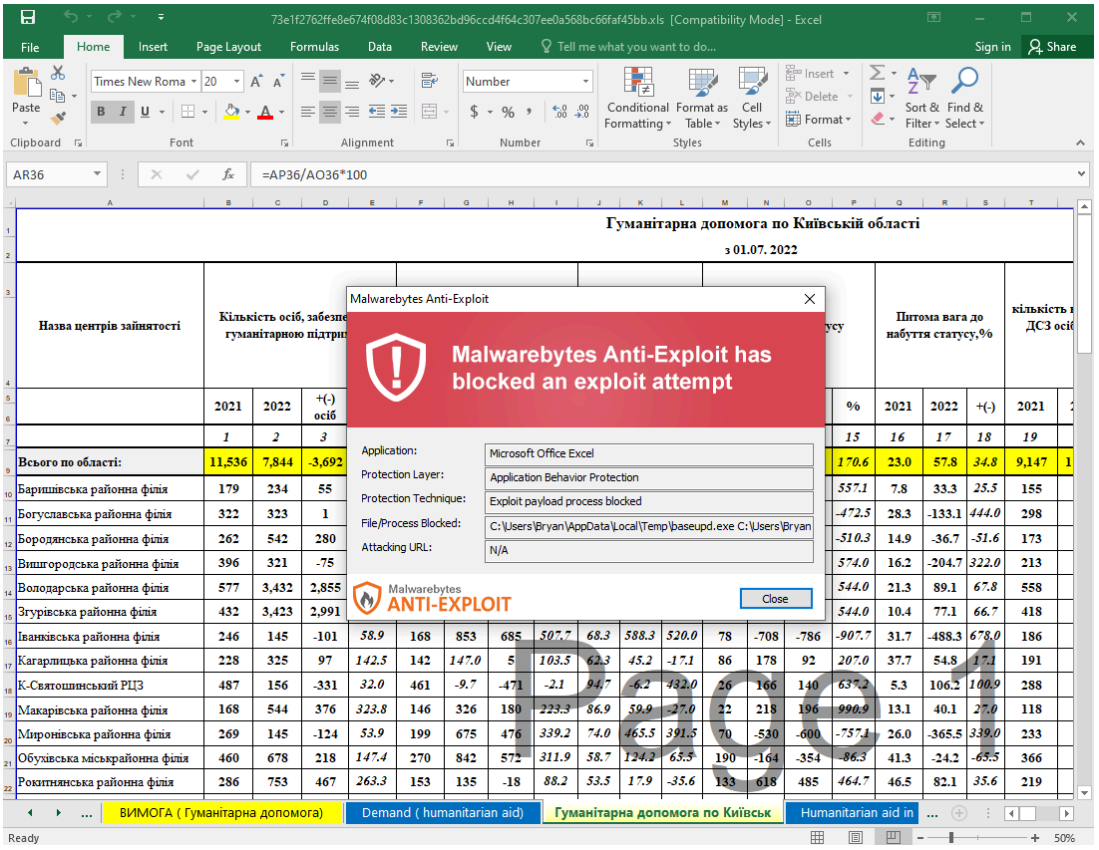
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

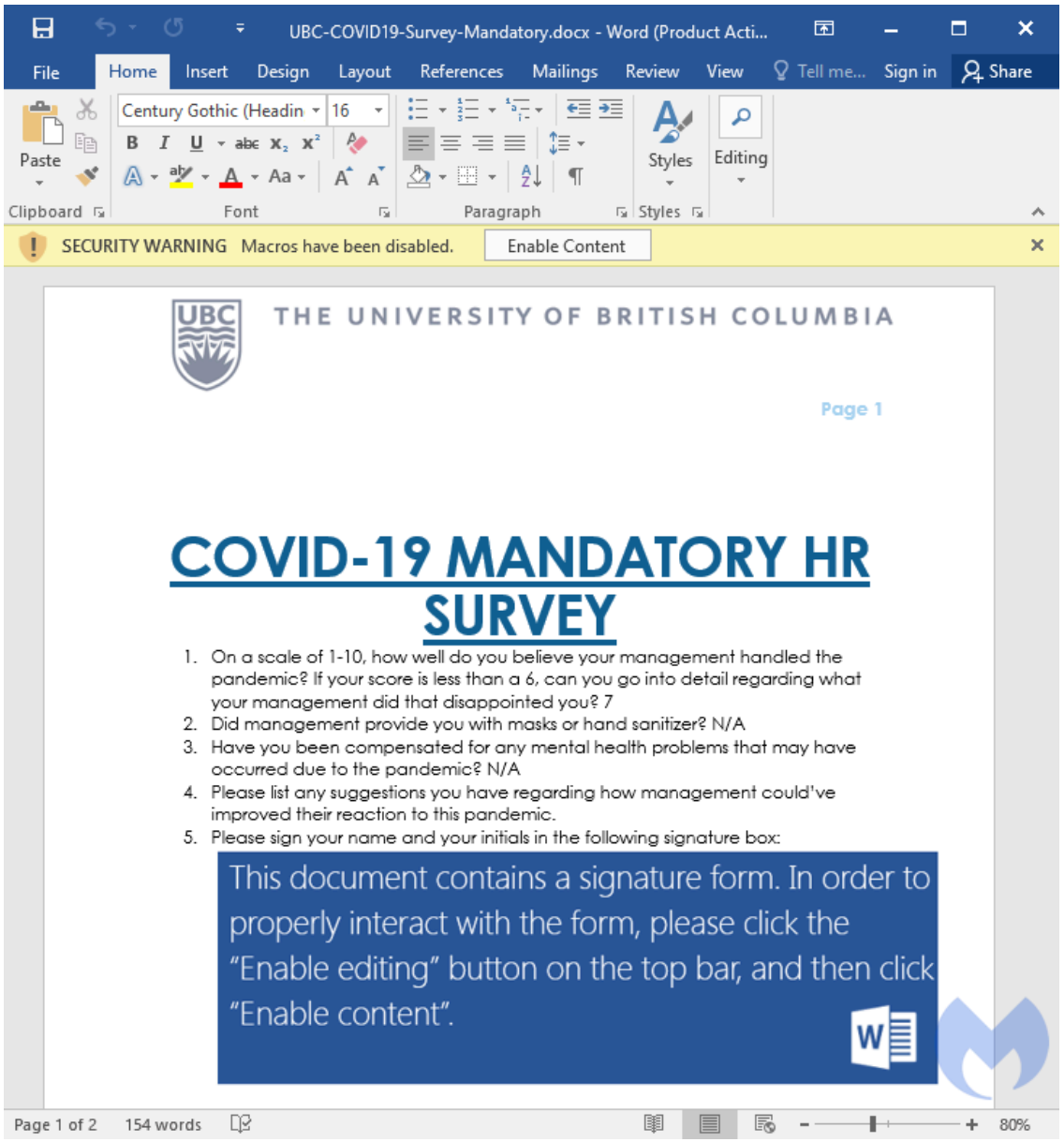
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

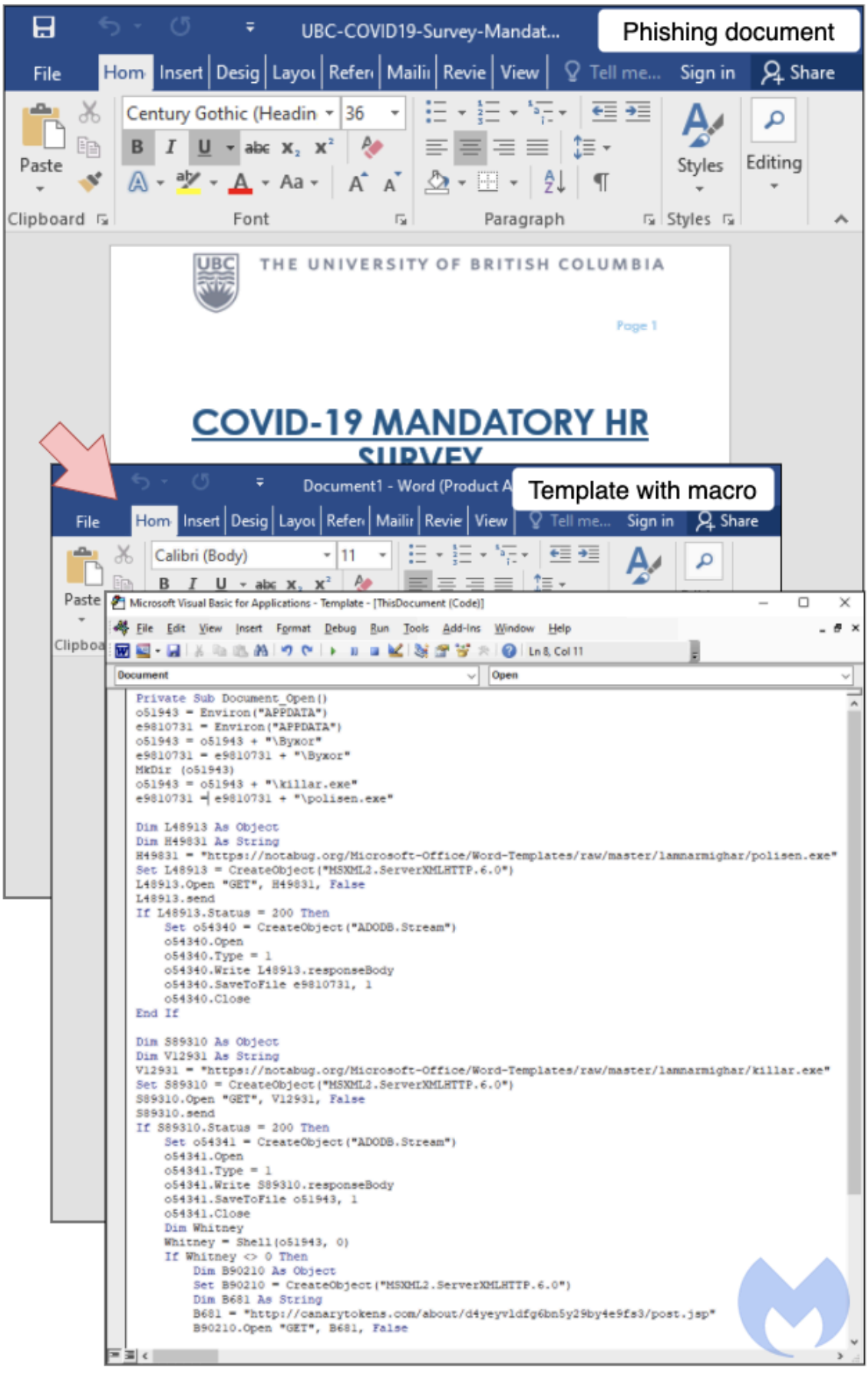
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

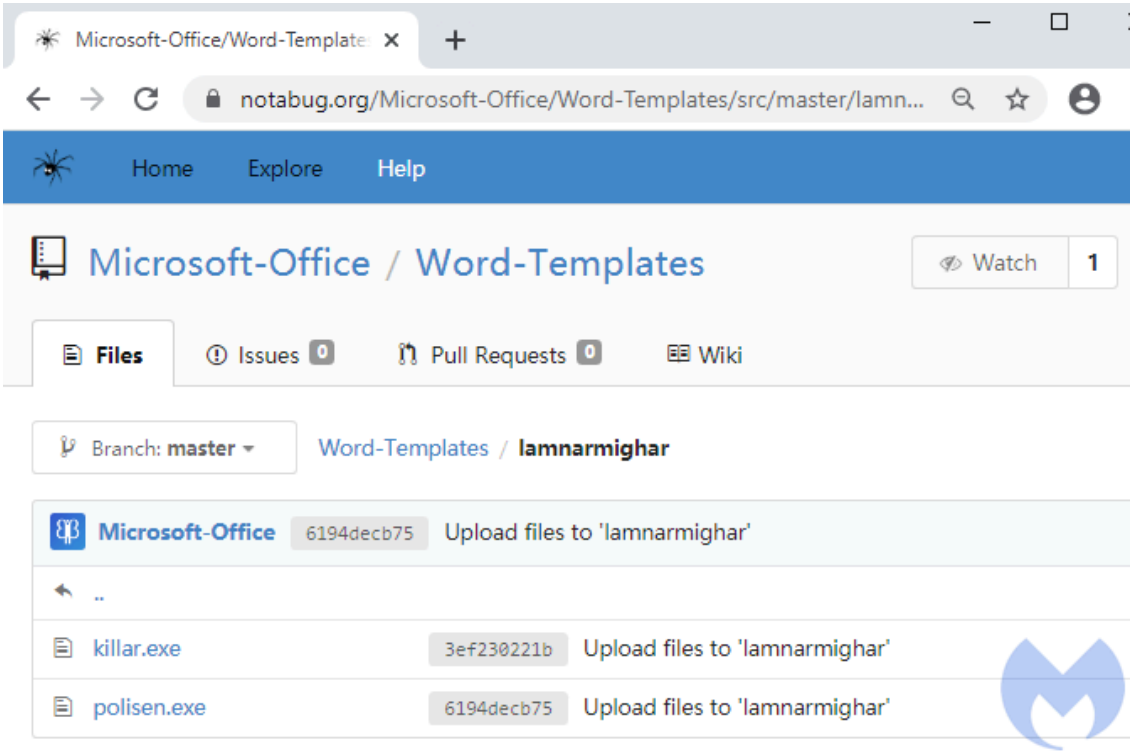
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



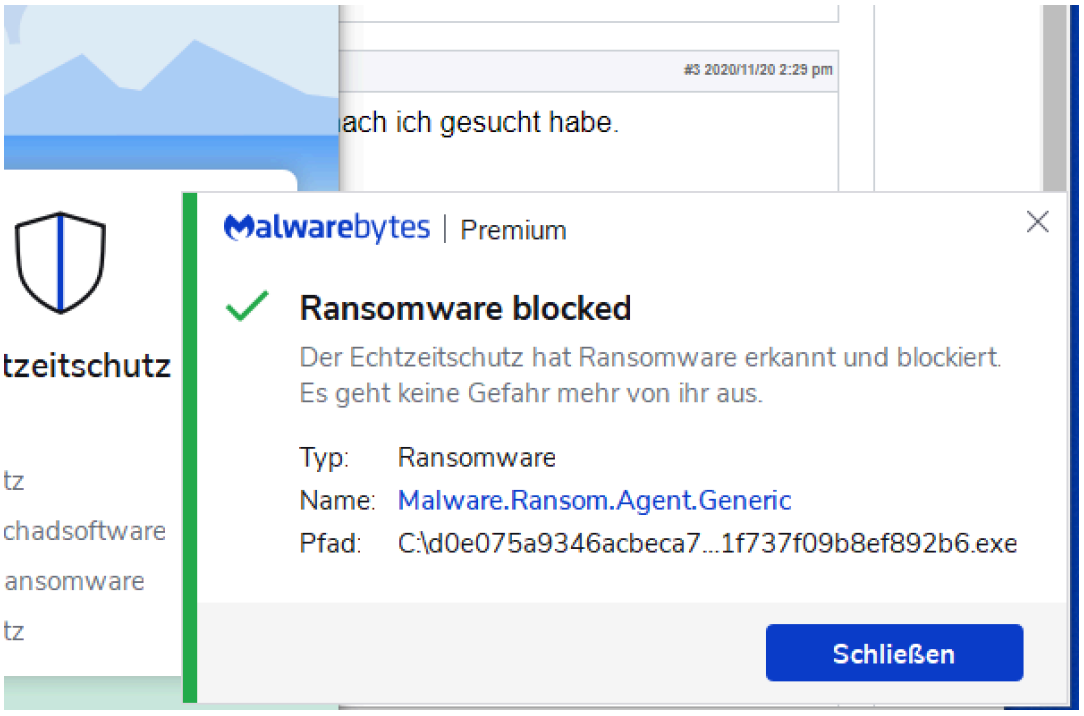
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

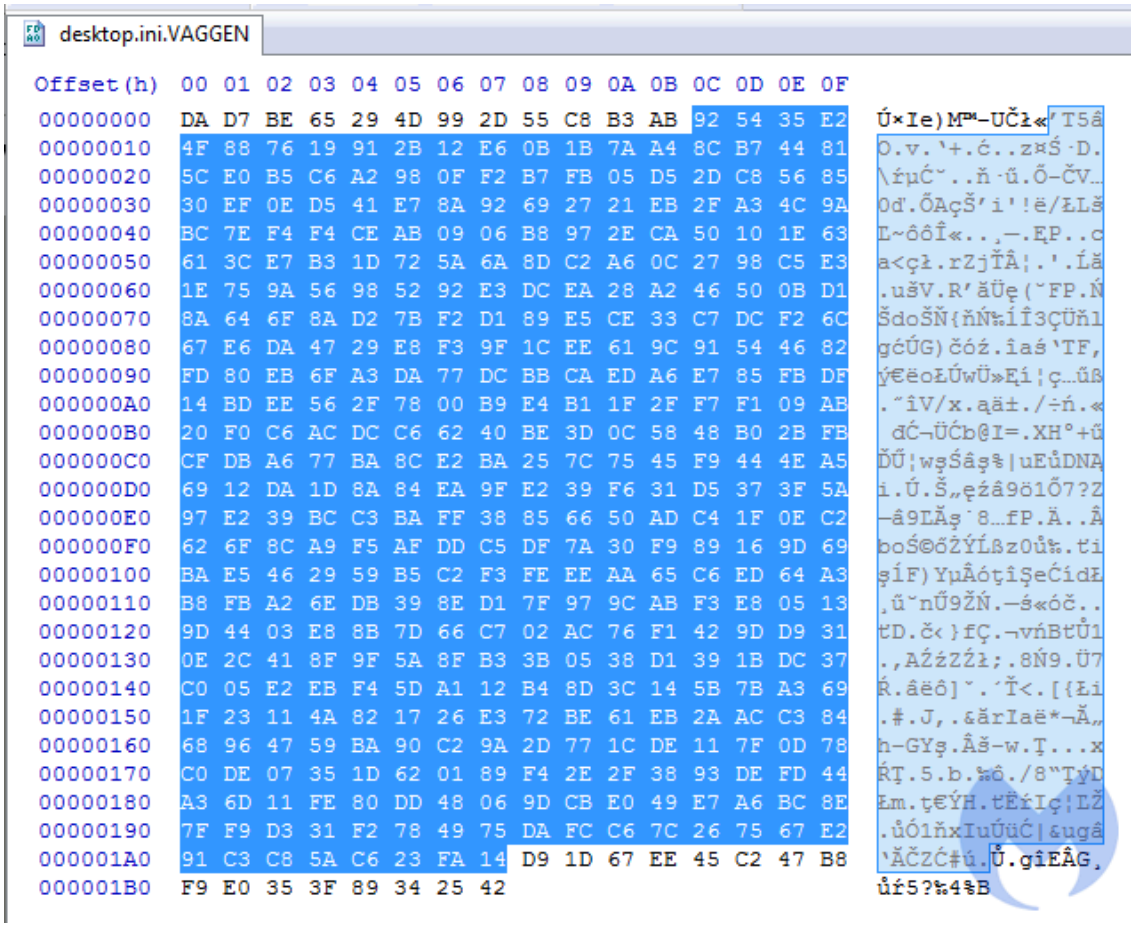
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

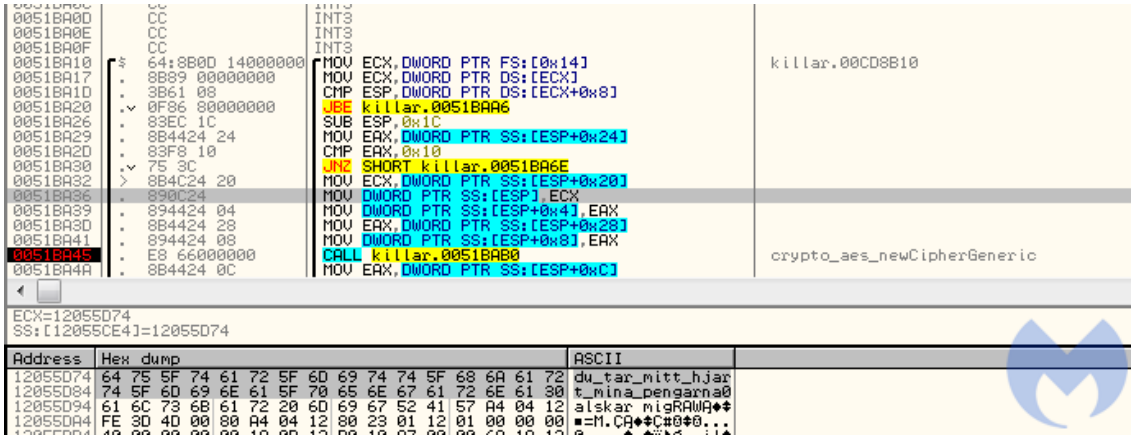


The content of the output file (with .VAGGEN extension) contains:

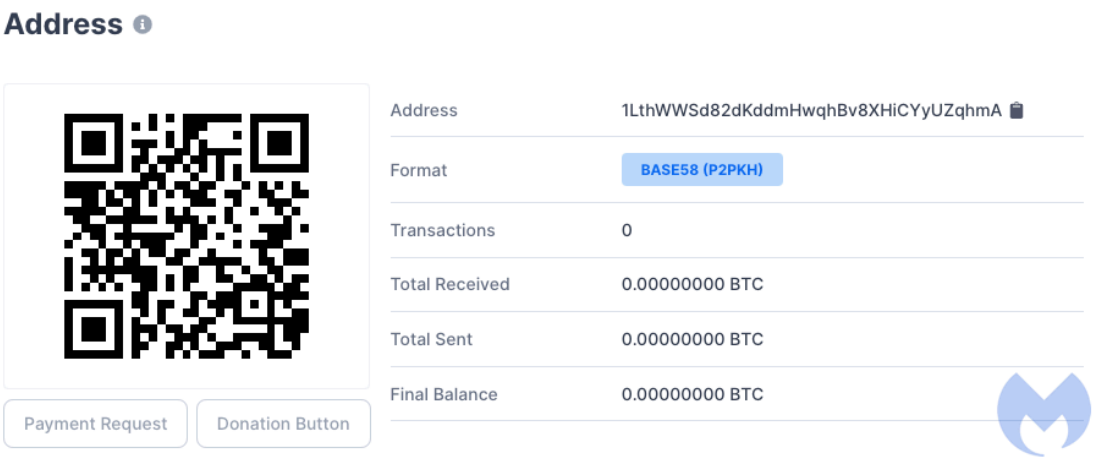
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

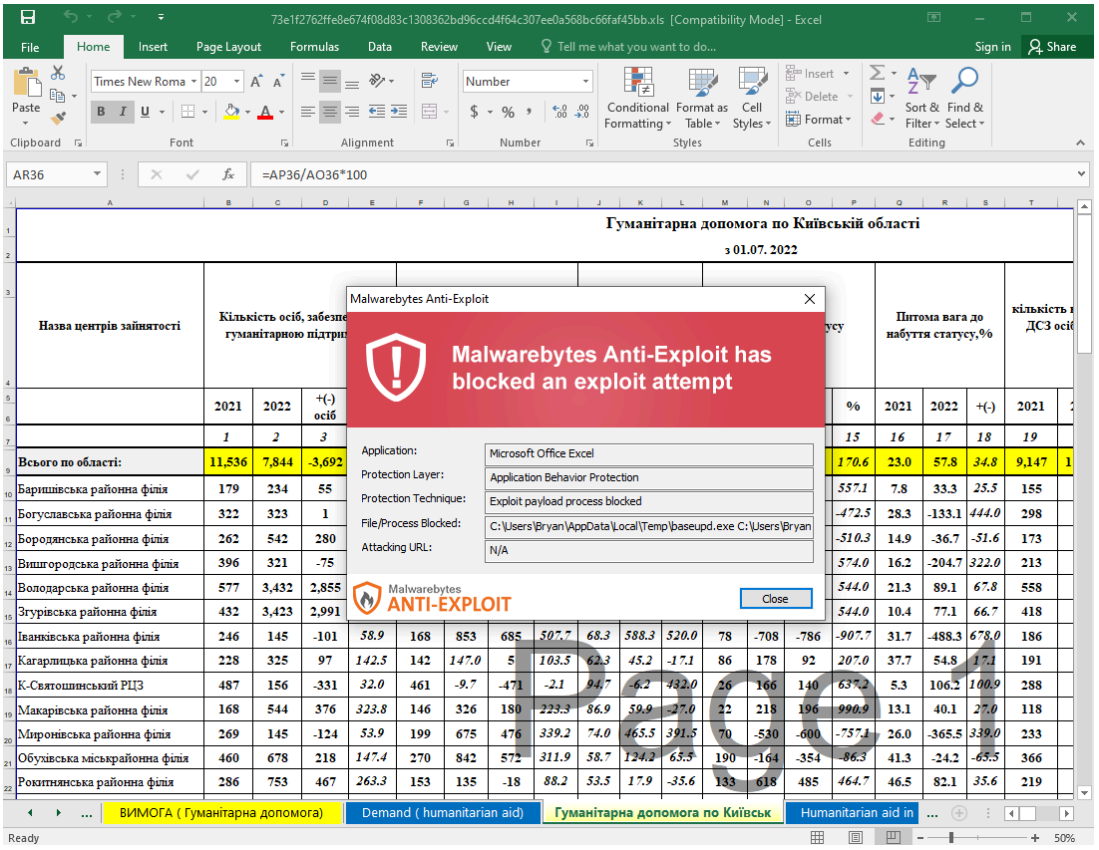
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

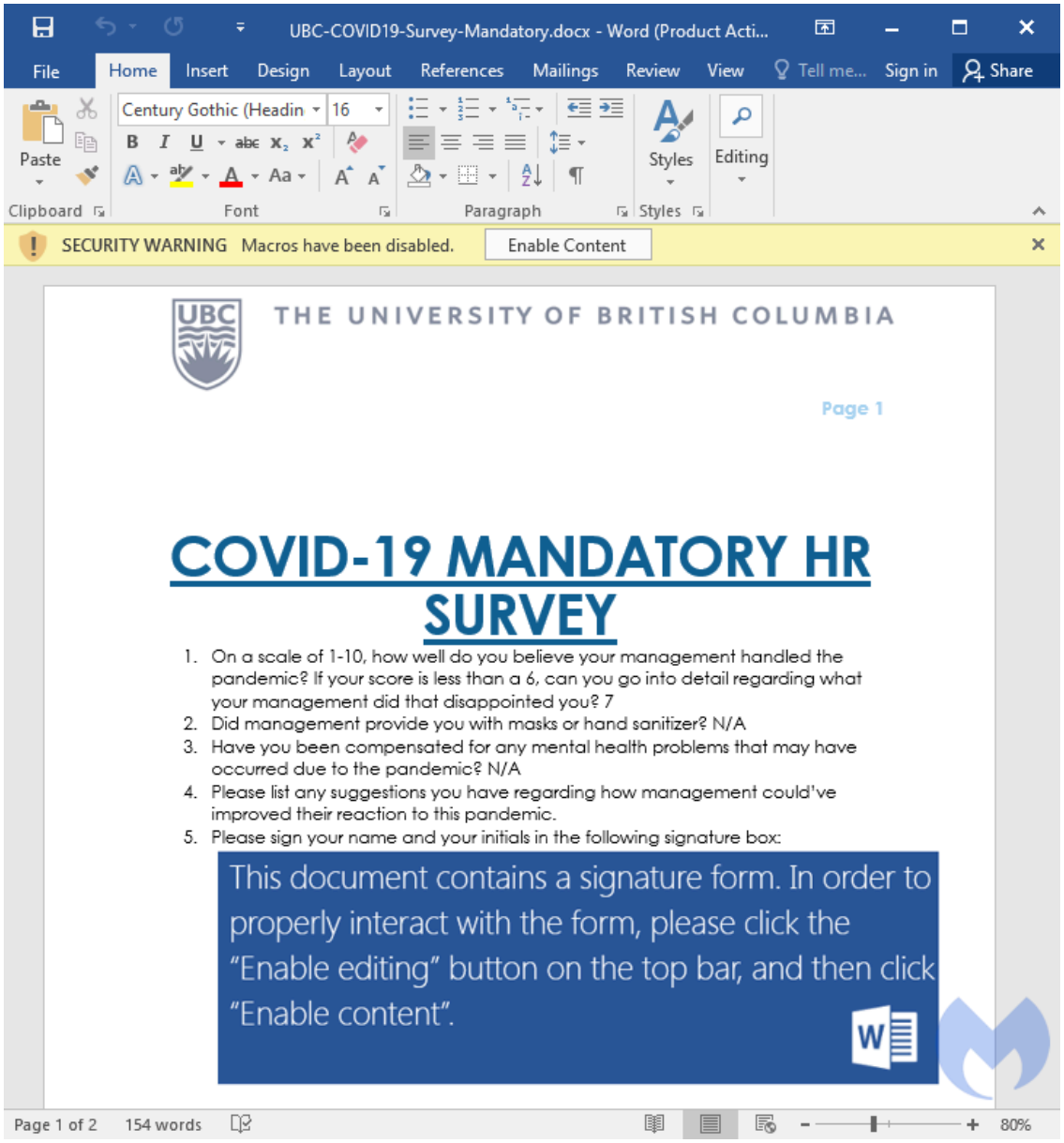
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

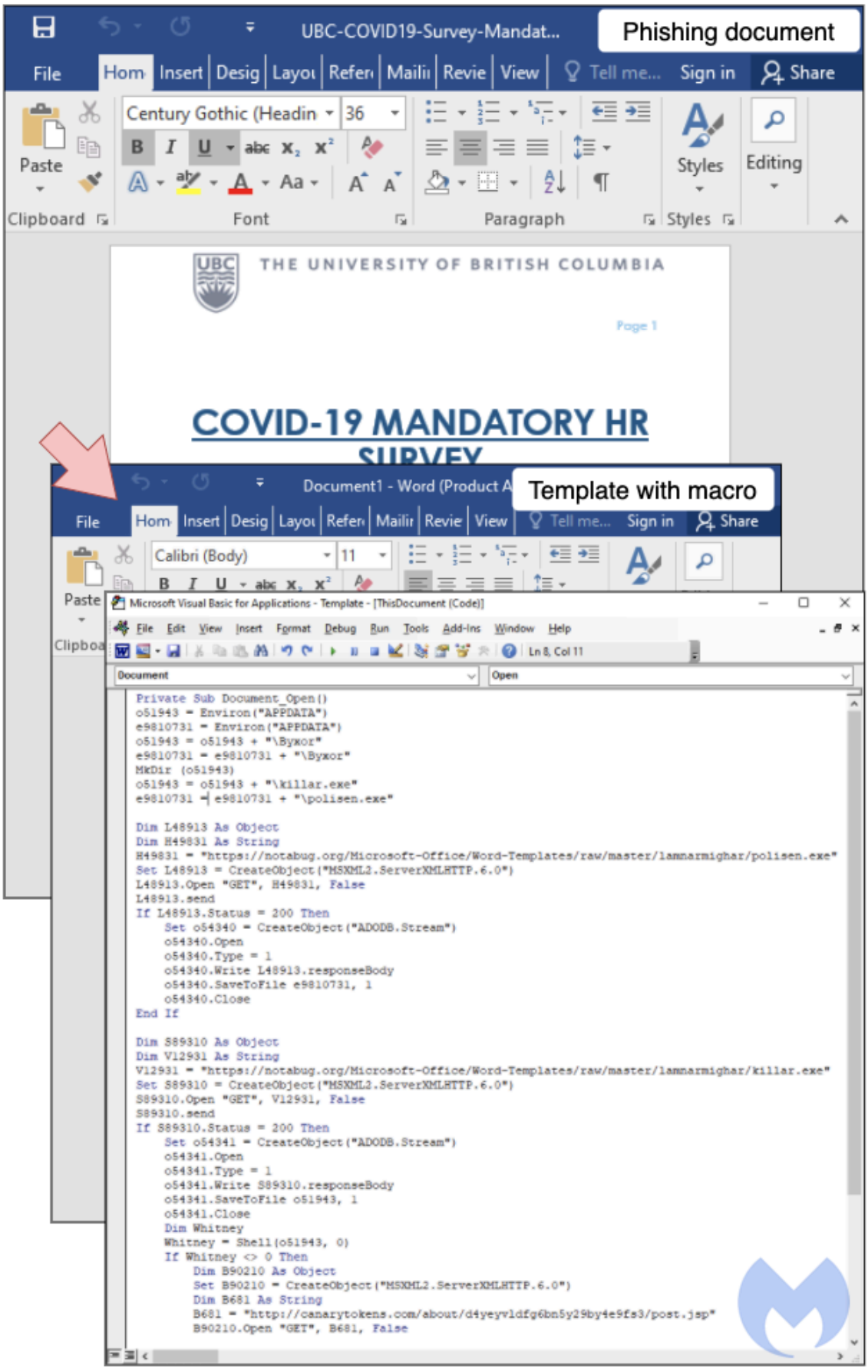
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

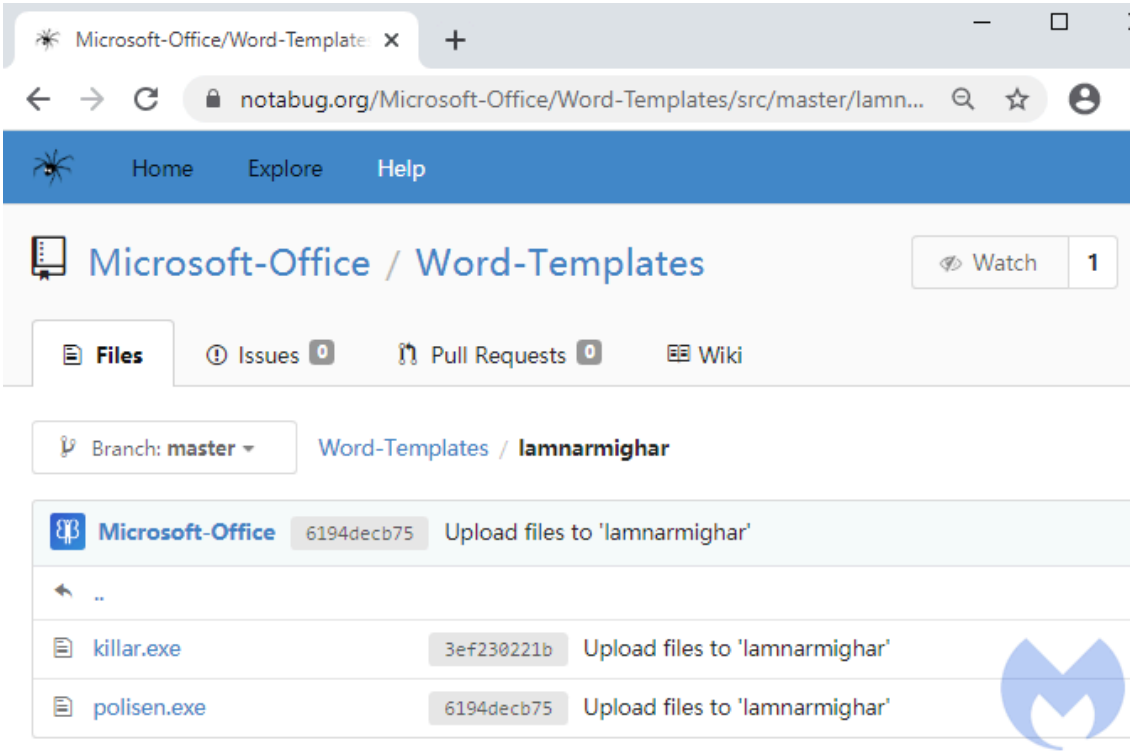
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



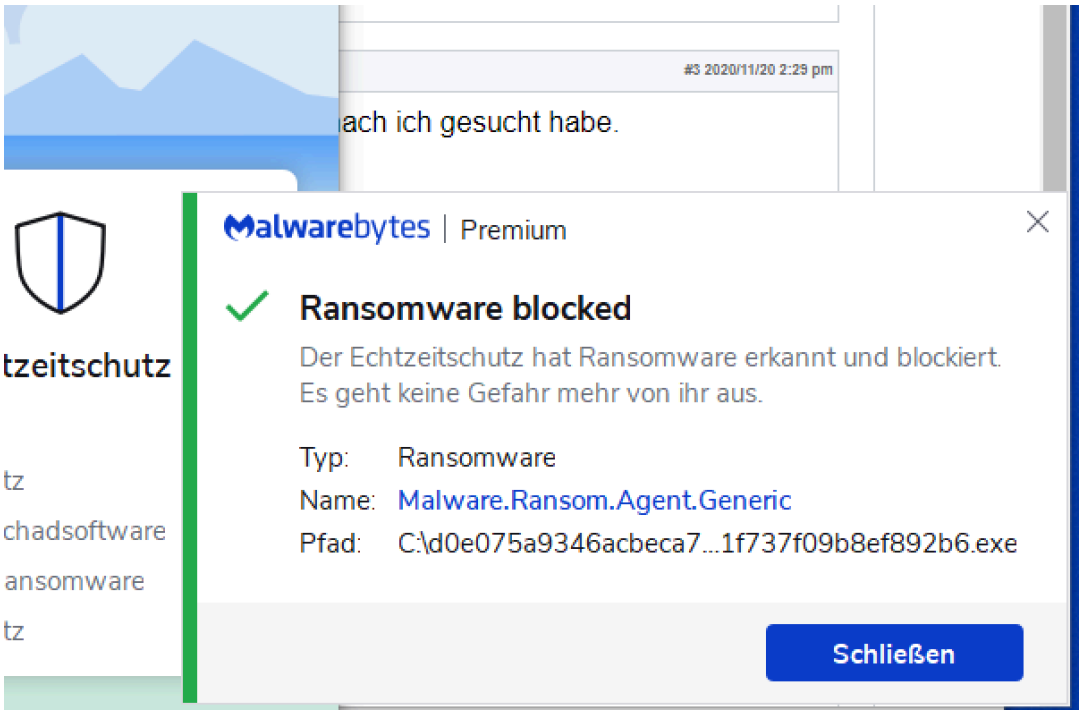
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

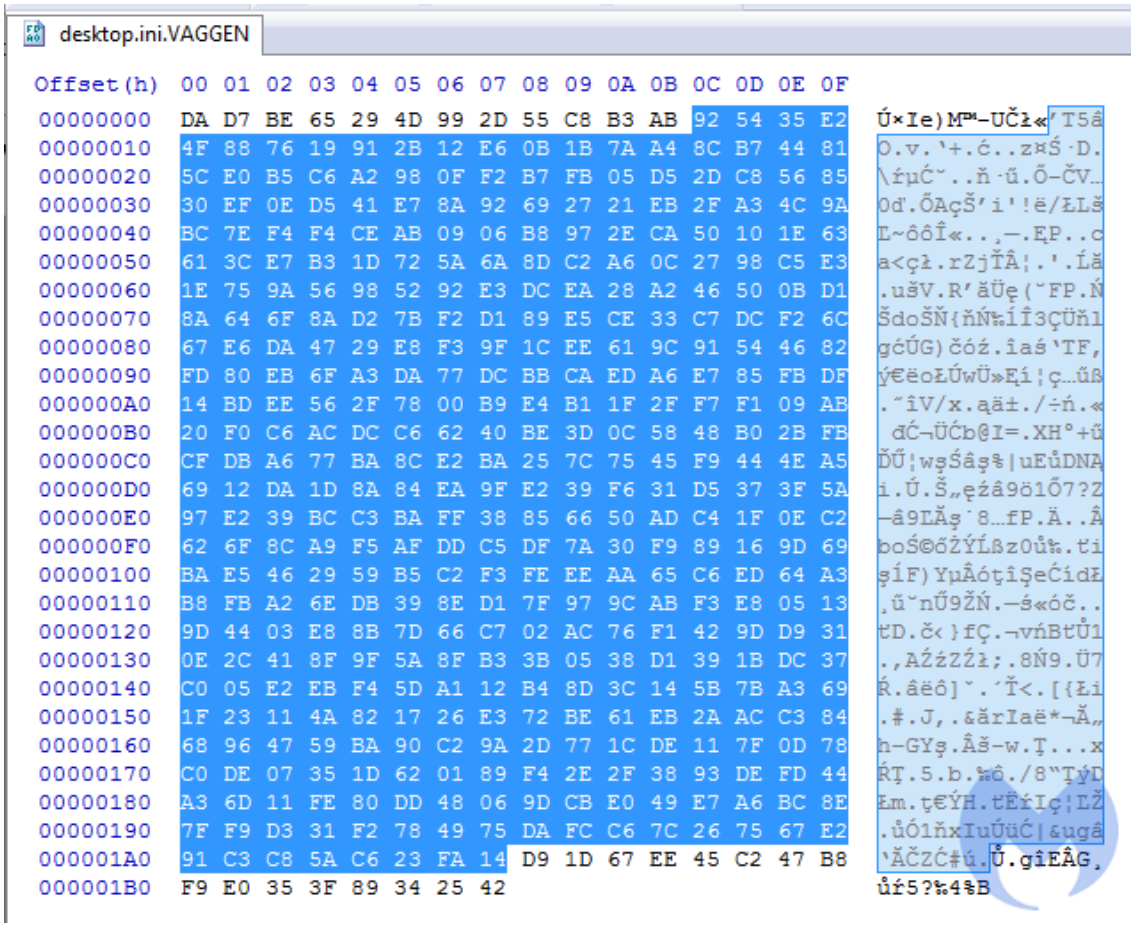
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

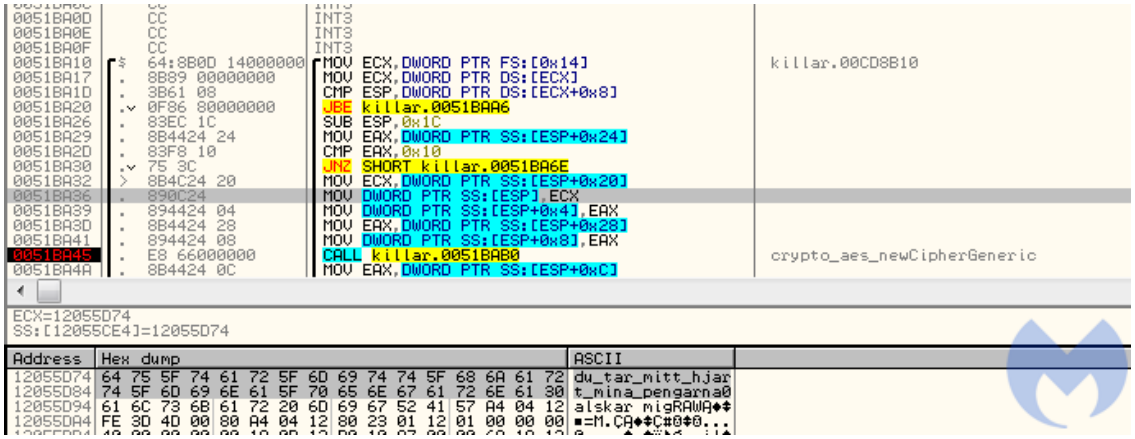


The content of the output file (with .VAGGEN extension) contains:

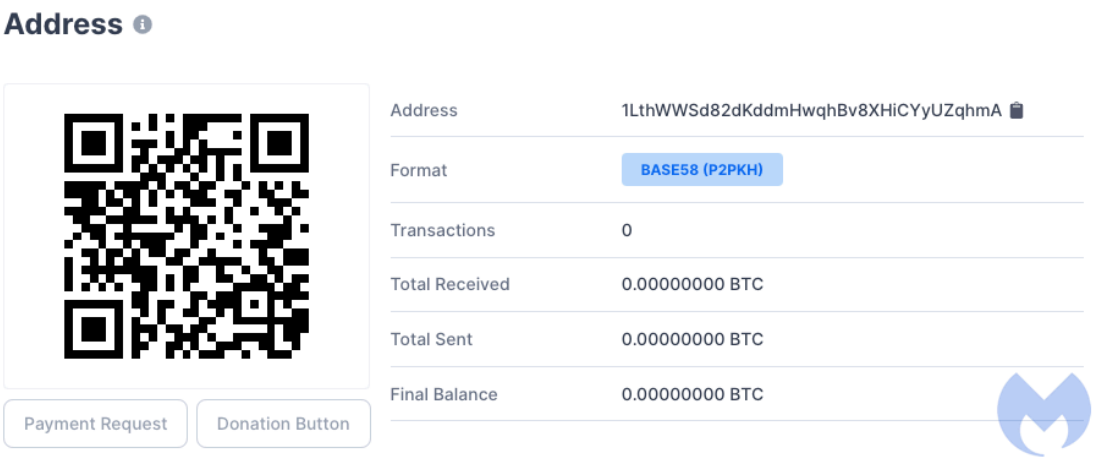
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

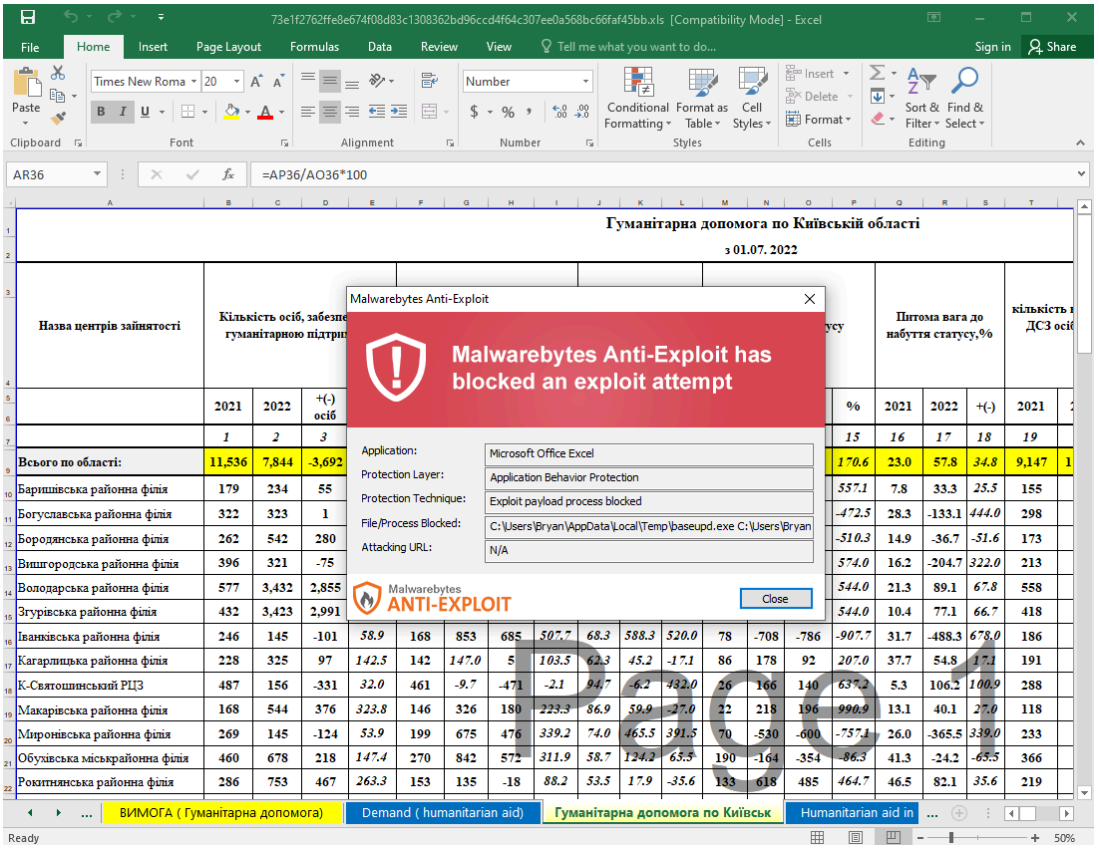
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

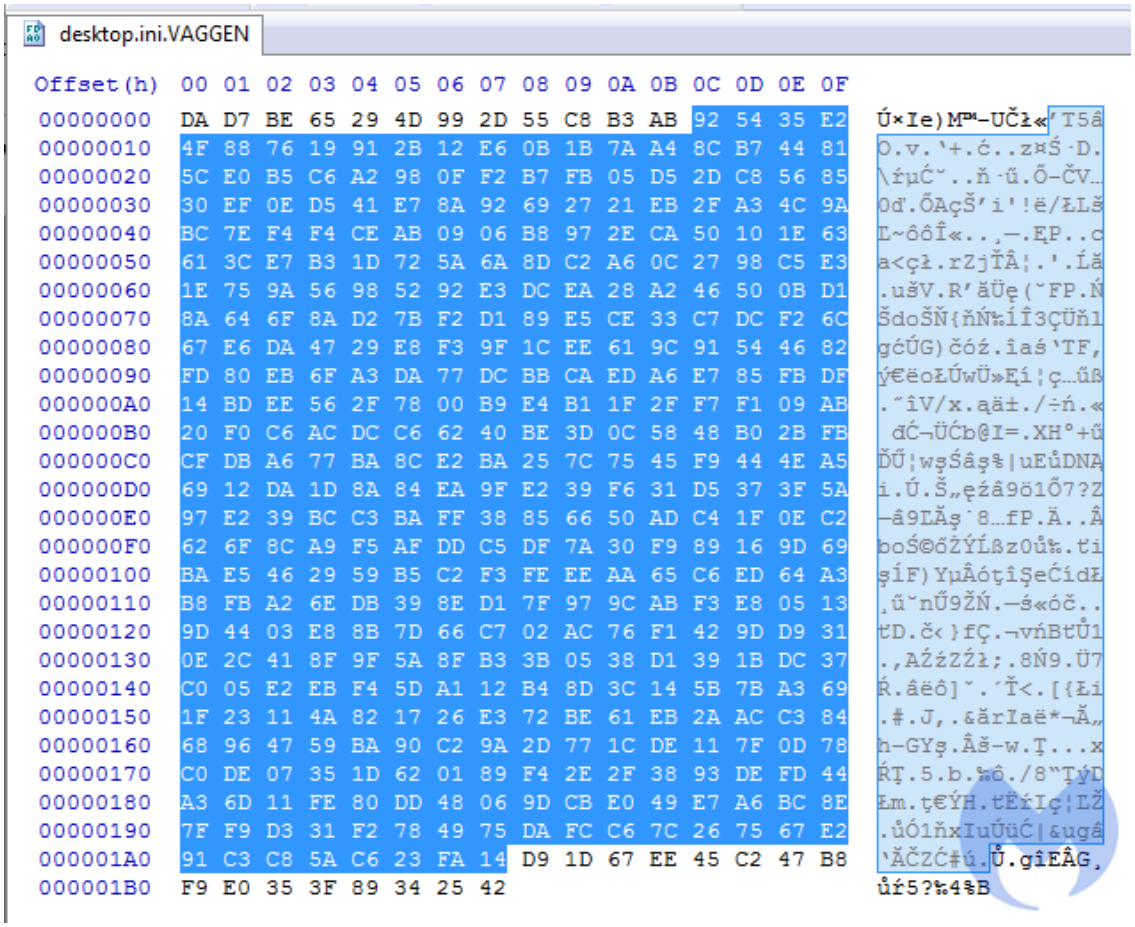
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

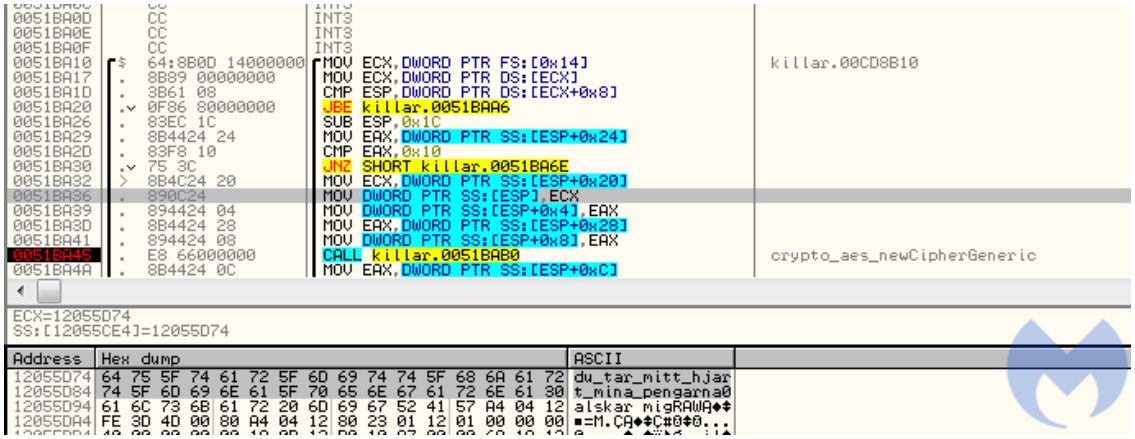


The content of the output file (with .VAGGEN extension) contains:

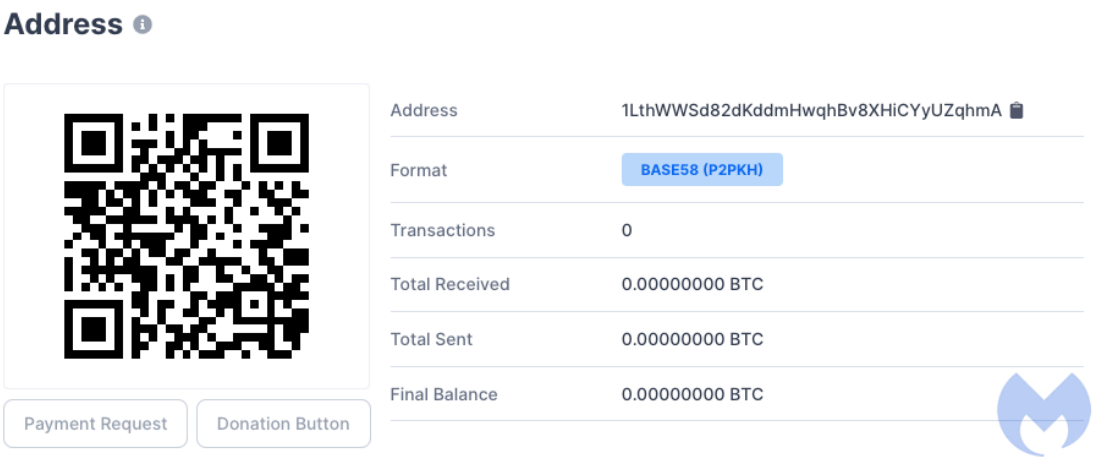
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

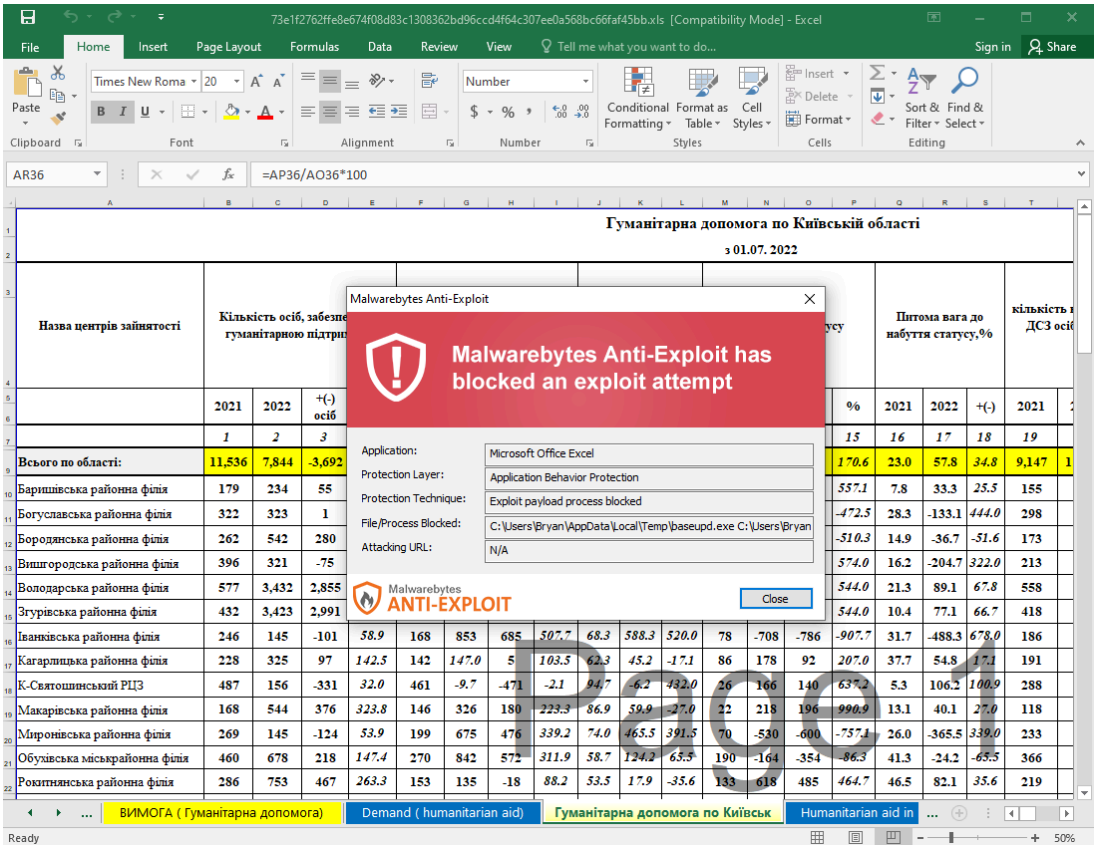
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

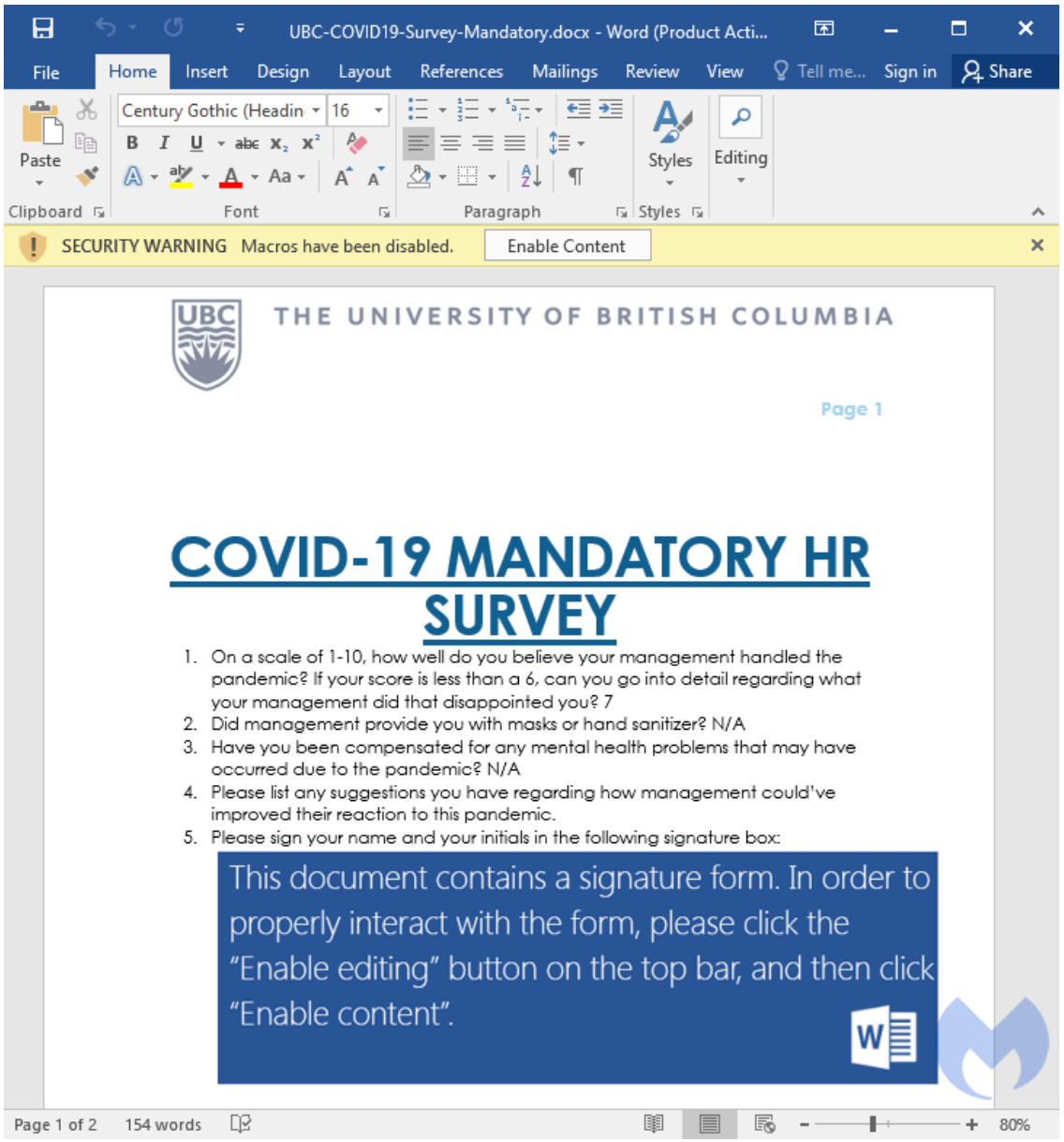
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

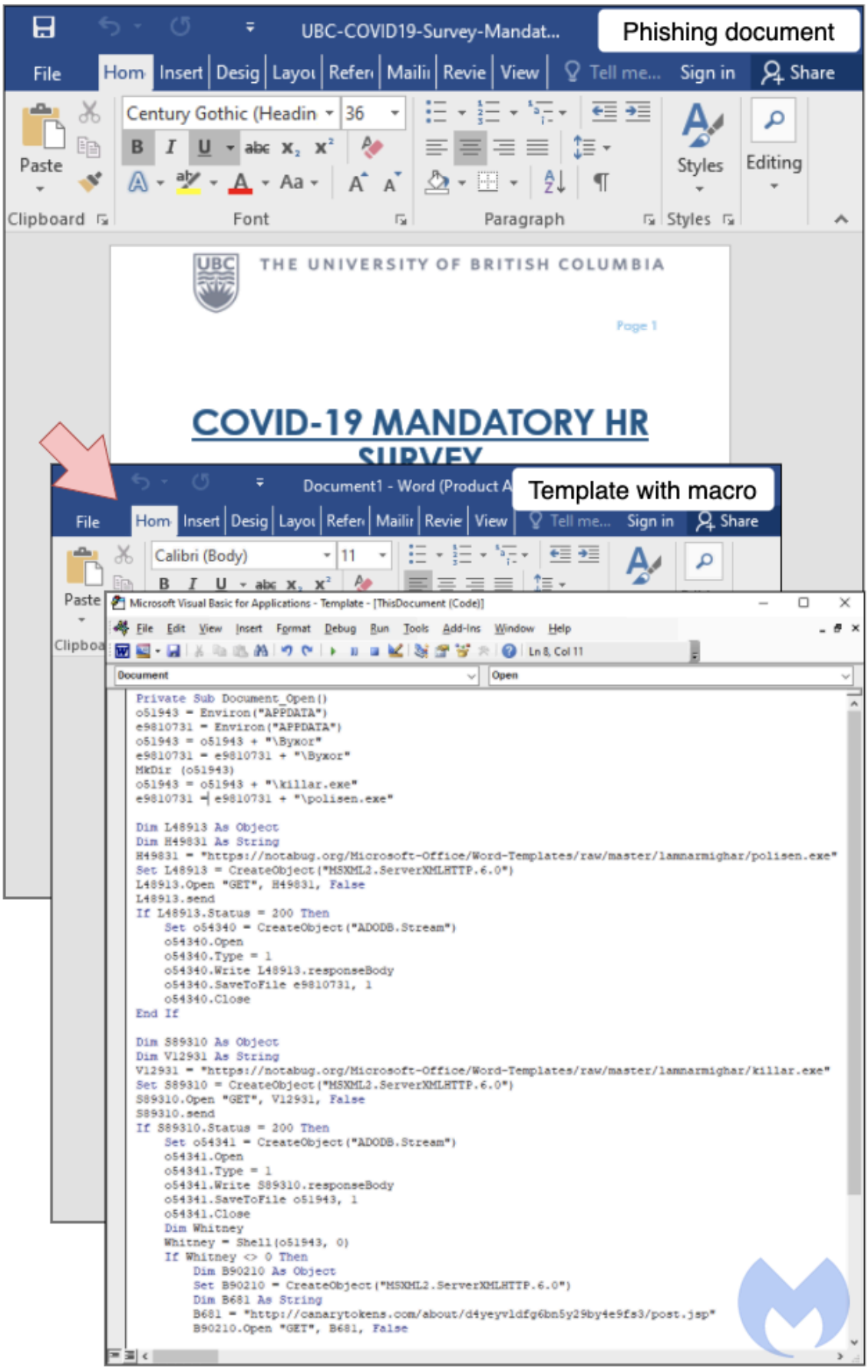
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

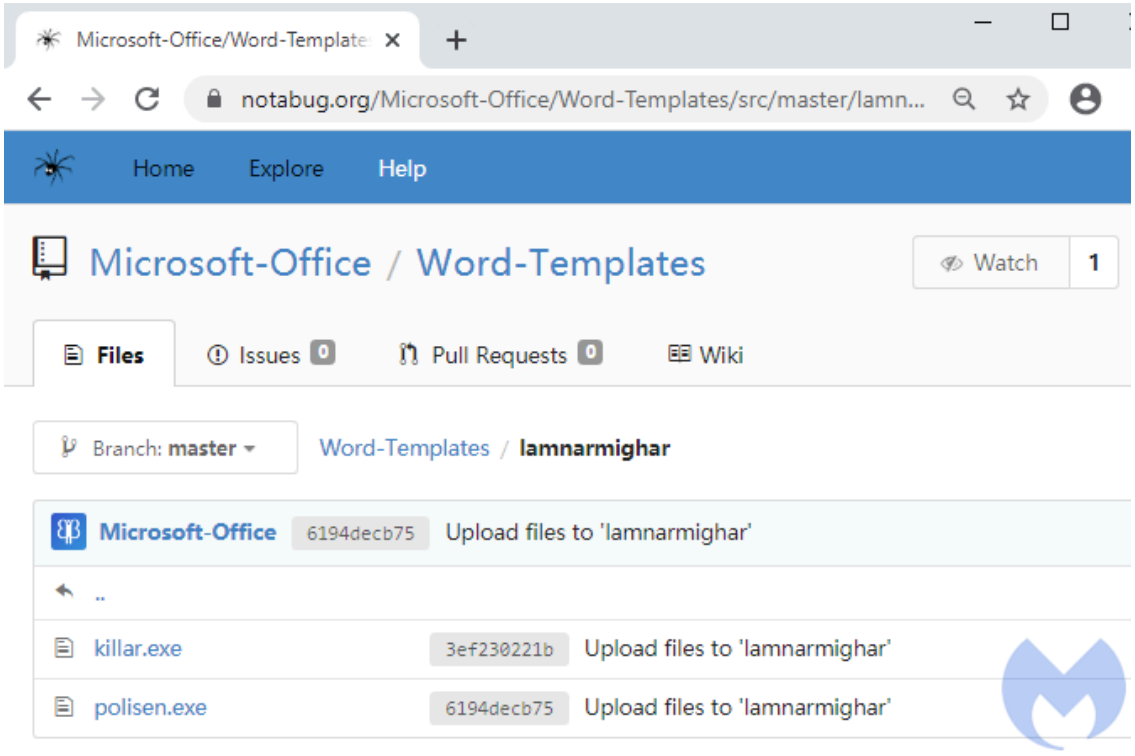
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



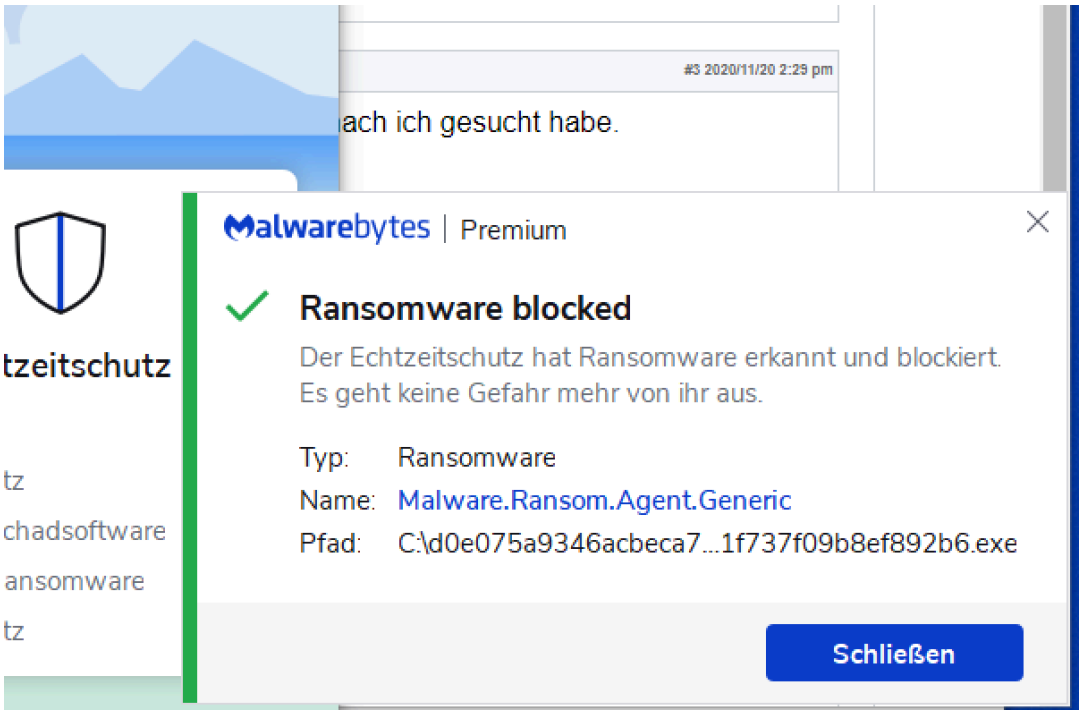
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

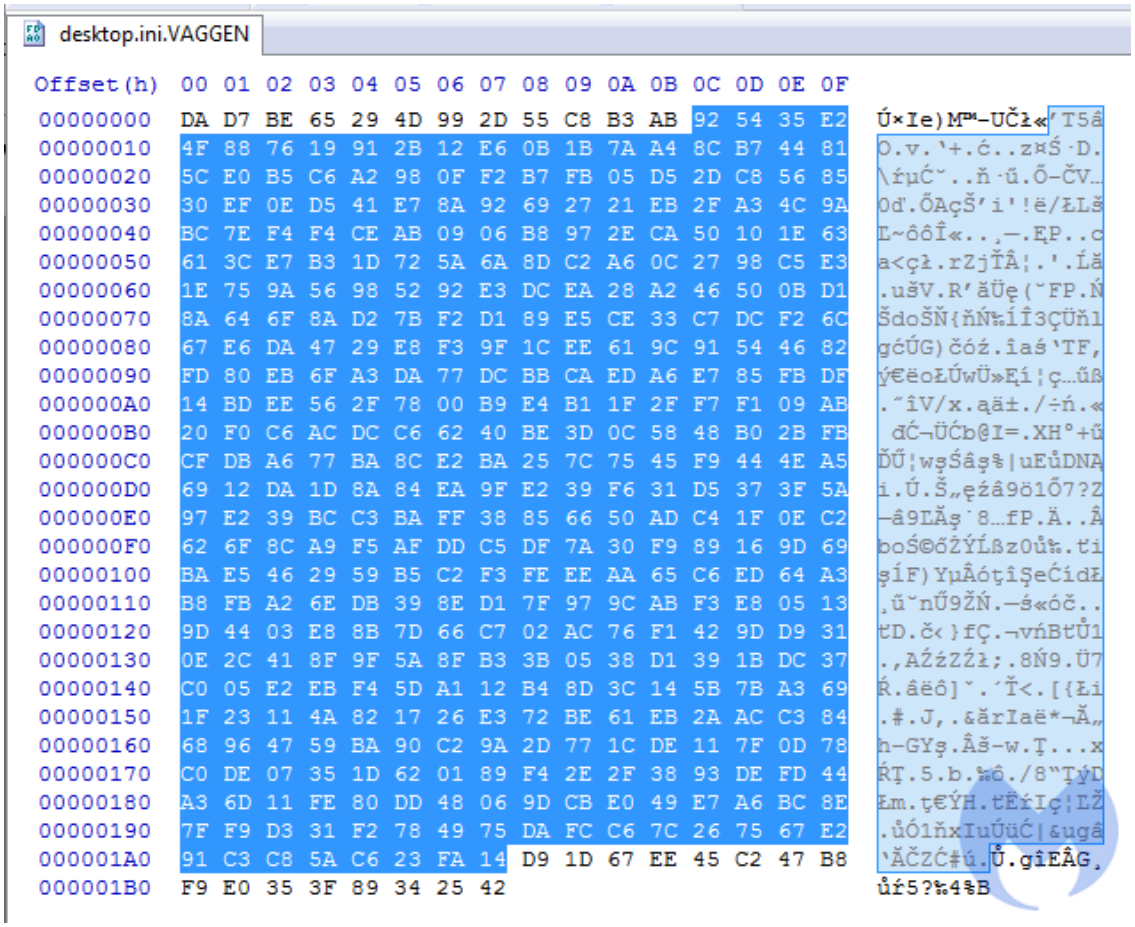
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

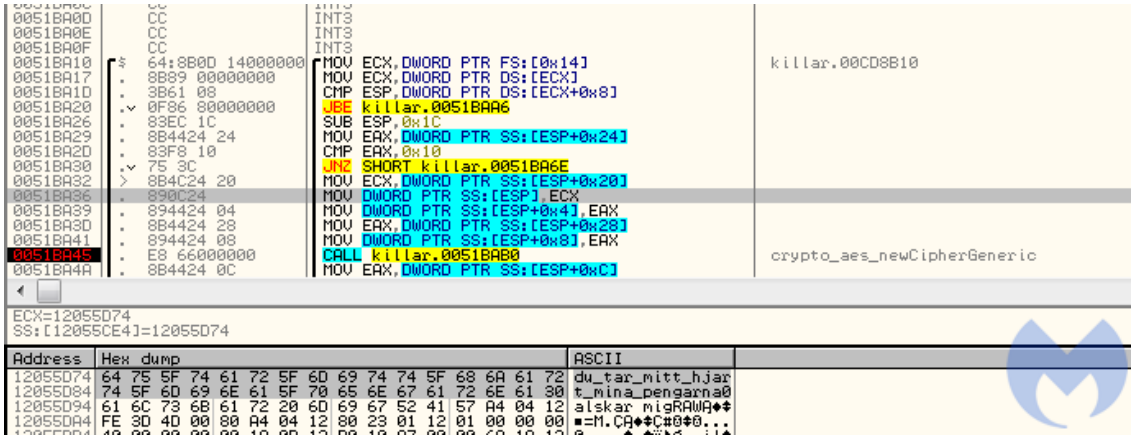


The content of the output file (with .VAGGEN extension) contains:

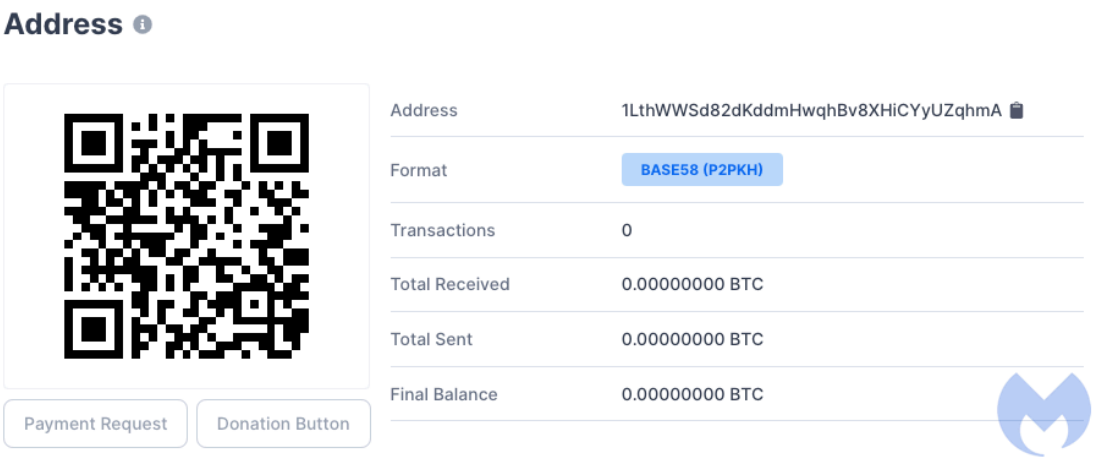
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

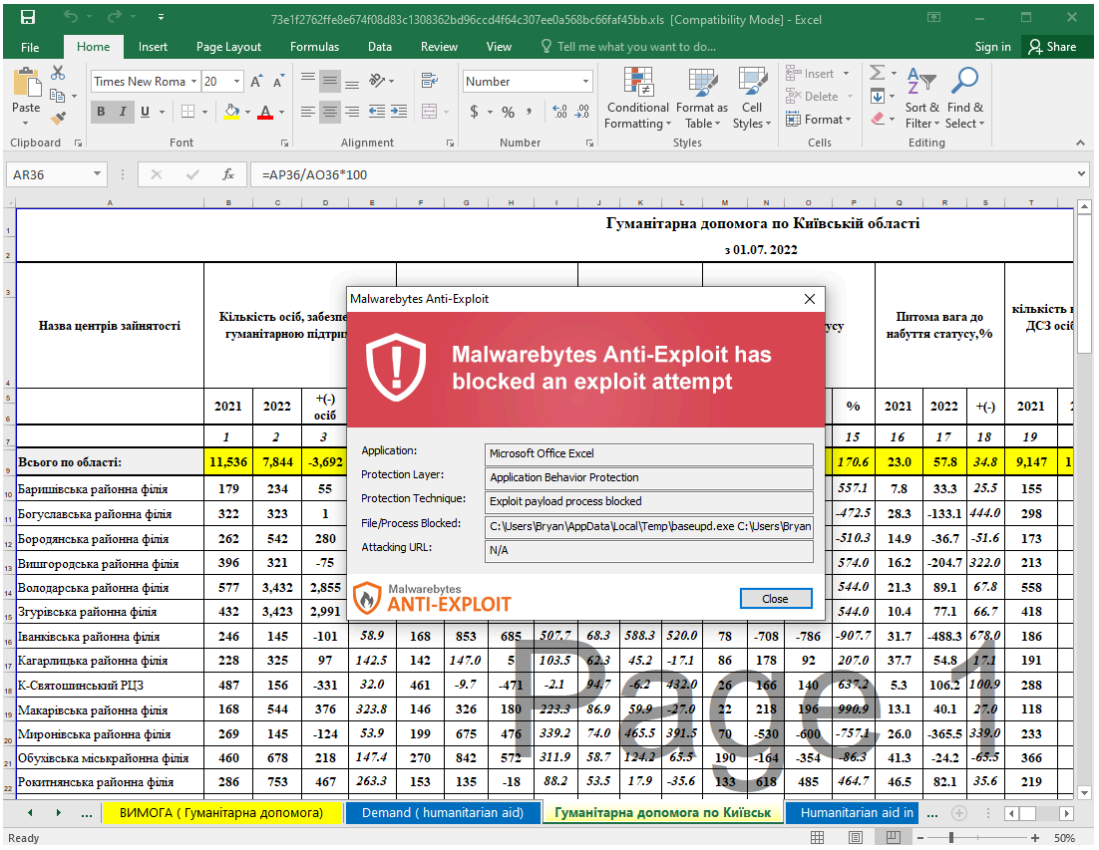
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

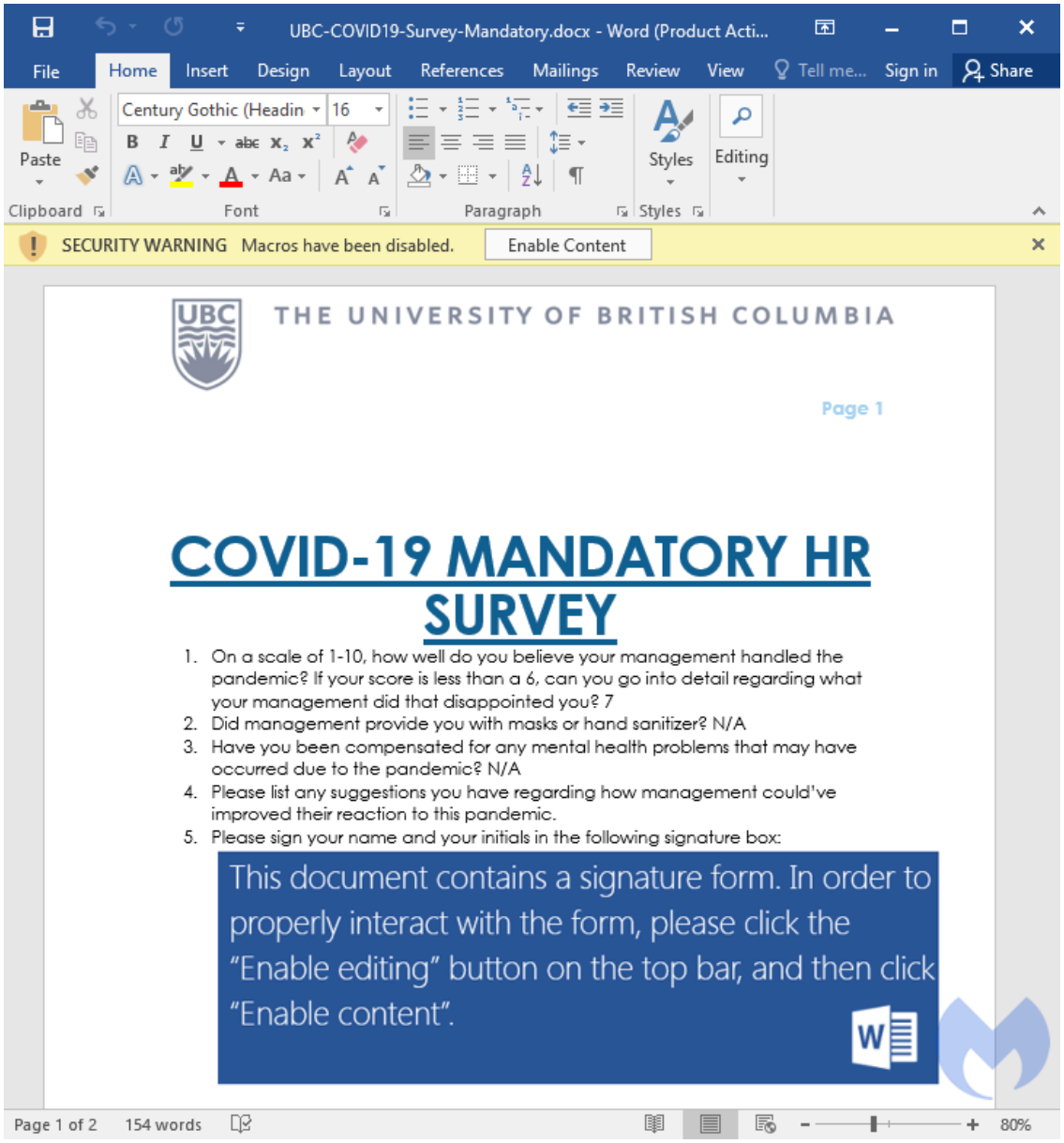
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

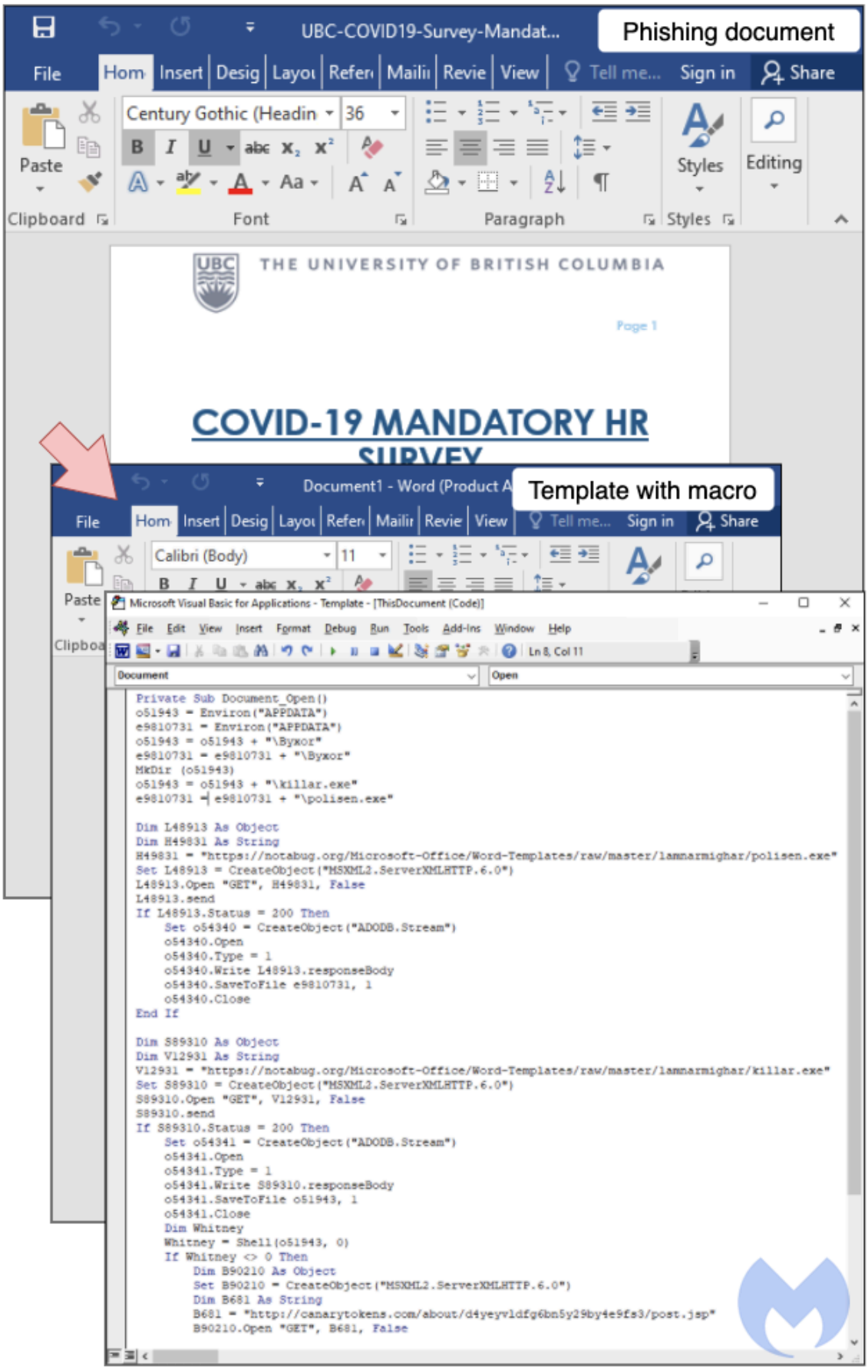
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

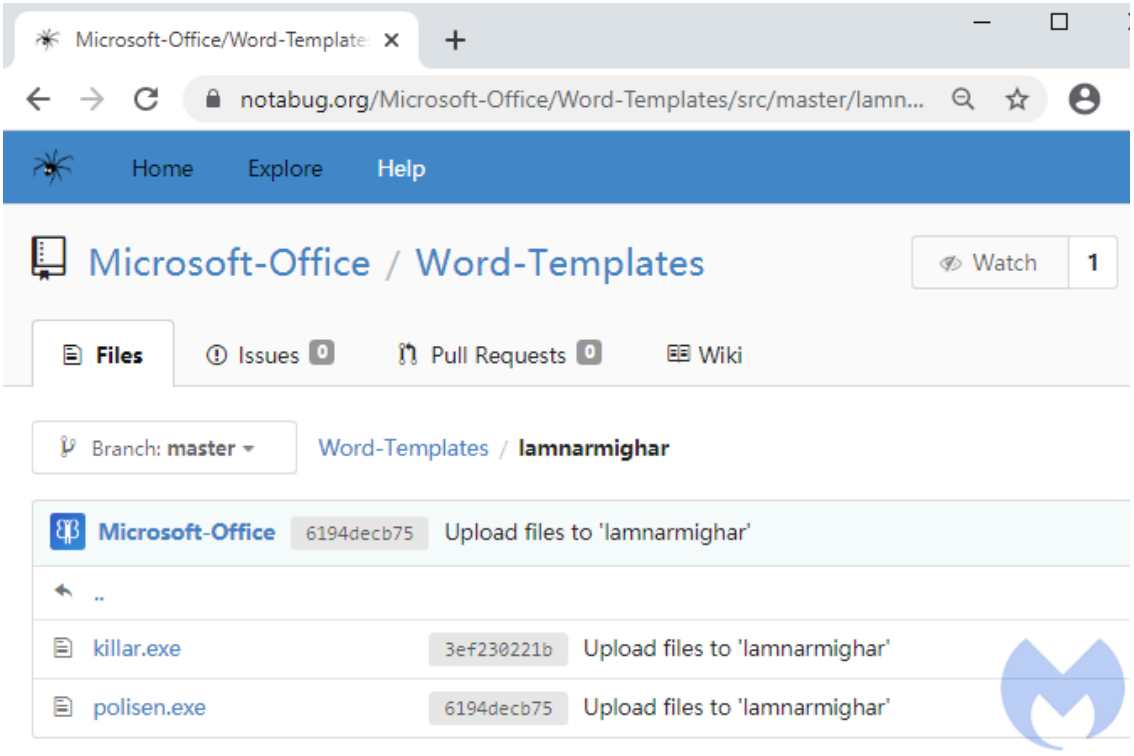
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



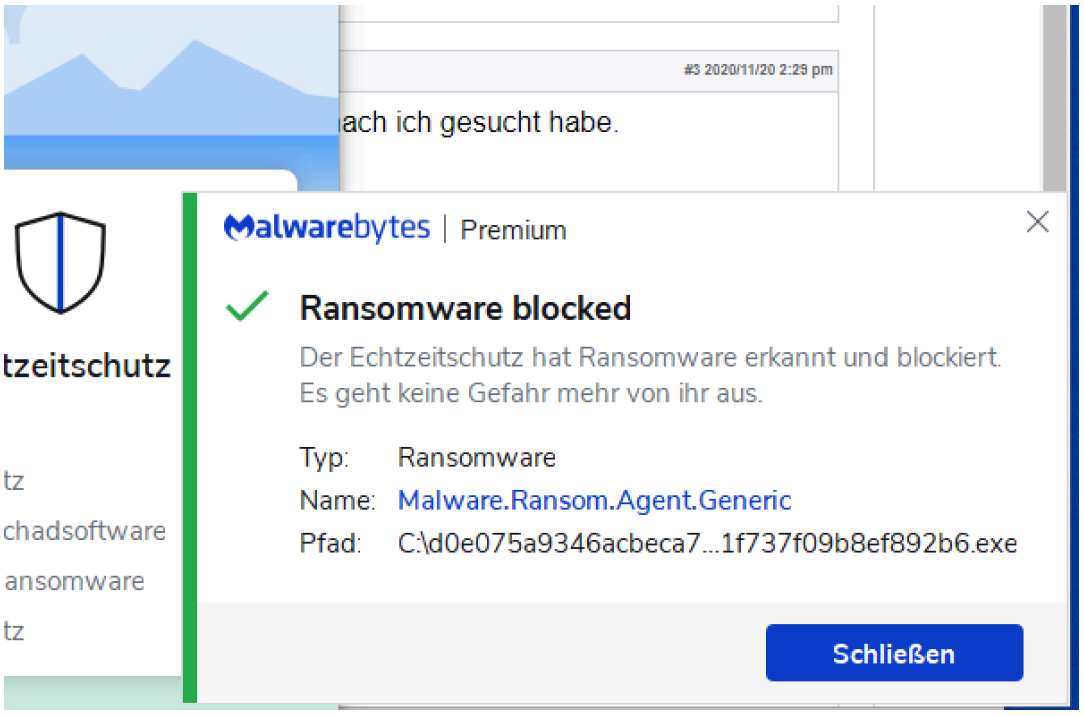
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

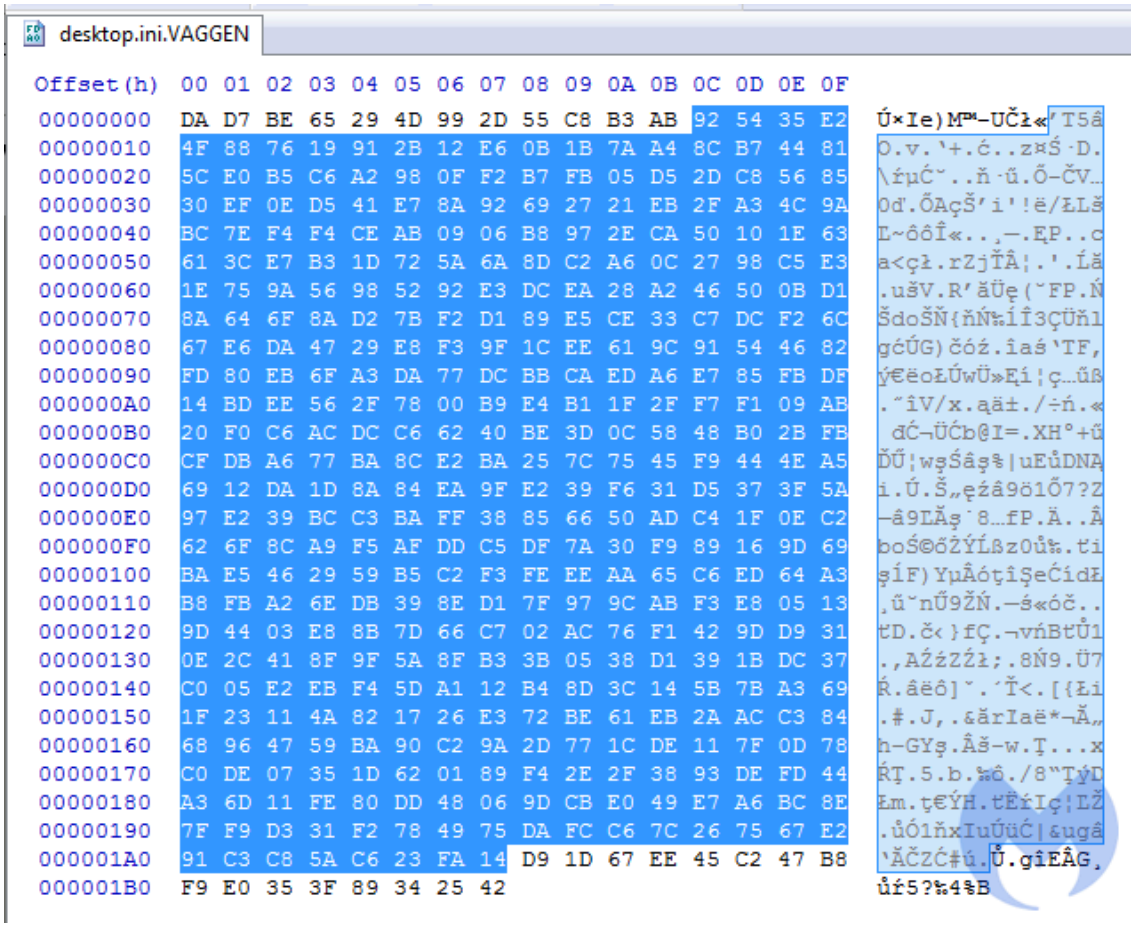
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

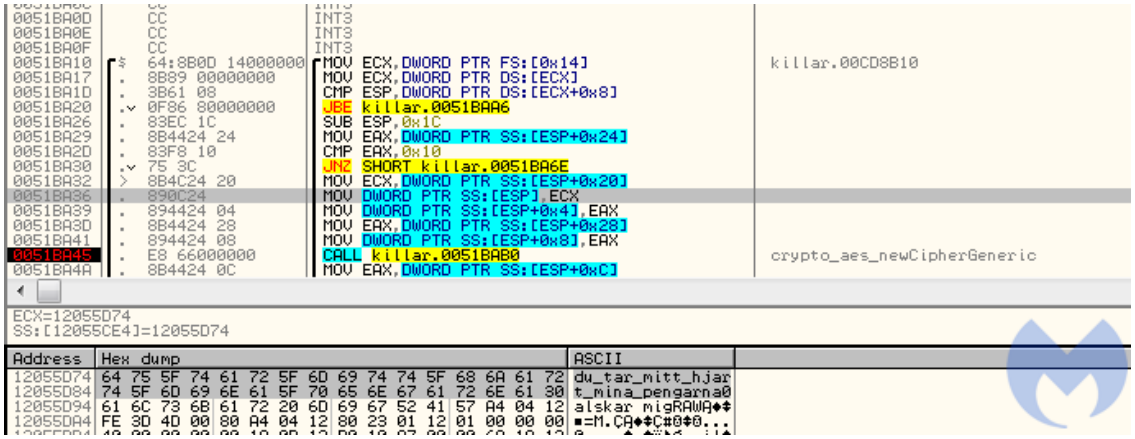


The content of the output file (with .VAGGEN extension) contains:

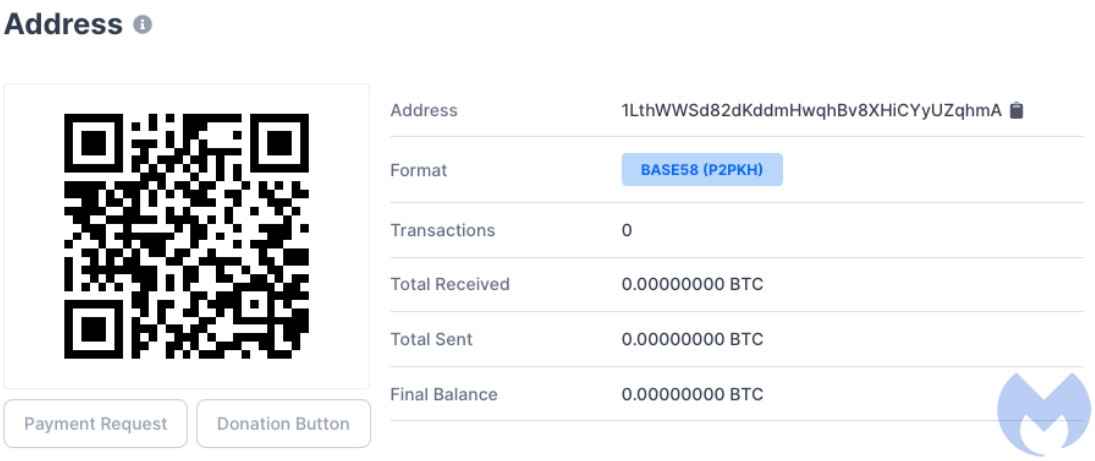
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

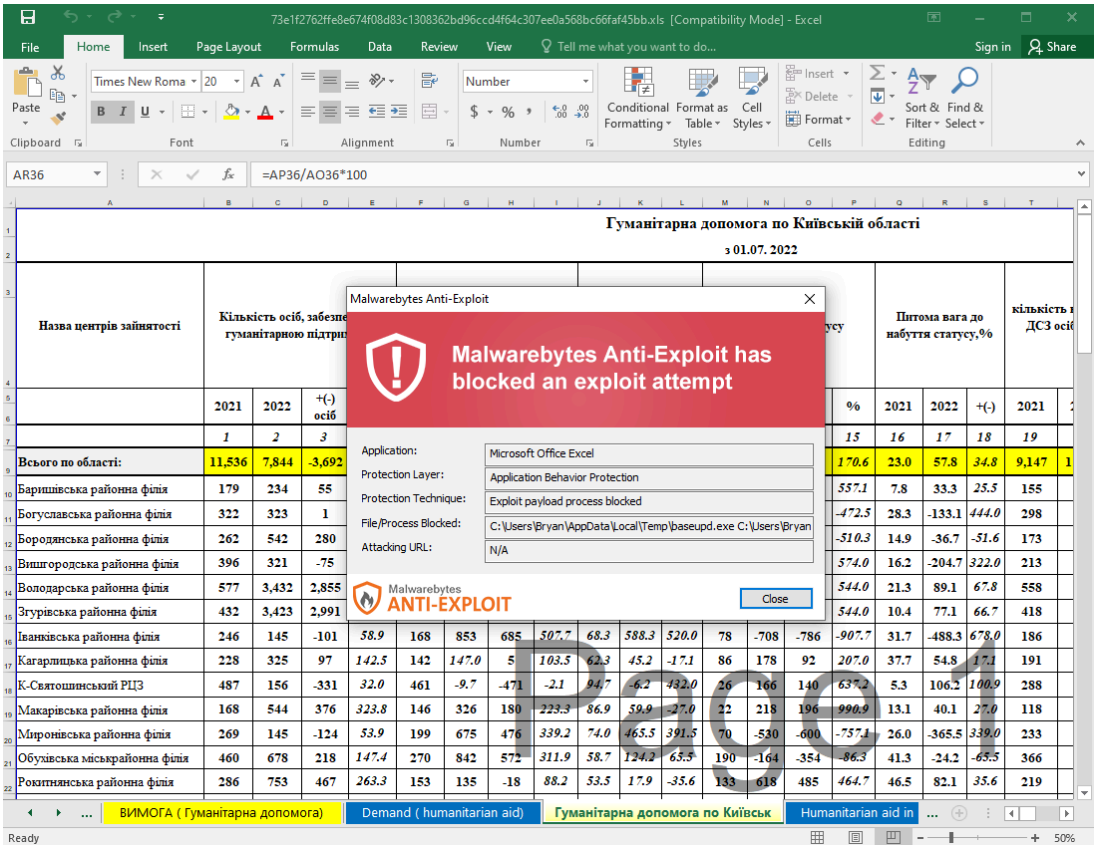
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

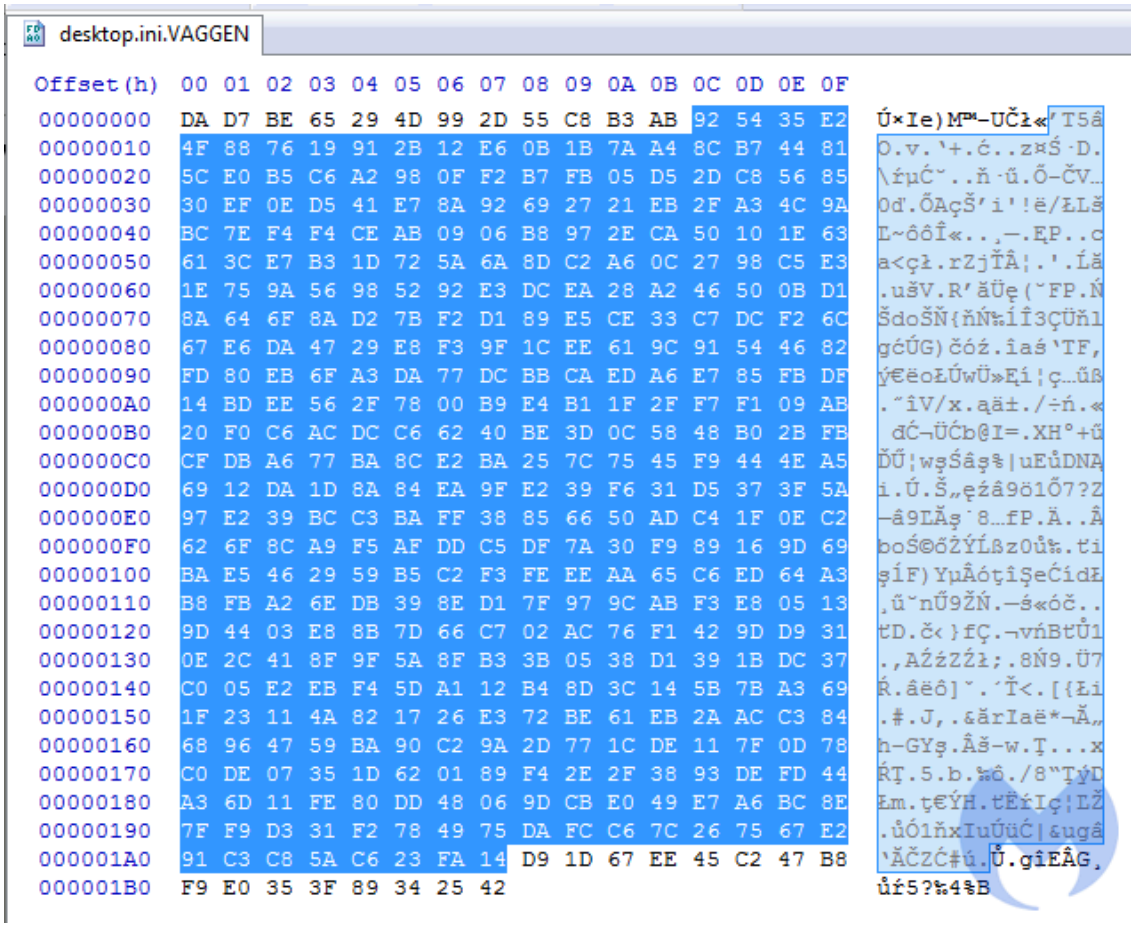
```
46  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
47    runtime_morestack_noctxt();
48  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
49  ((void (*)(void))loc_45CDA0)();
50  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
51  v38 = v4;
52  v30 = v5;
53  v31 = v6;
54  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
55  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
56  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
57  v32 = v10;
58  v11 = *(void (__cdecl **)(int))(v10 + 16);
59  v39 = v12;
60  v11(v12);
61  v29 = v13;
62  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
63  v40 = v15;
64  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
65  v17 = v27;
66  if ( v27 )
67  {
68    if ( v27 )
69      v17 = *(_DWORD *)(v27 + 4);
70    v43 = v17;
71    v44 = v28;
72    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
73  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

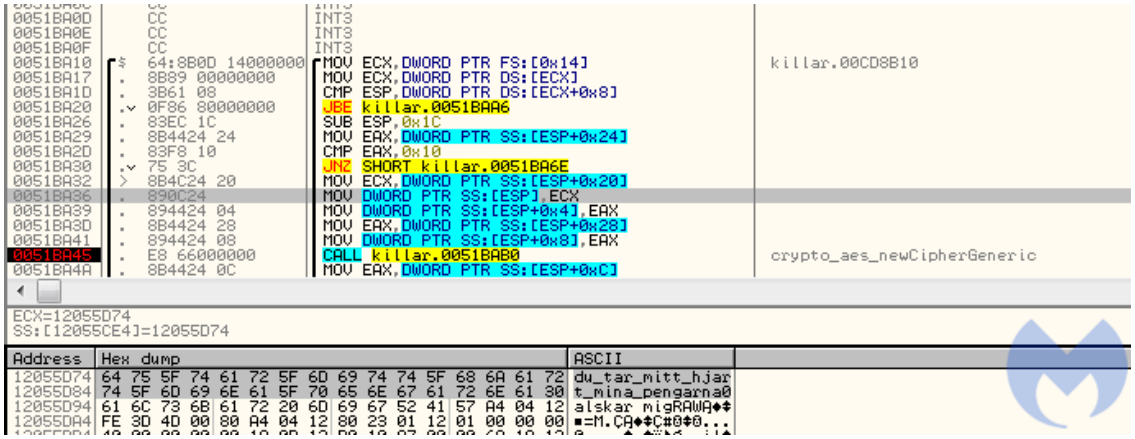


The content of the output file (with .VAGGEN extension) contains:

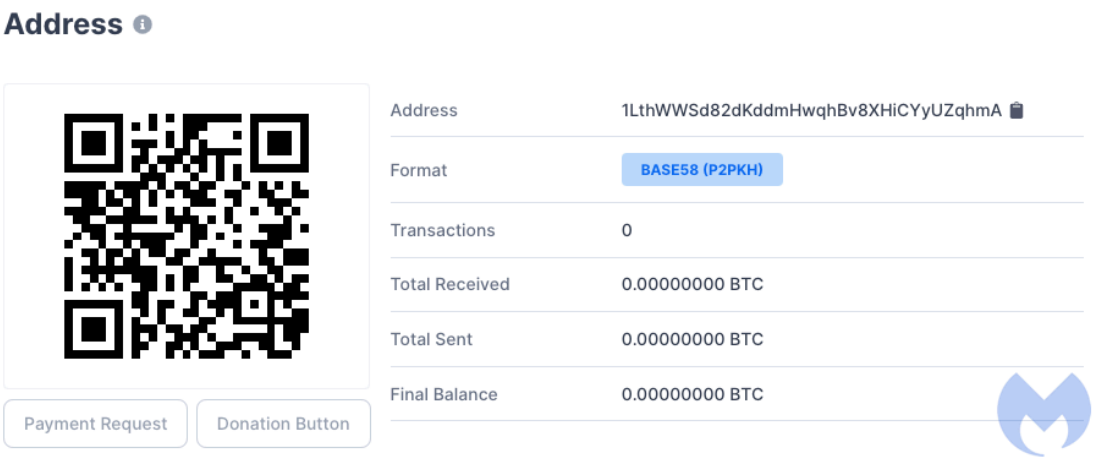
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

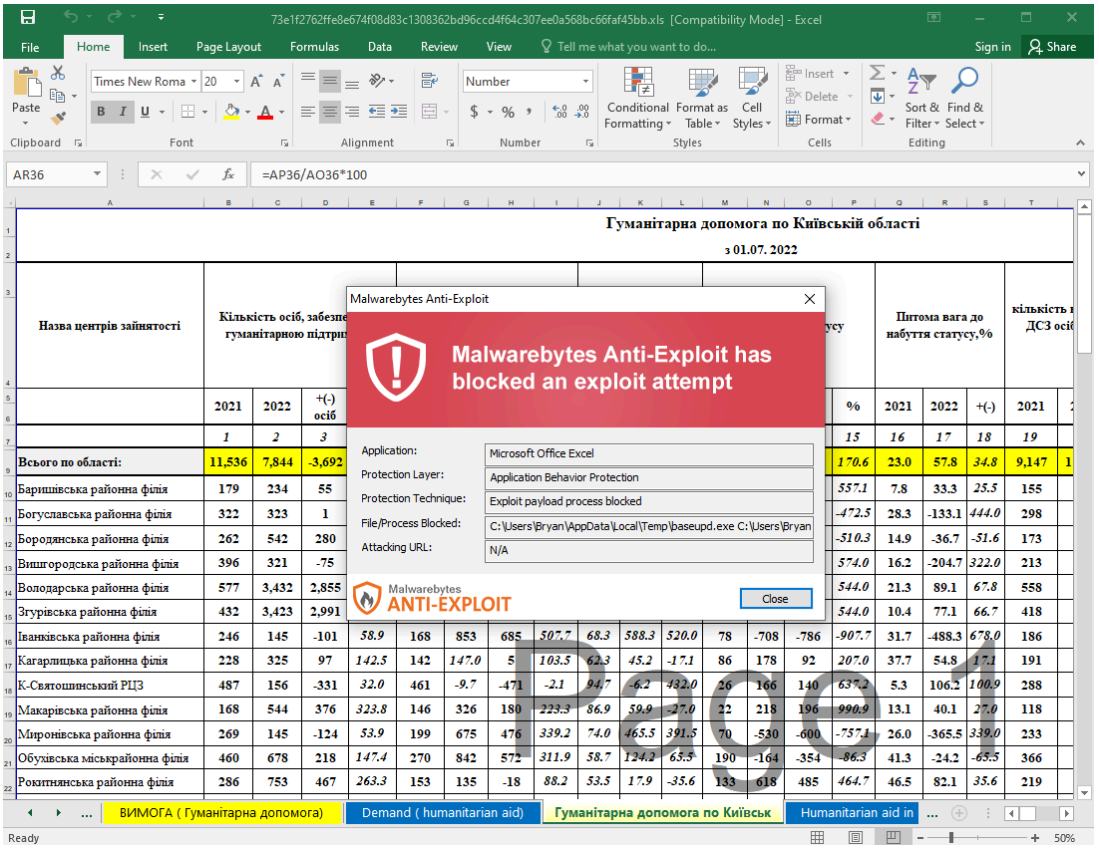
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

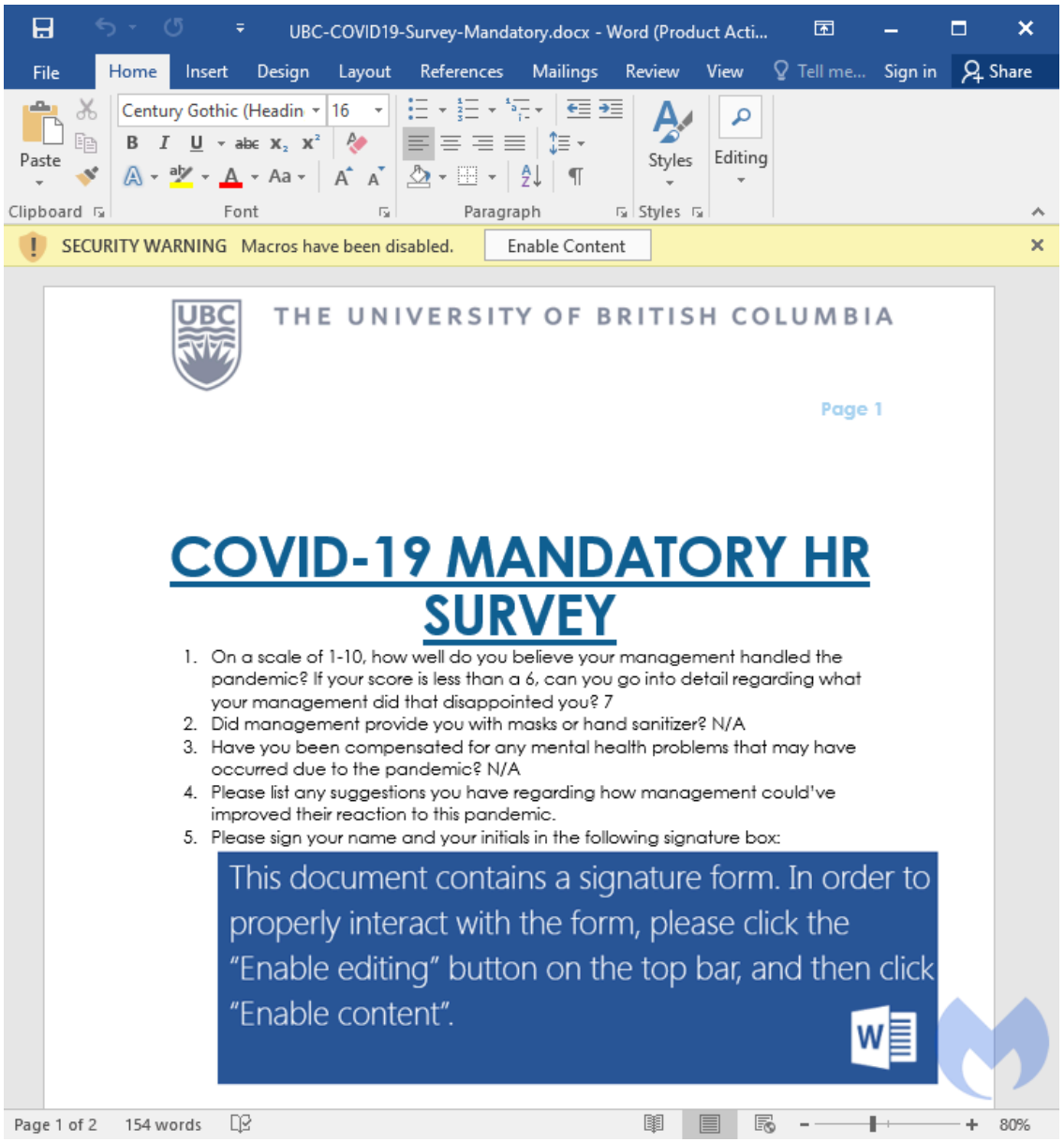
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

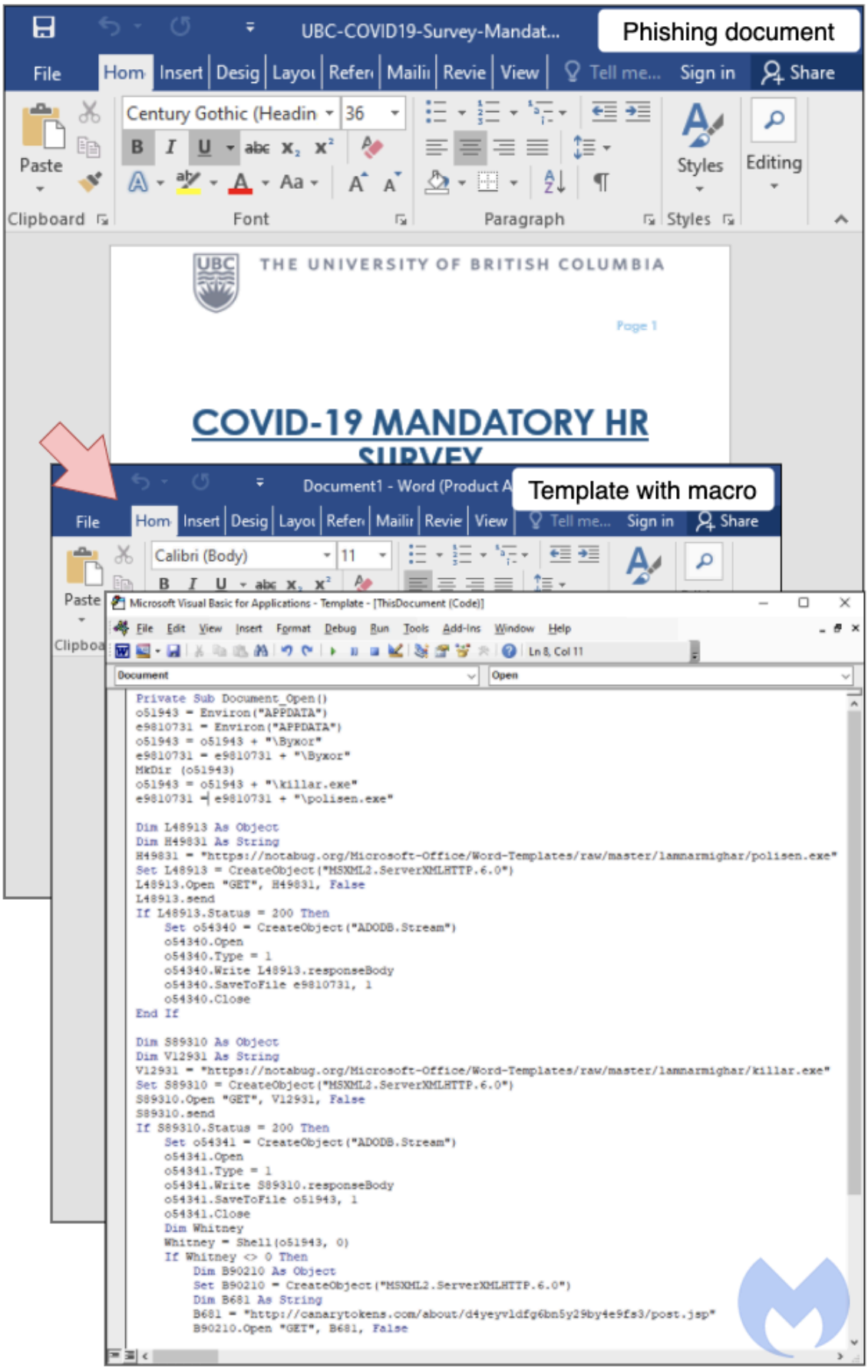
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

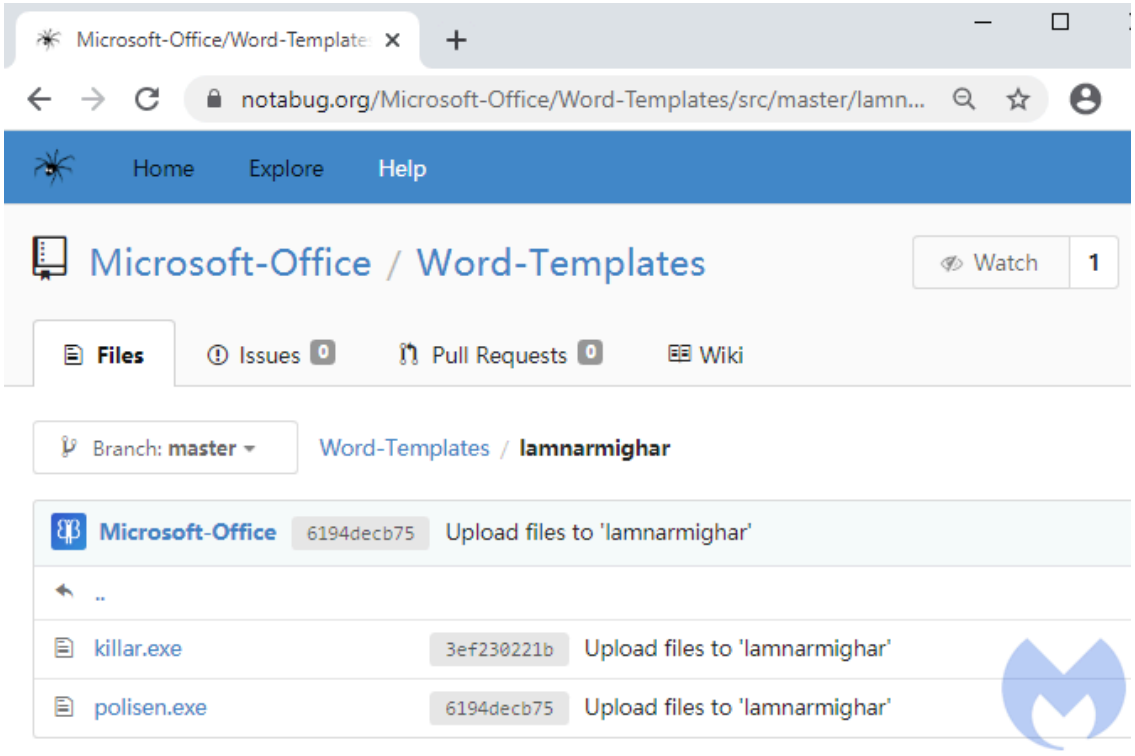
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



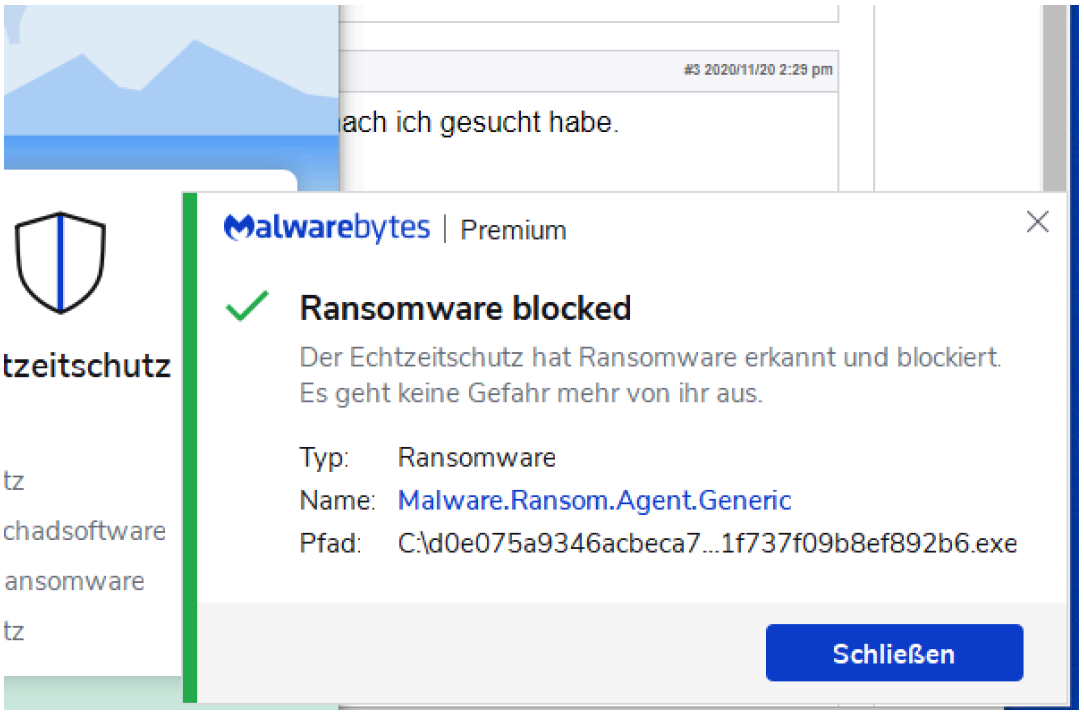
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

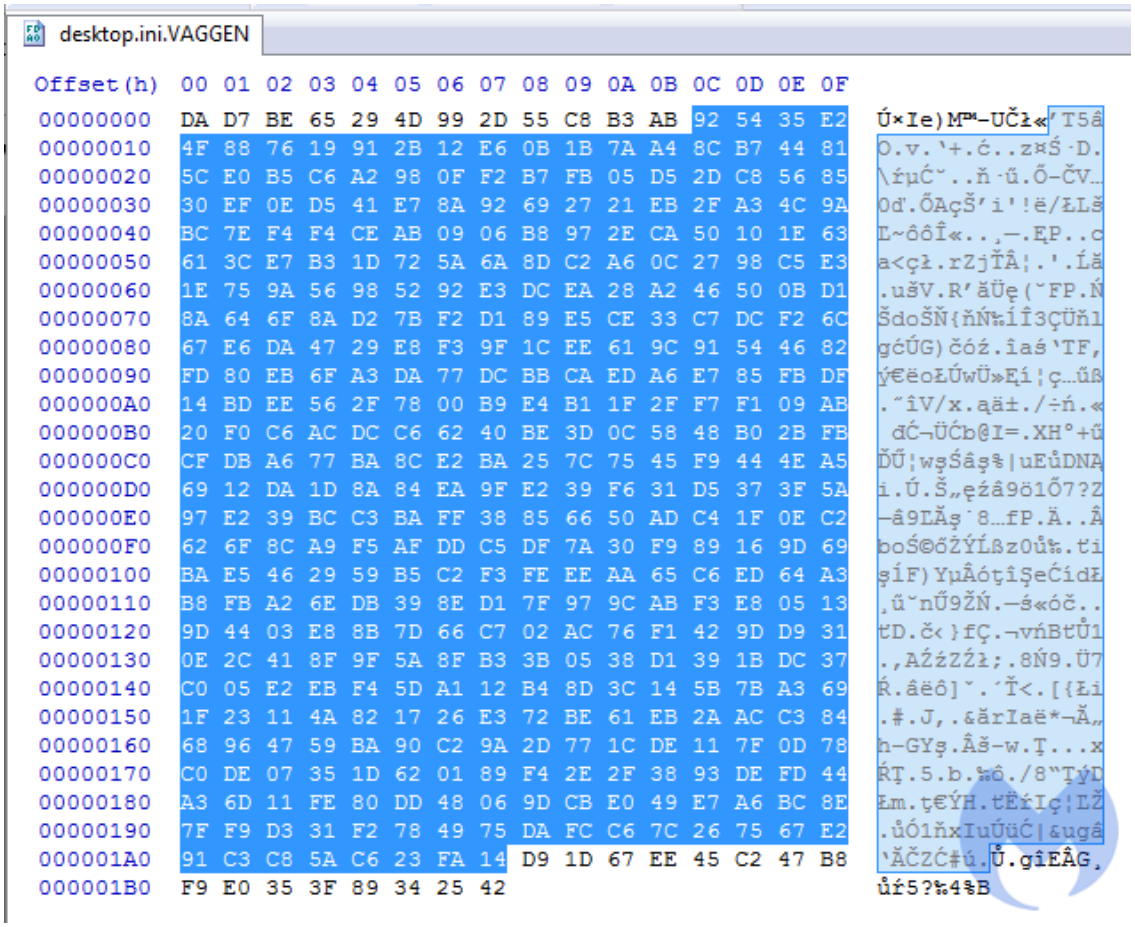
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

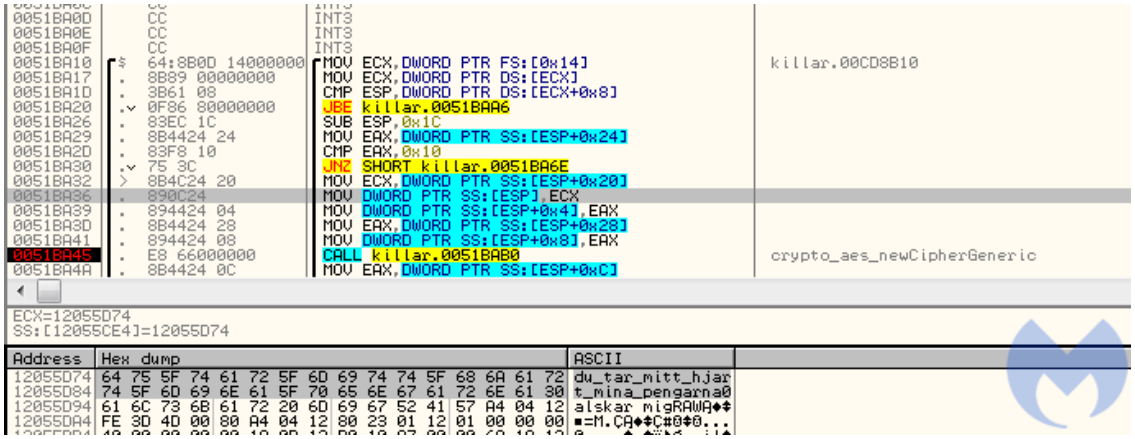


The content of the output file (with .VAGGEN extension) contains:

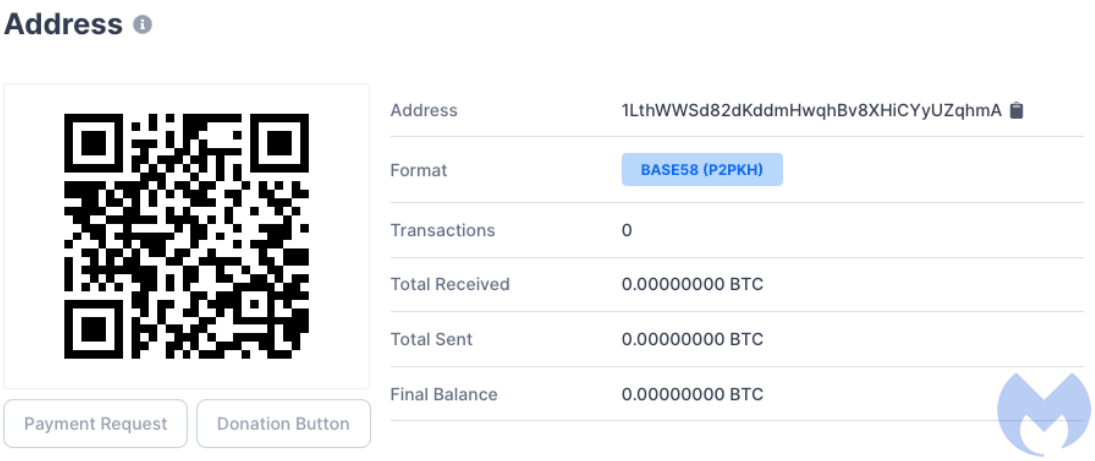
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

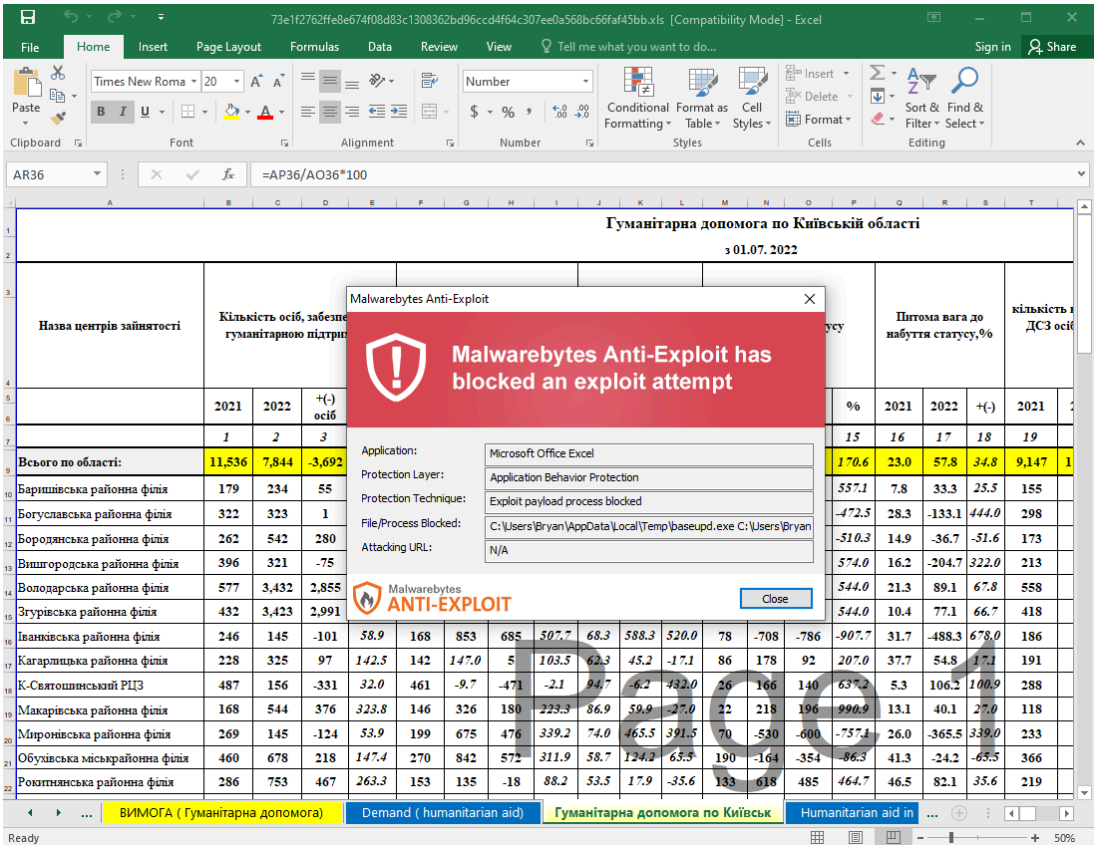
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

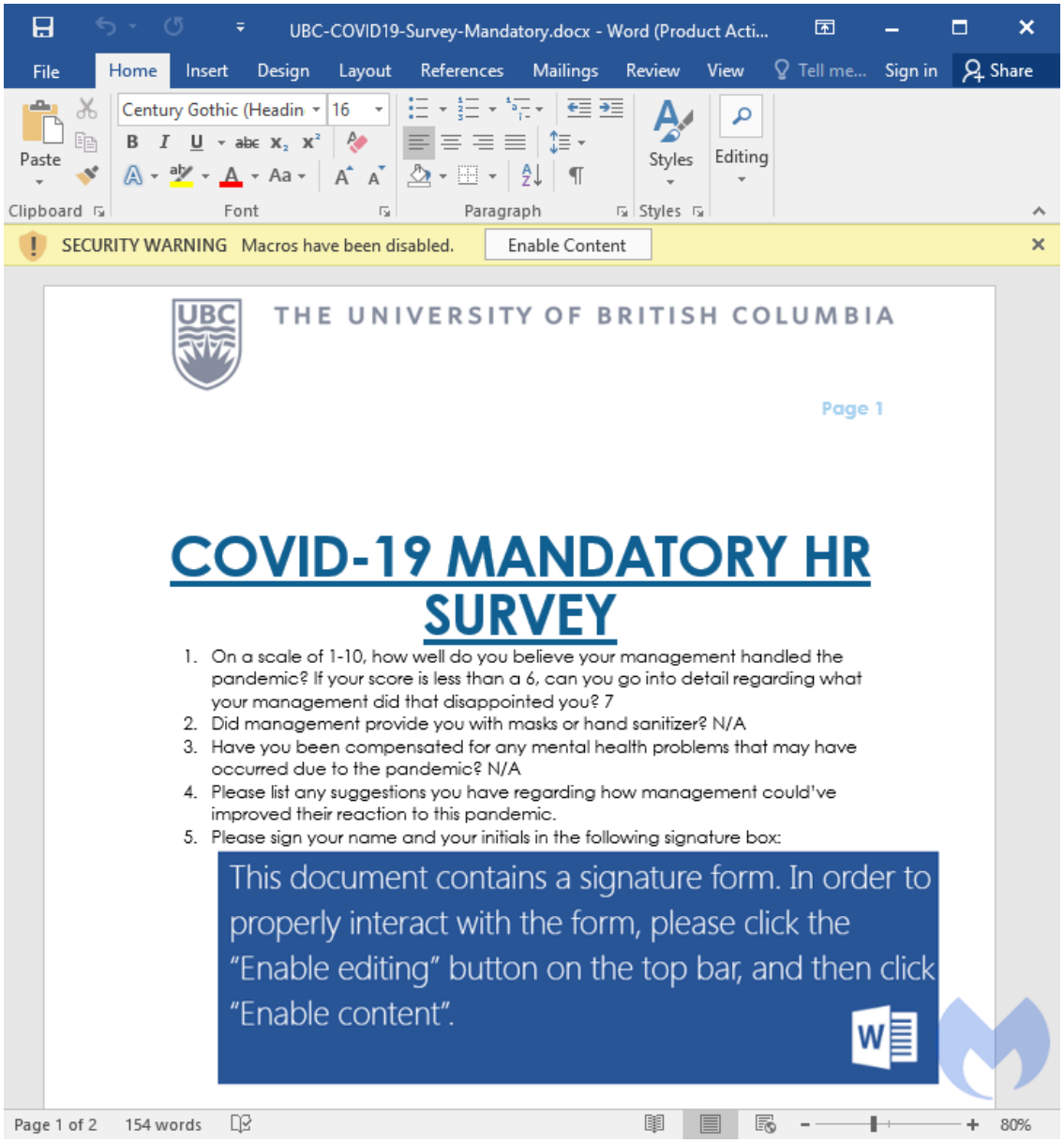
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

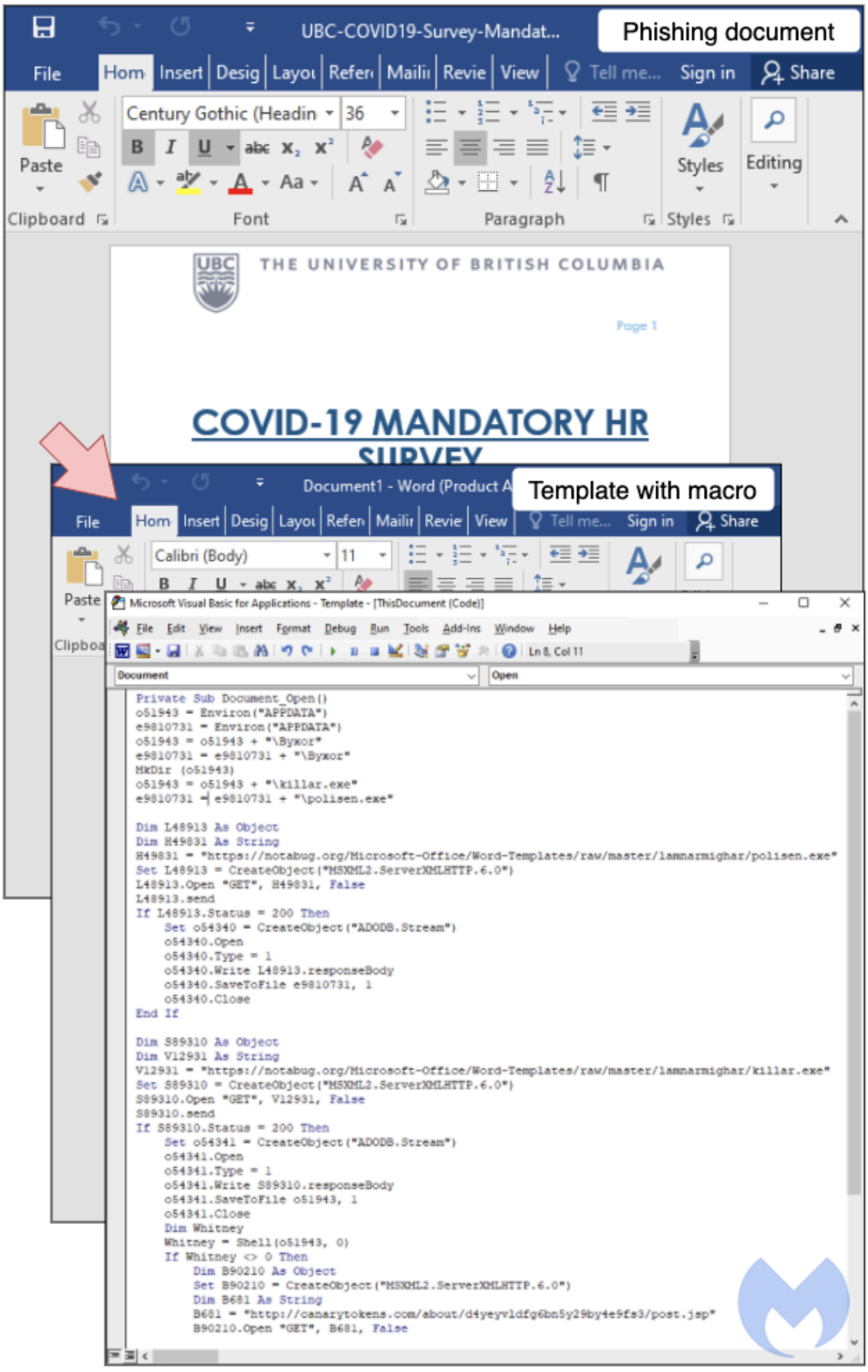
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

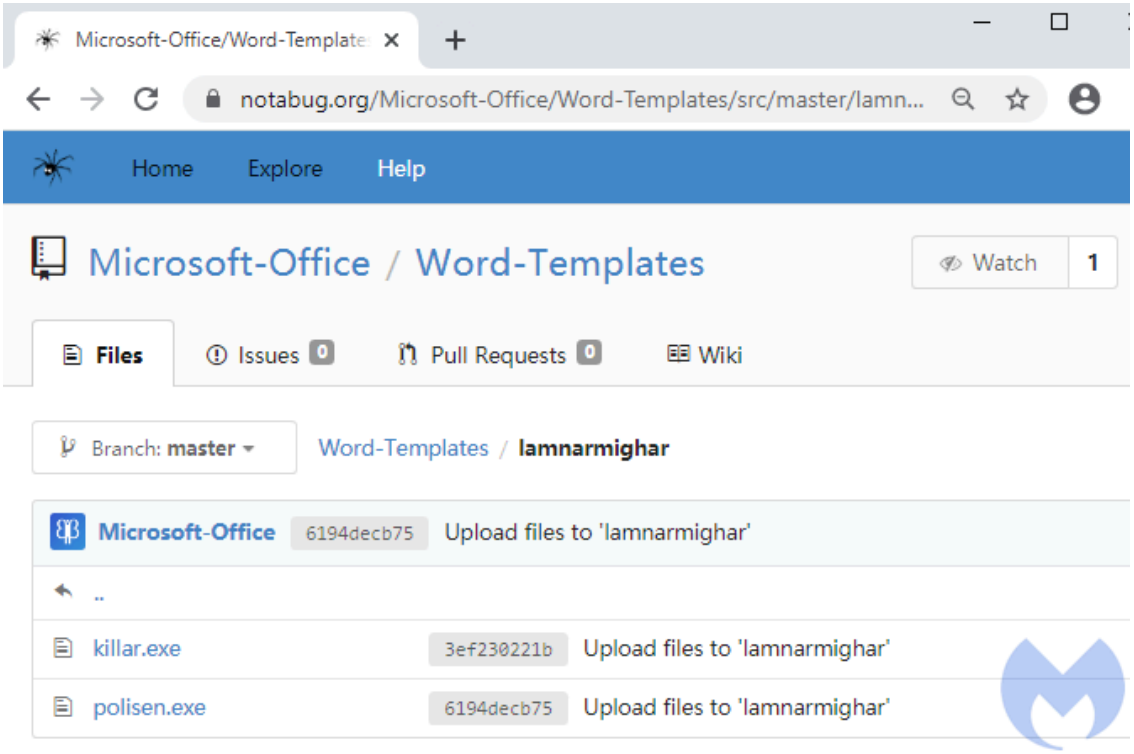
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



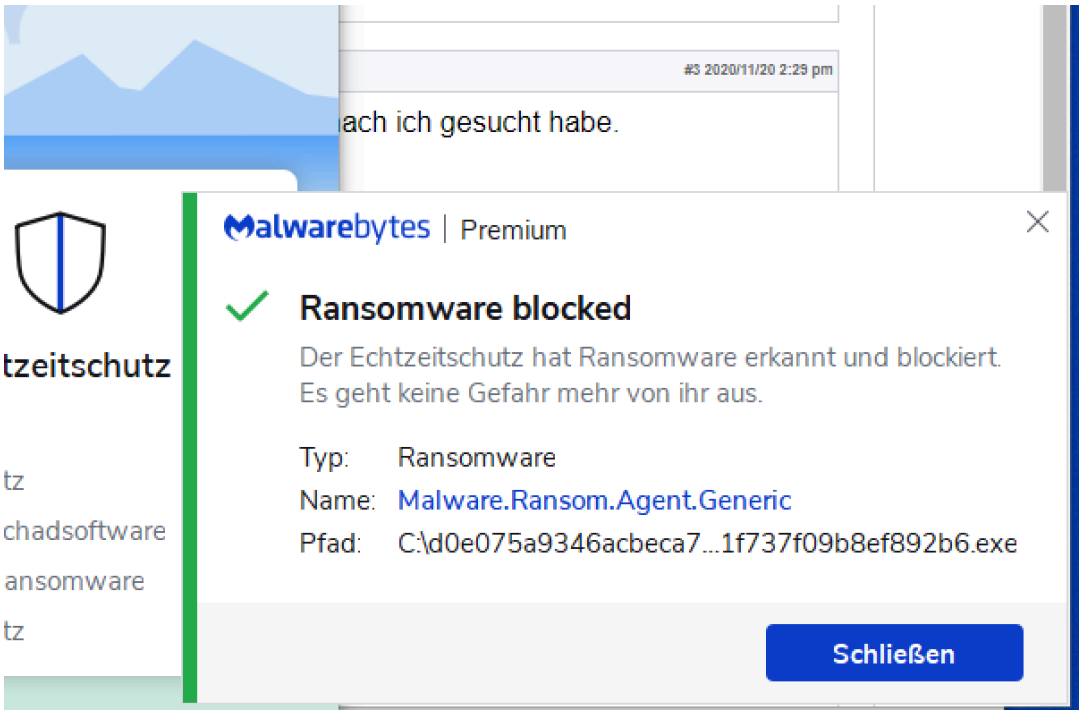
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

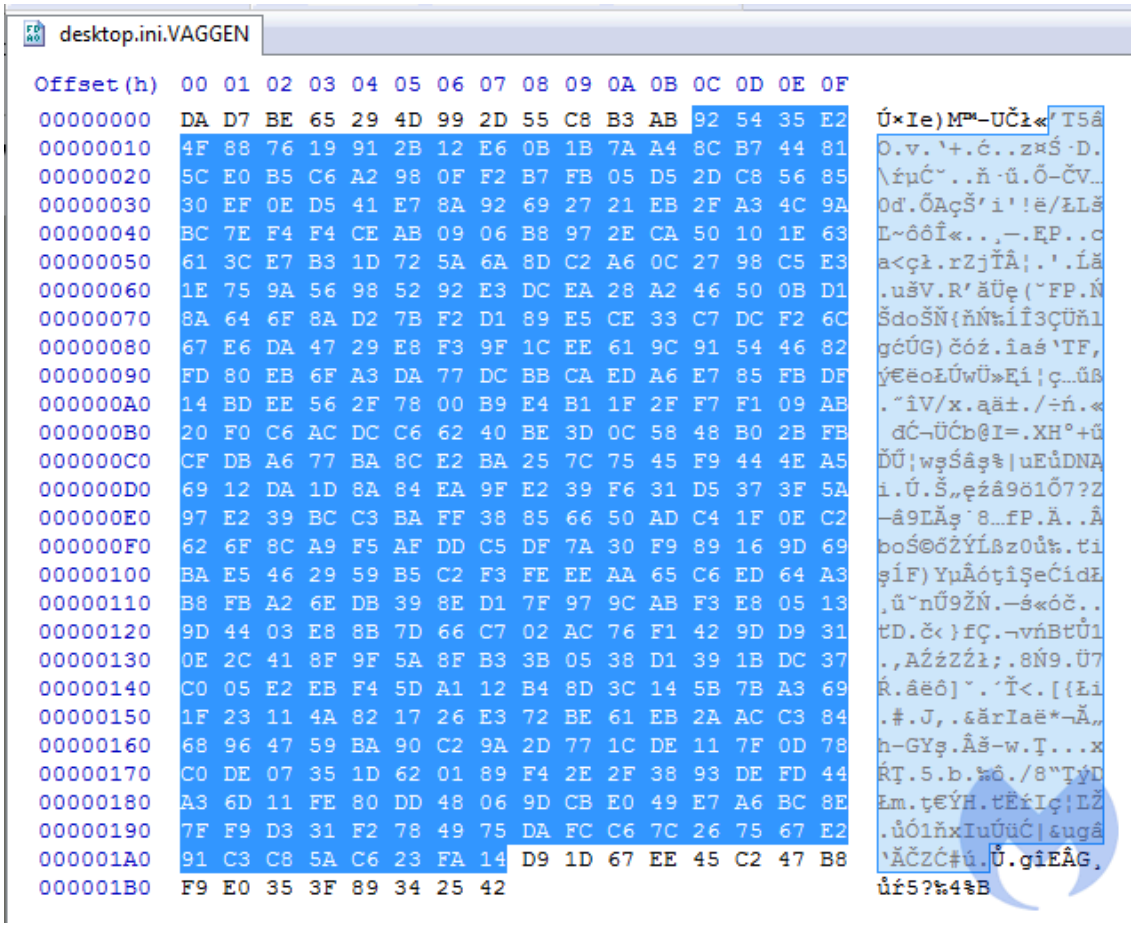
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

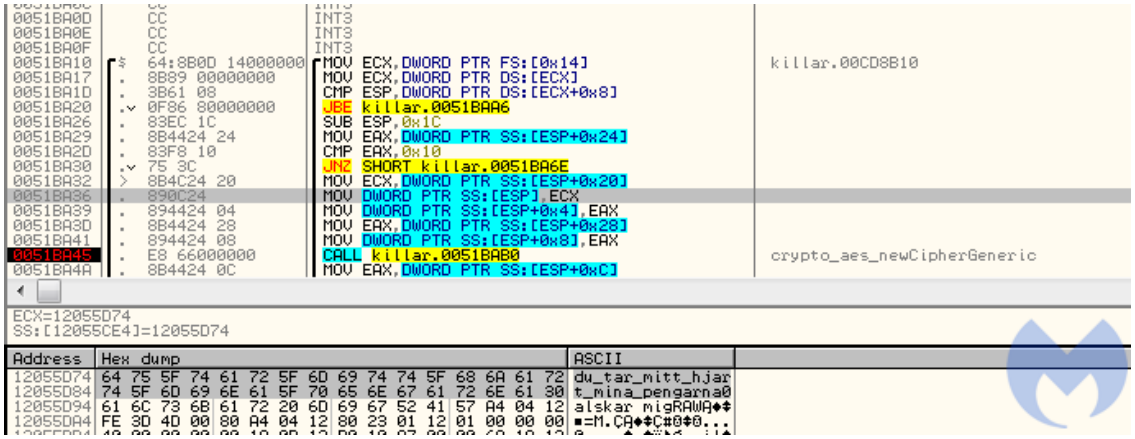


The content of the output file (with .VAGGEN extension) contains:

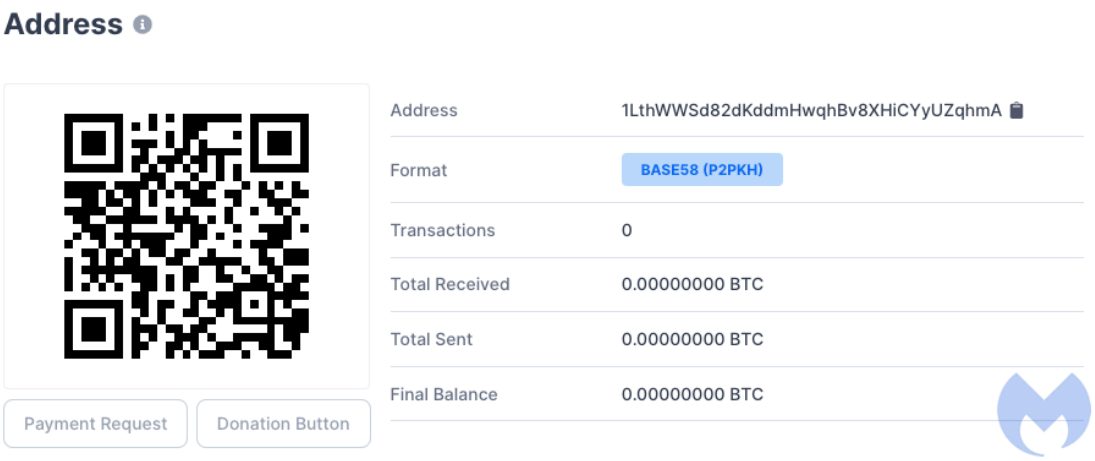
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

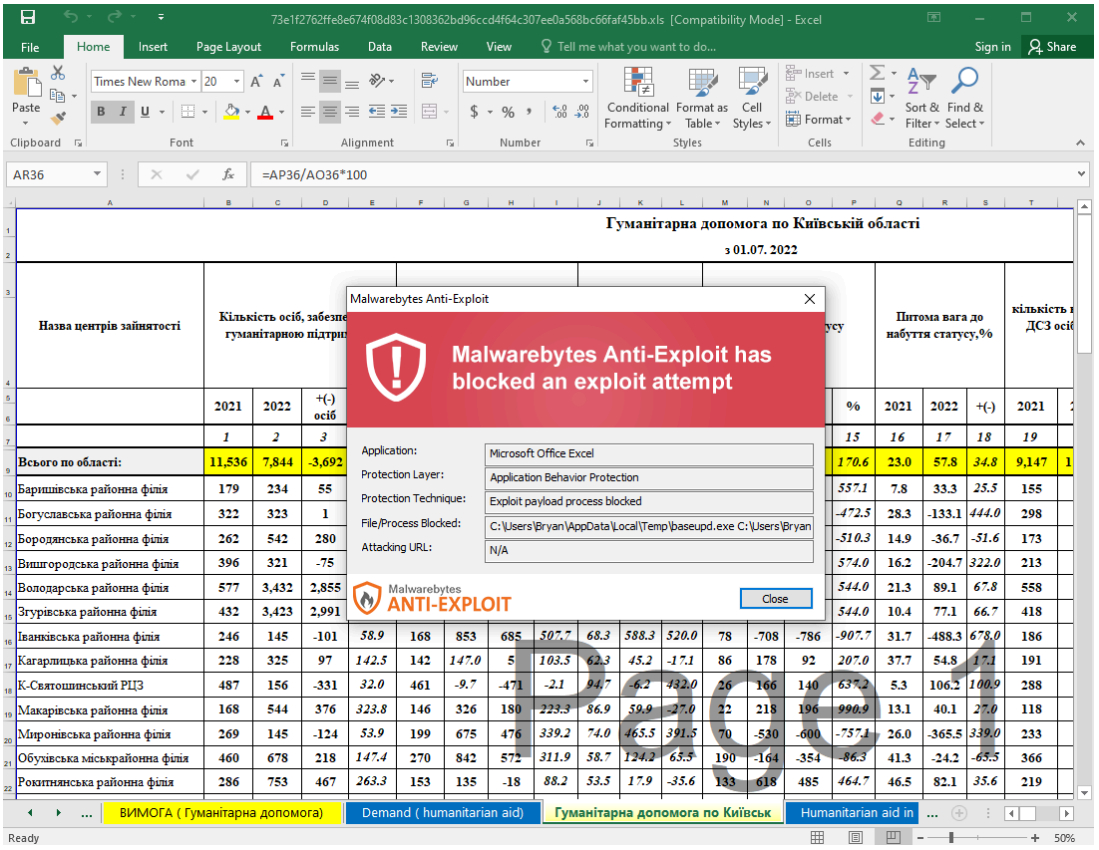
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

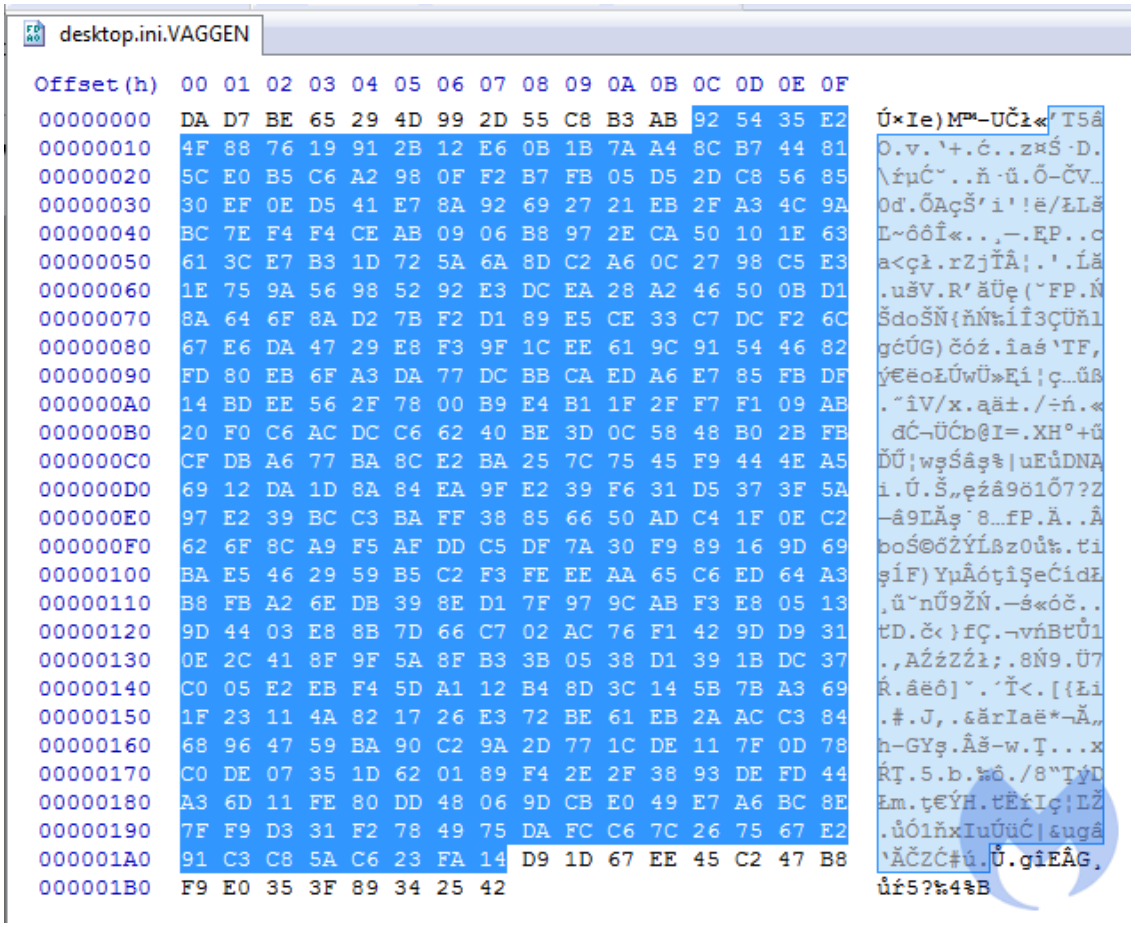
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

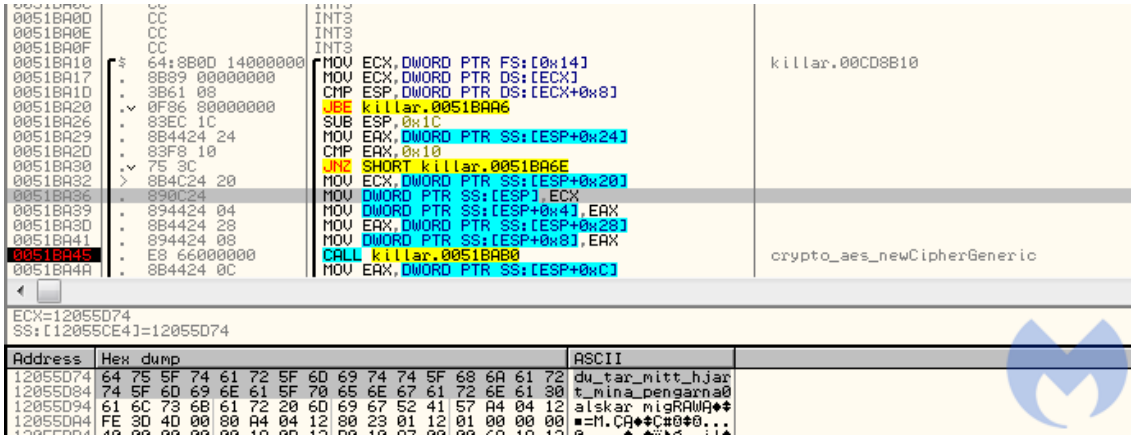


The content of the output file (with .VAGGEN extension) contains:

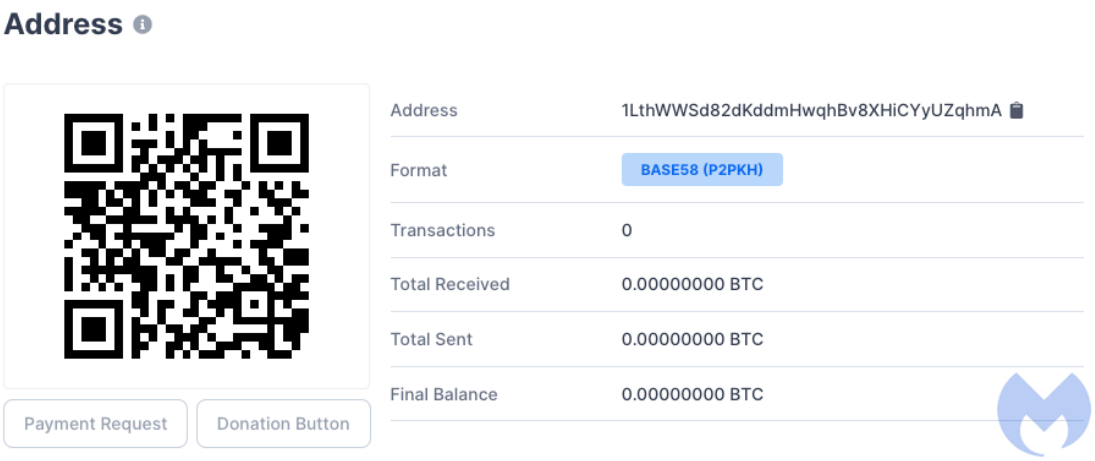
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

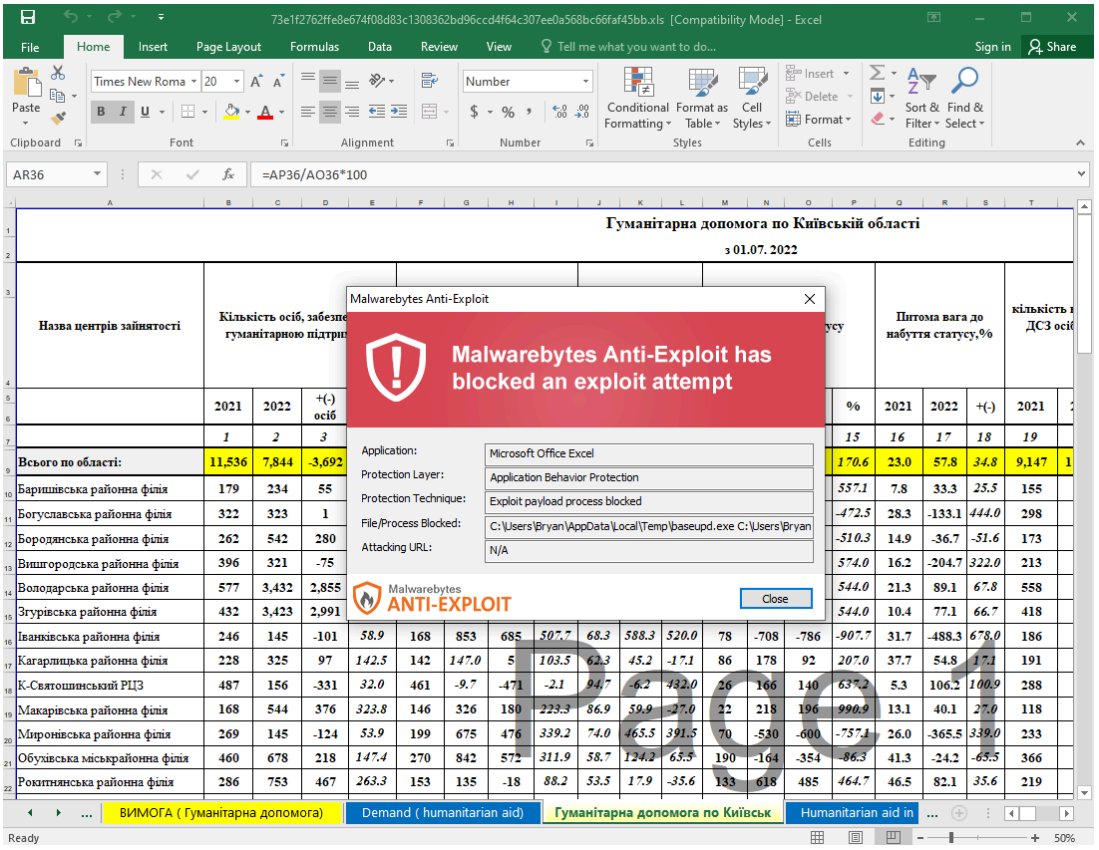
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

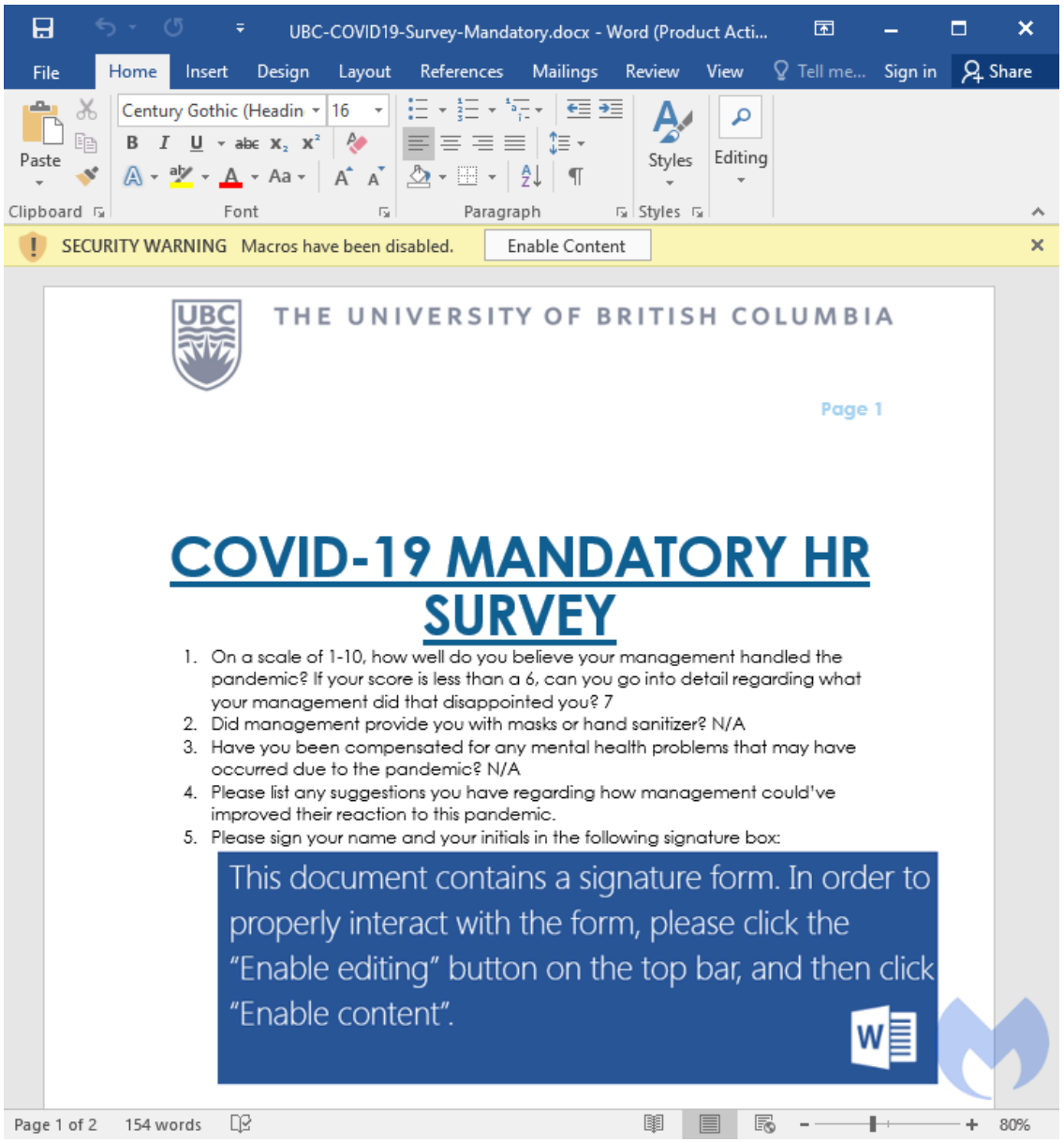
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

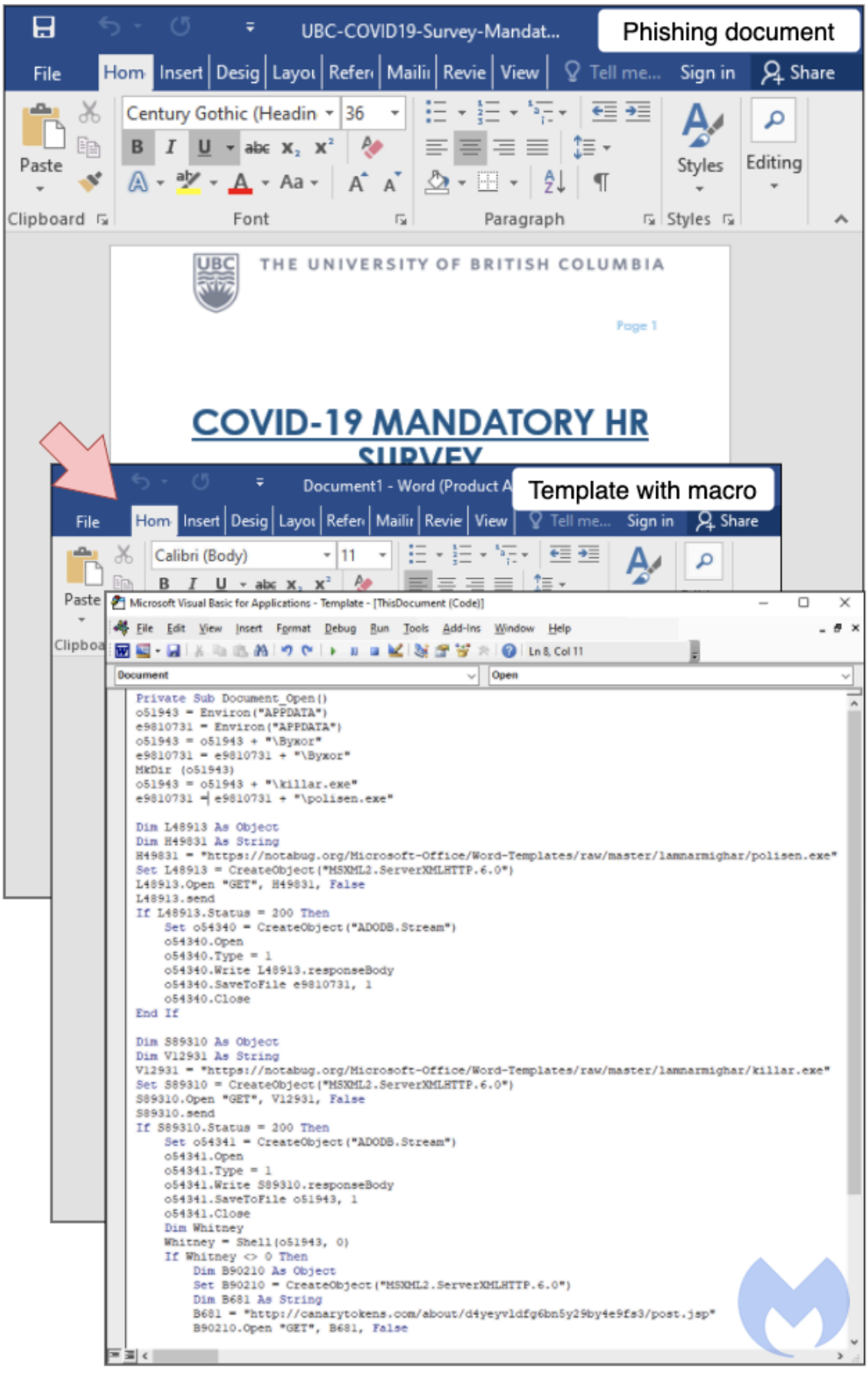
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

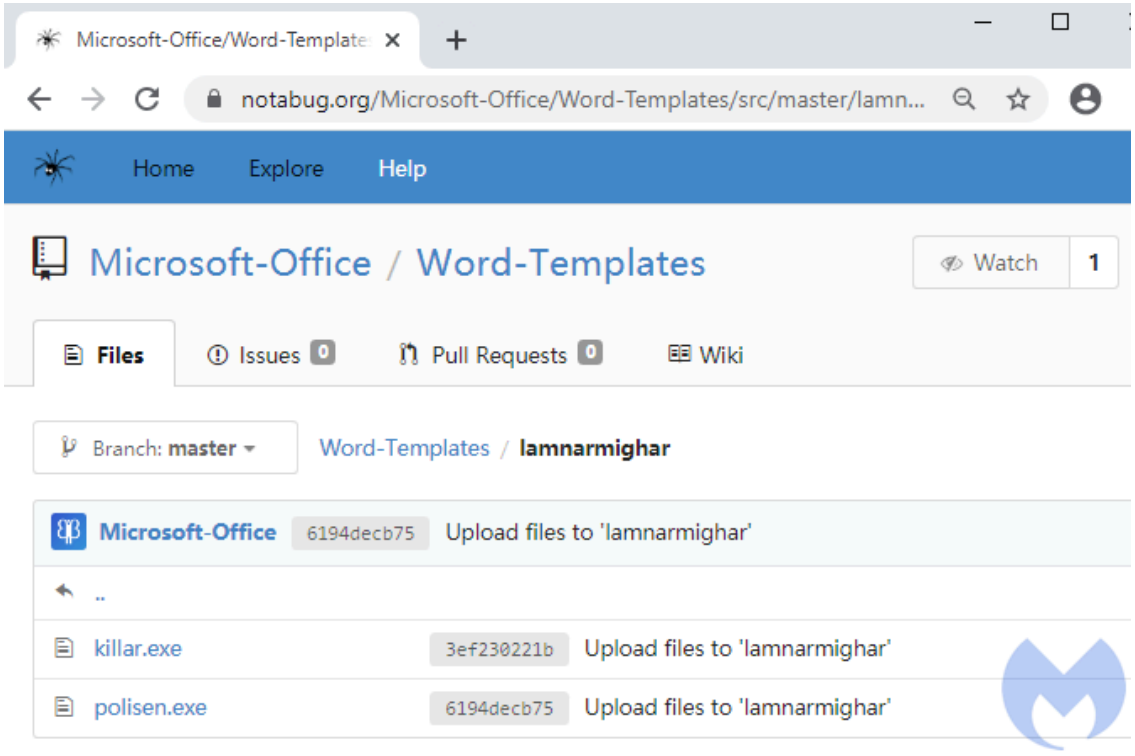
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



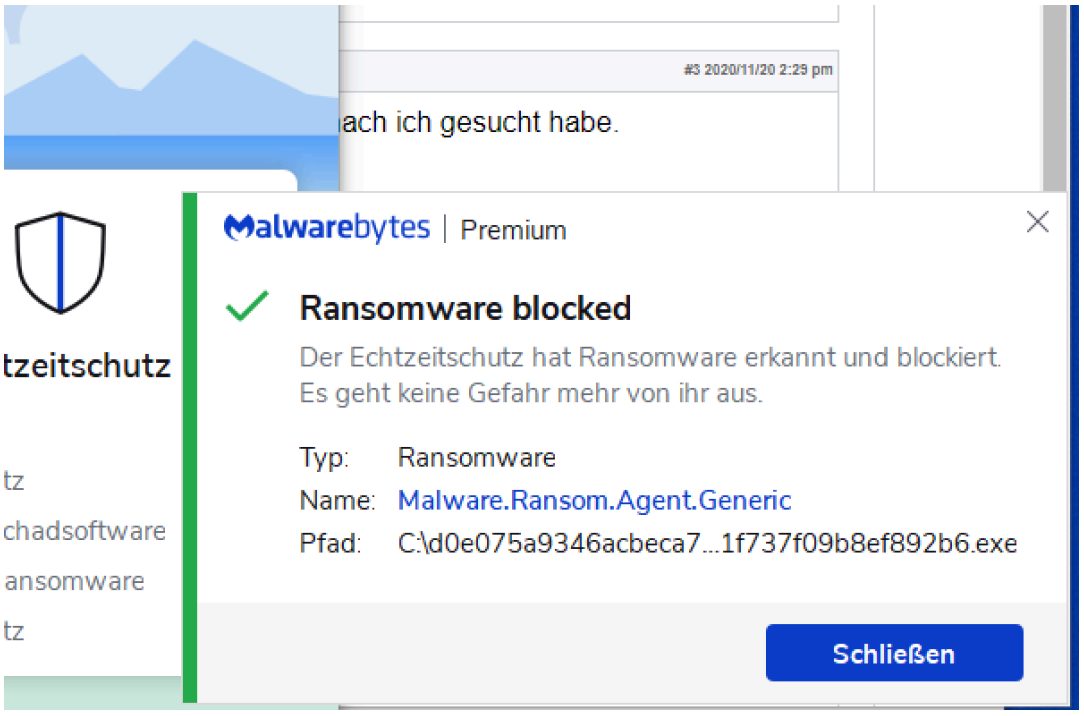
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

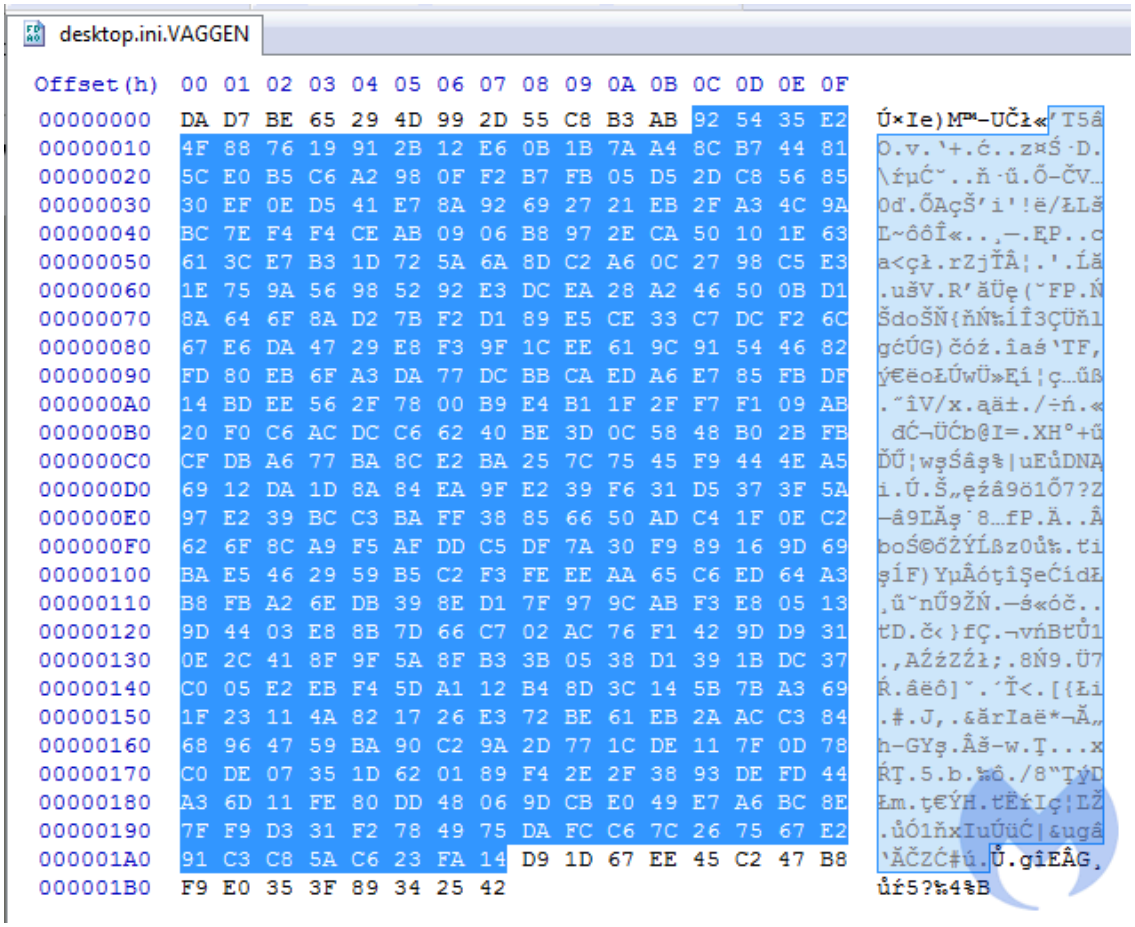
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

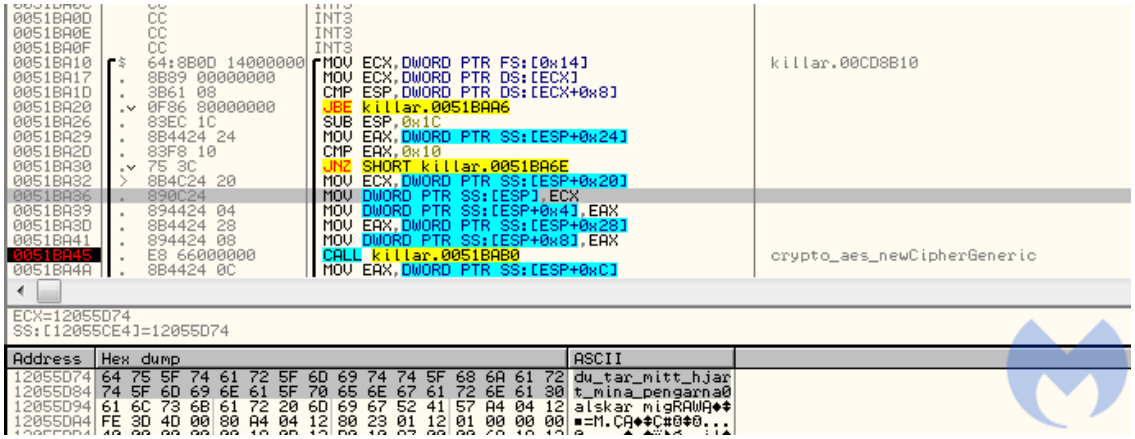


The content of the output file (with .VAGGEN extension) contains:

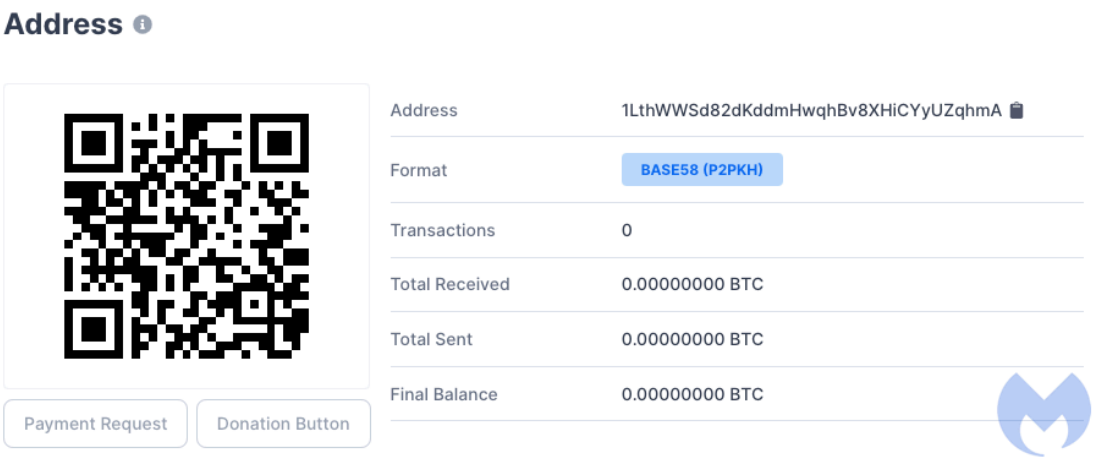
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

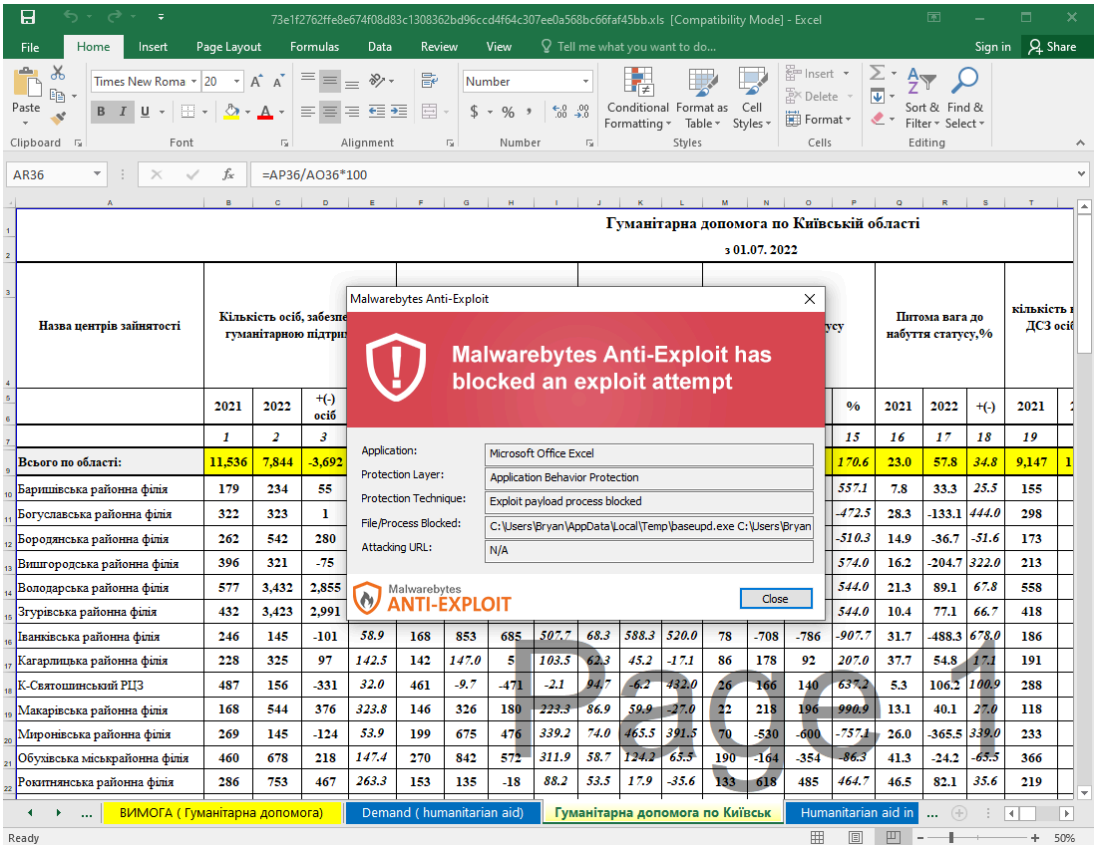
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

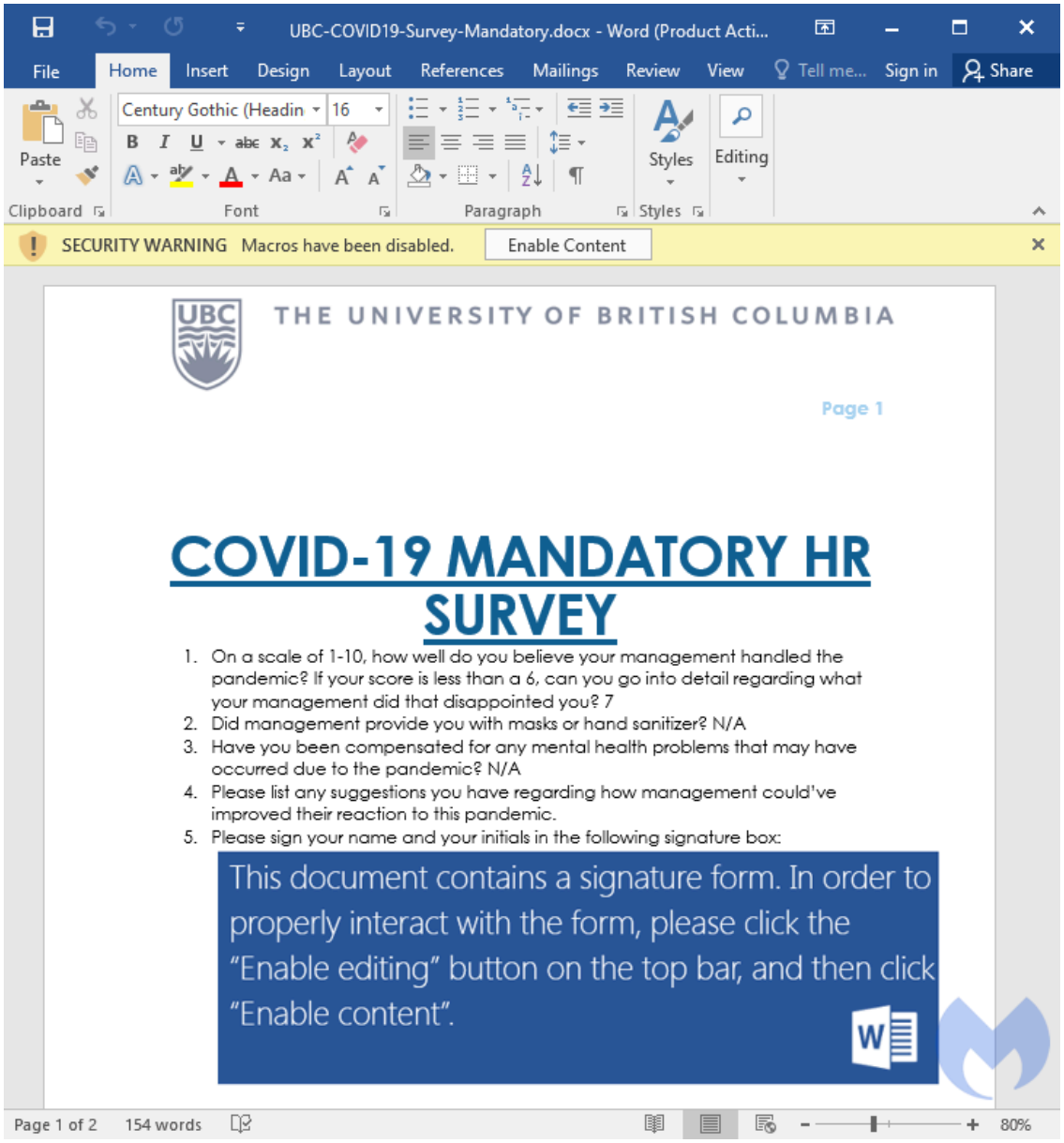
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

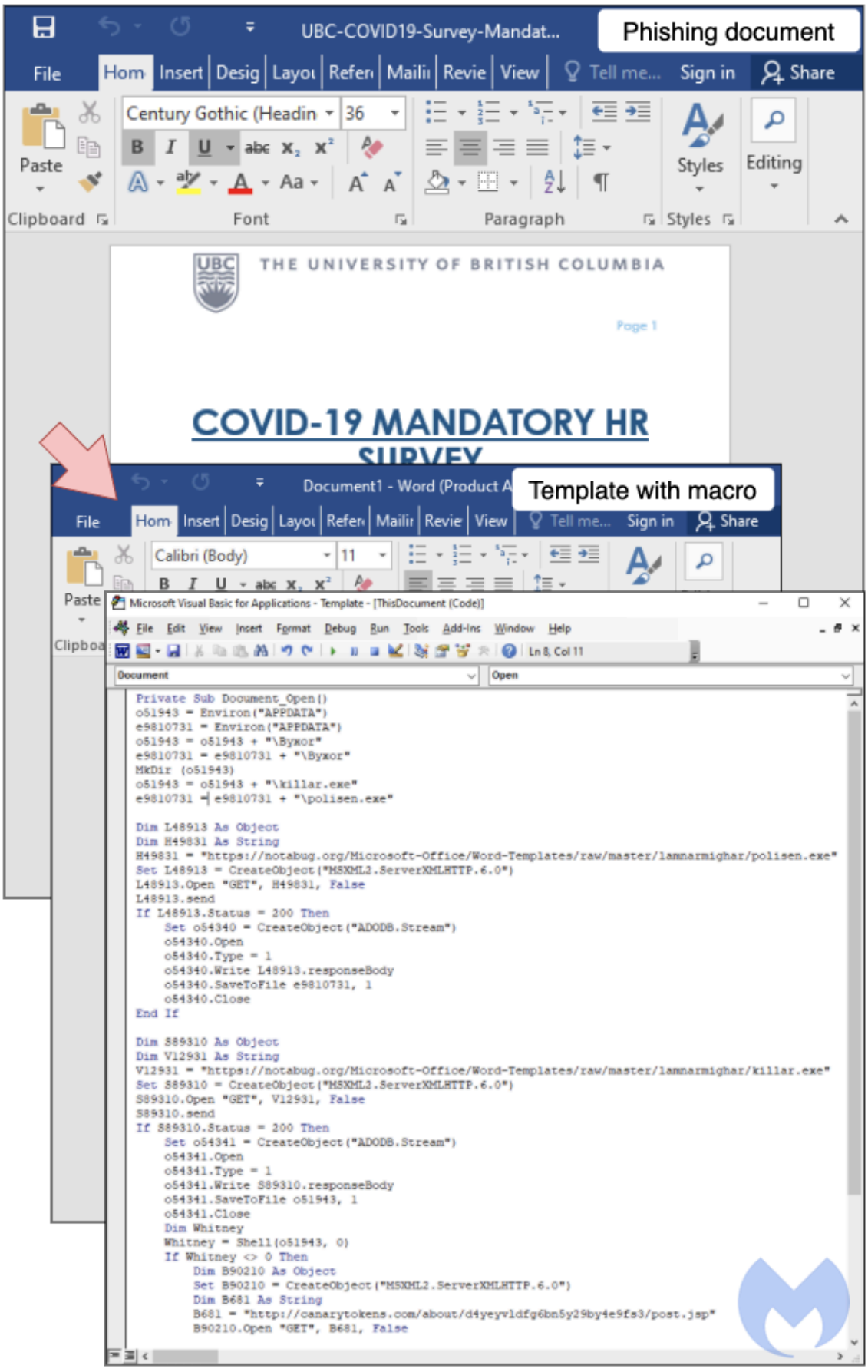
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

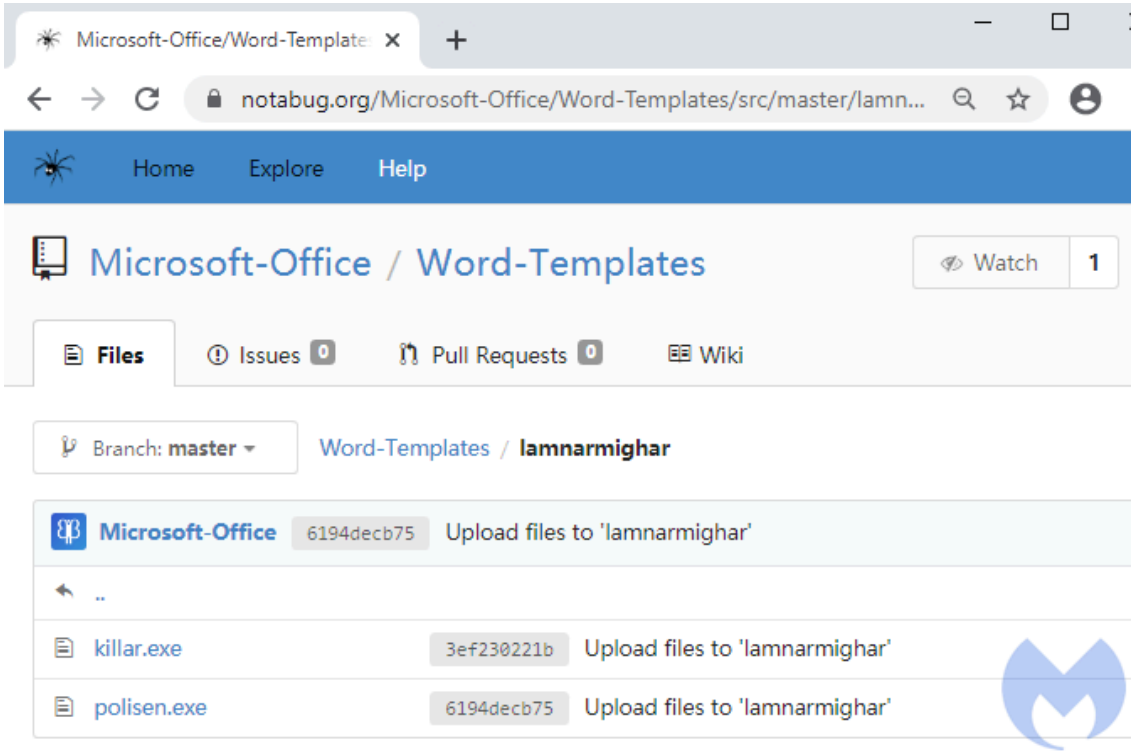
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



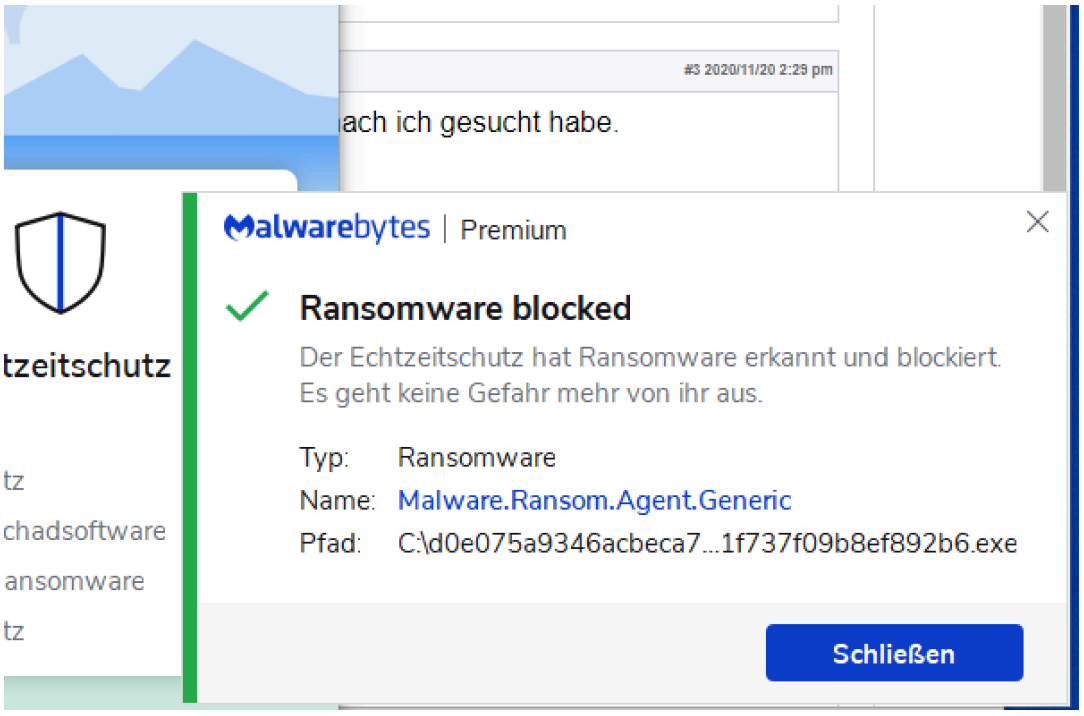
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

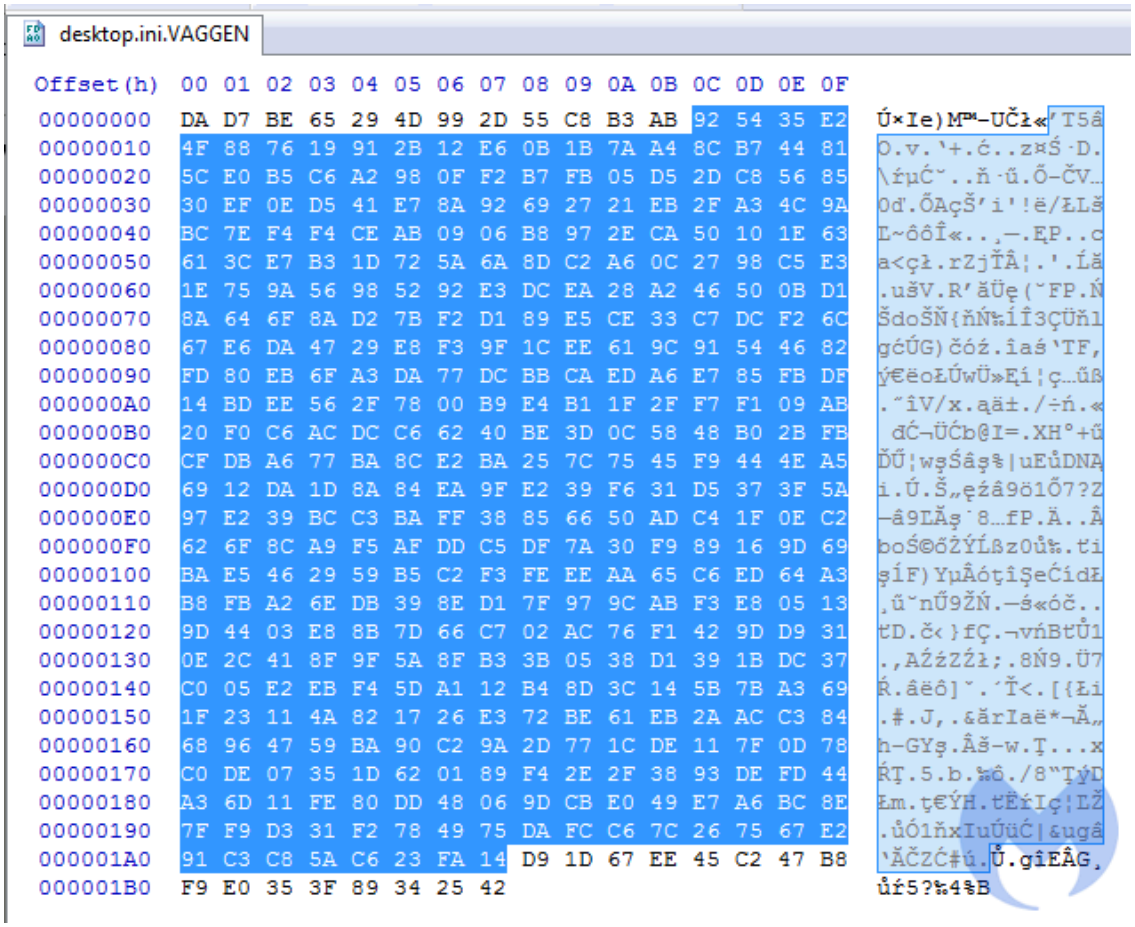
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

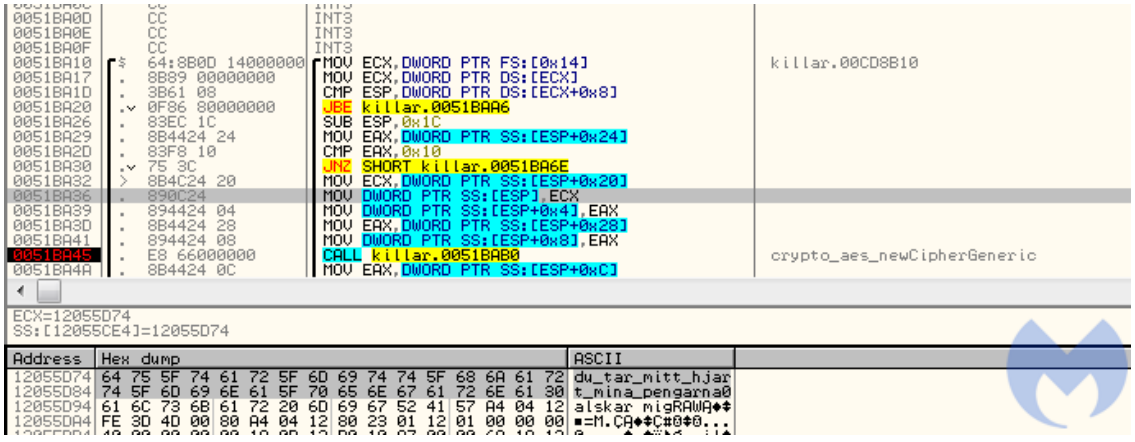


The content of the output file (with .VAGGEN extension) contains:

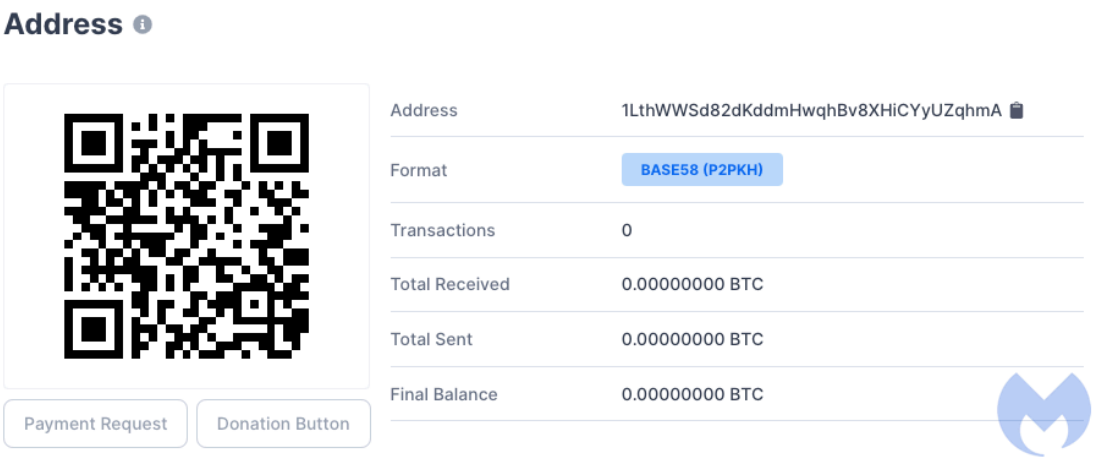
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

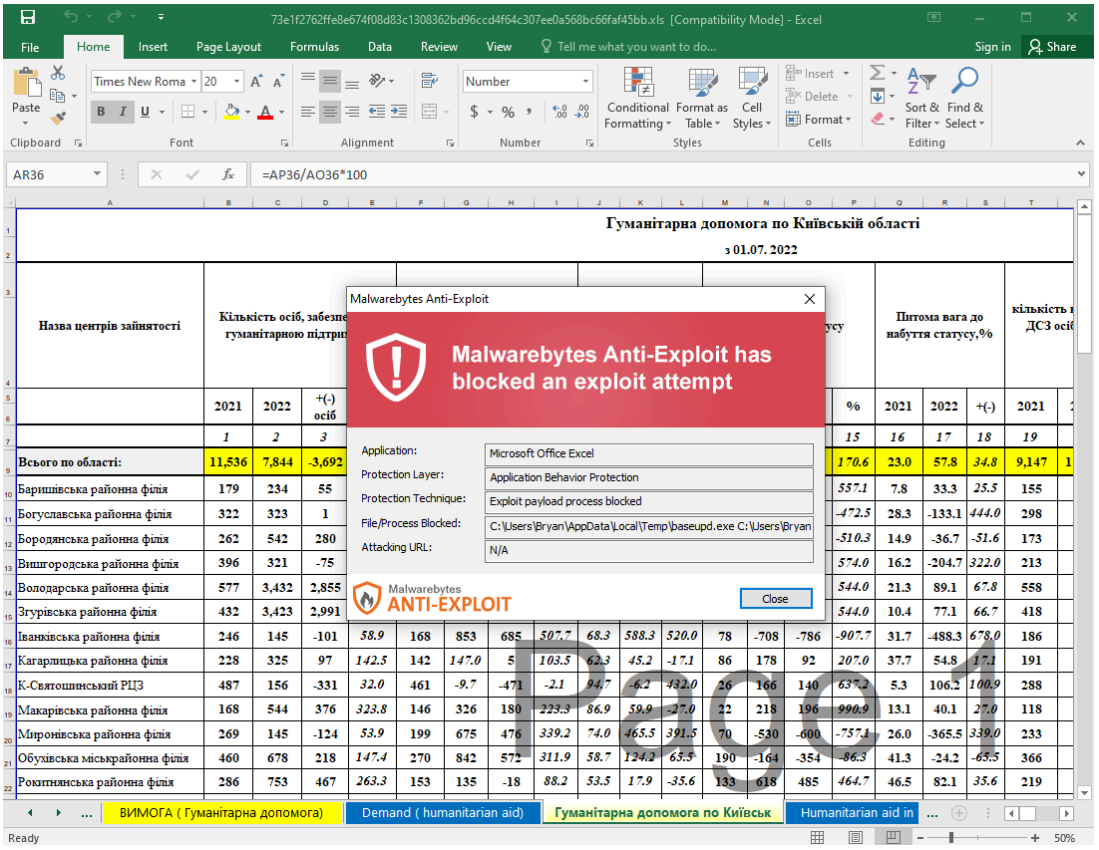
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

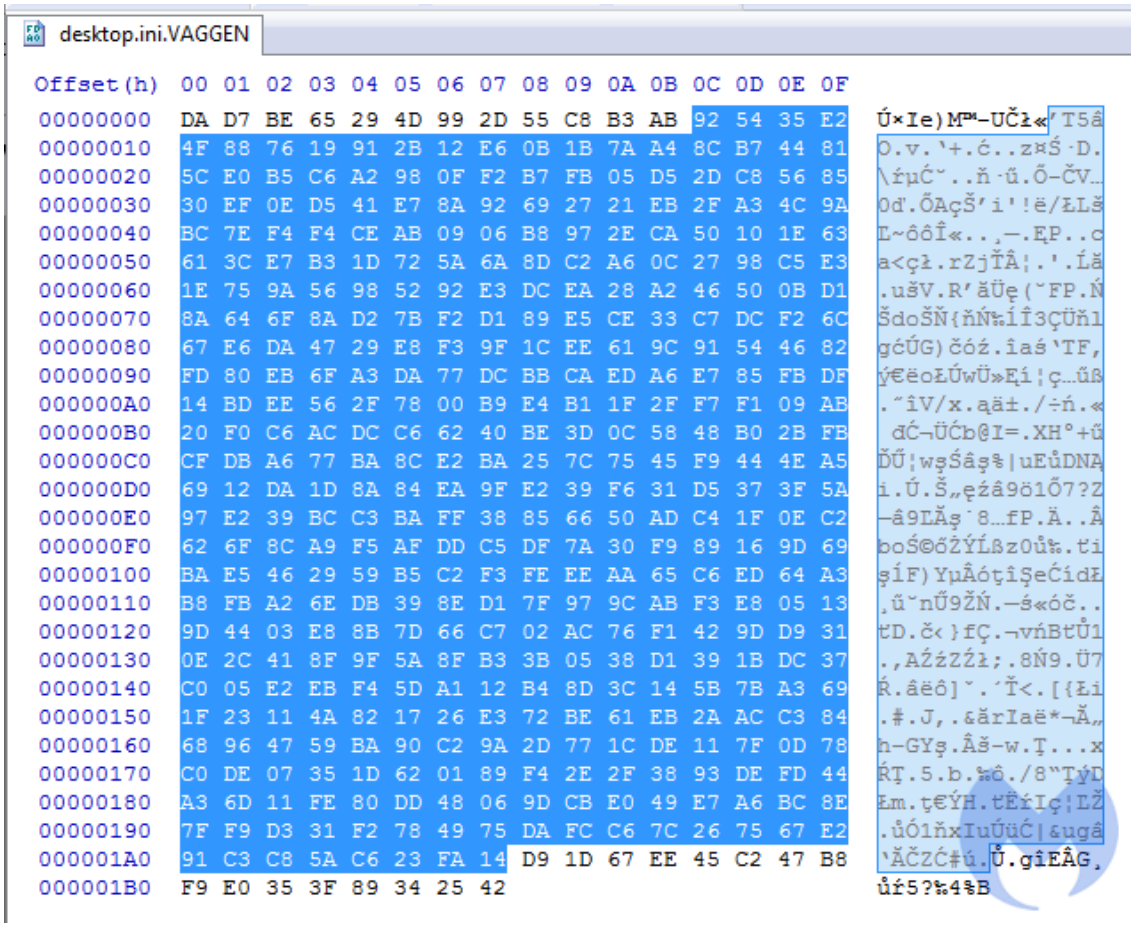
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

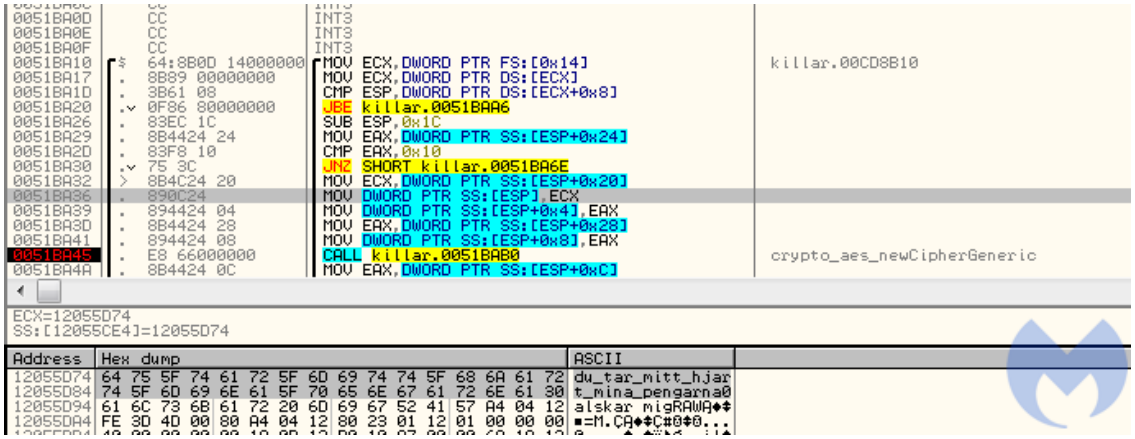


The content of the output file (with .VAGGEN extension) contains:

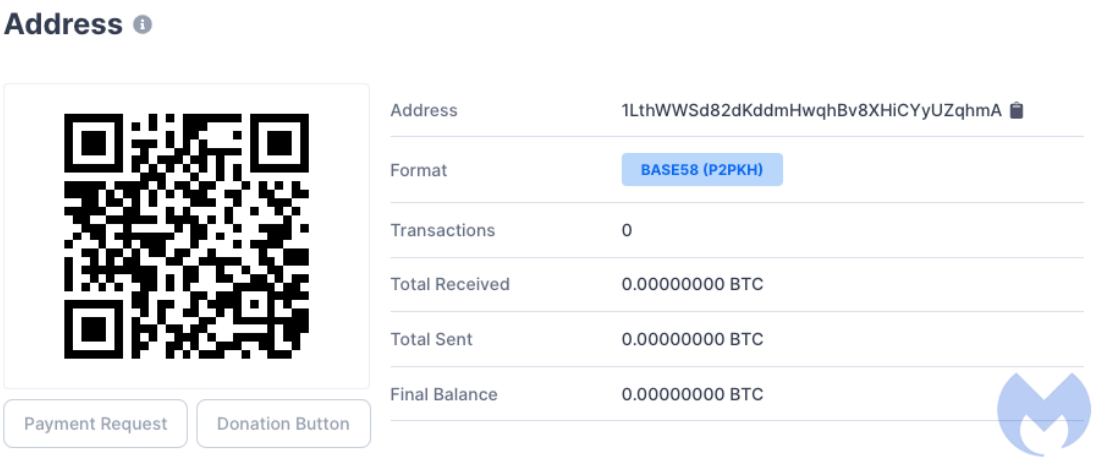
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

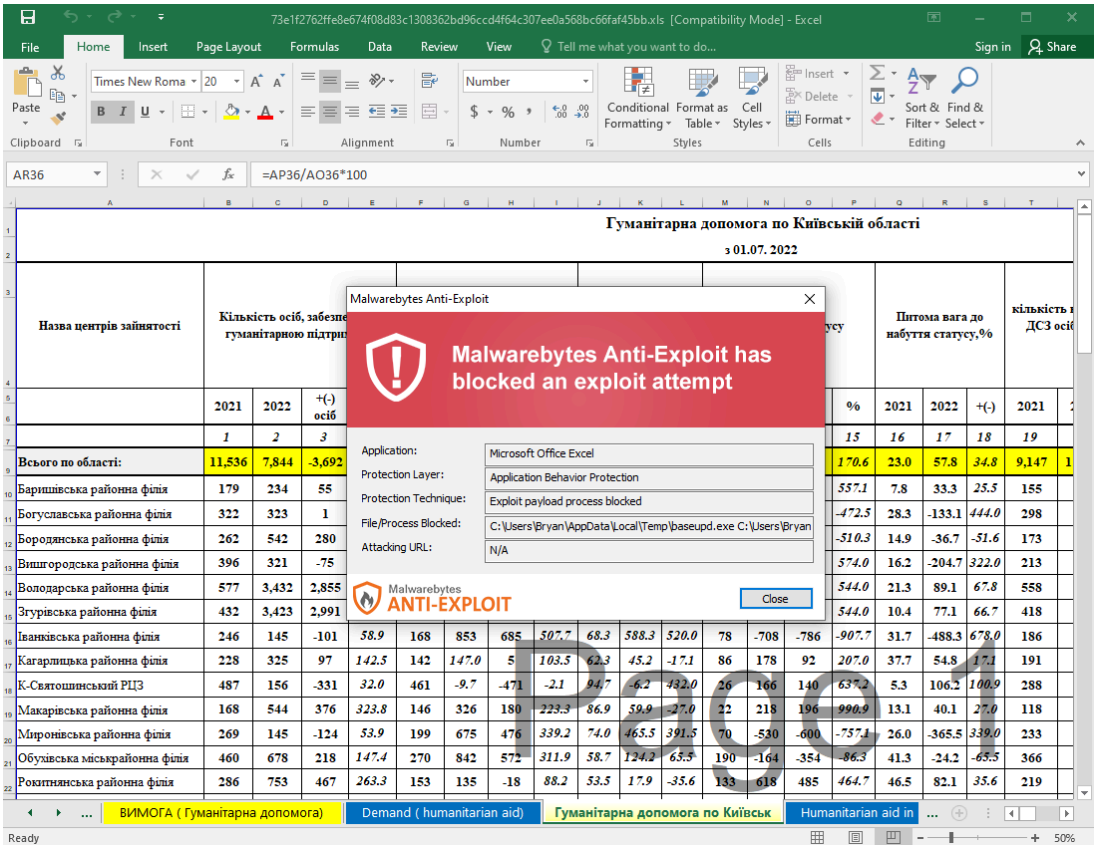
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

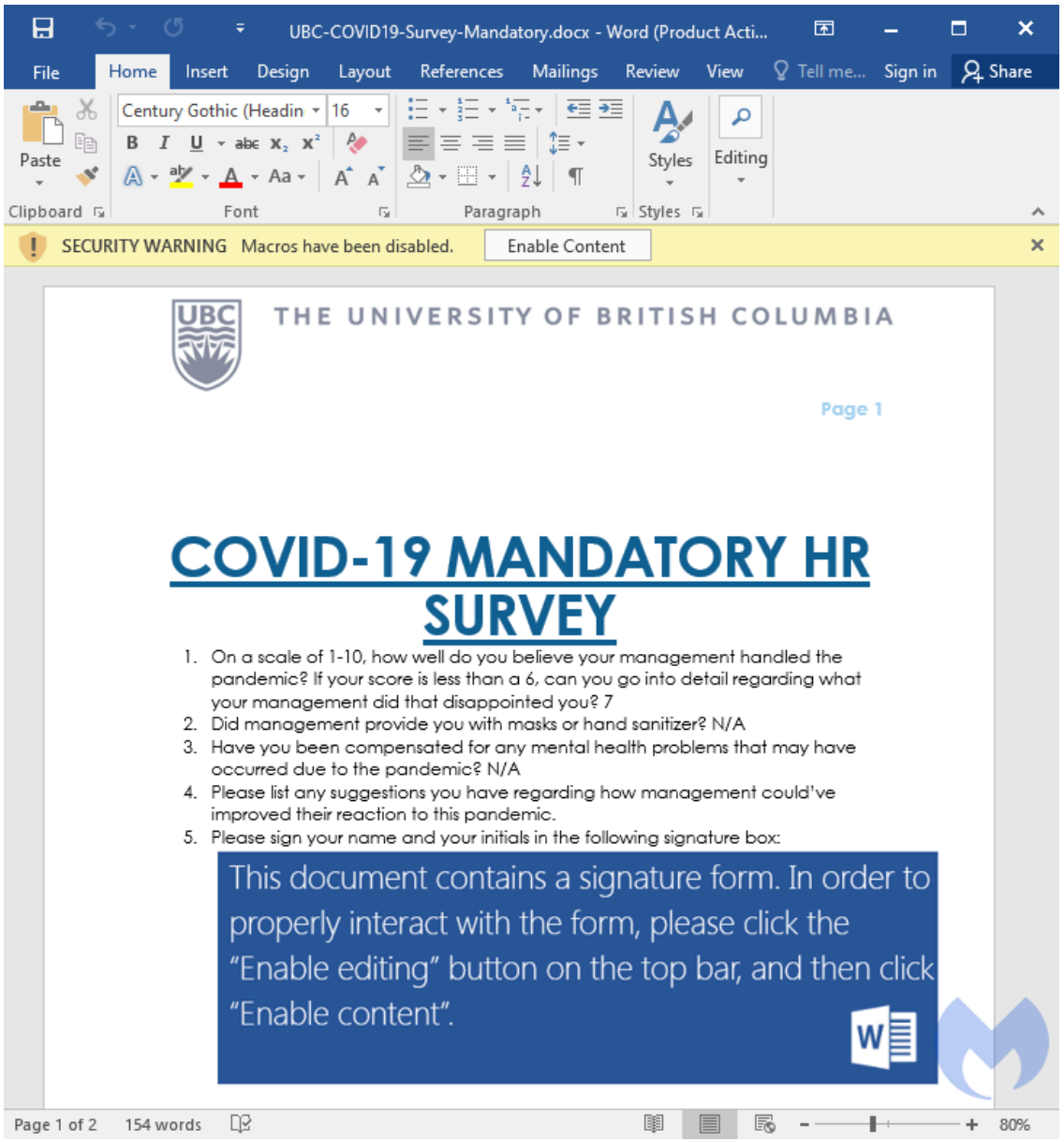
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

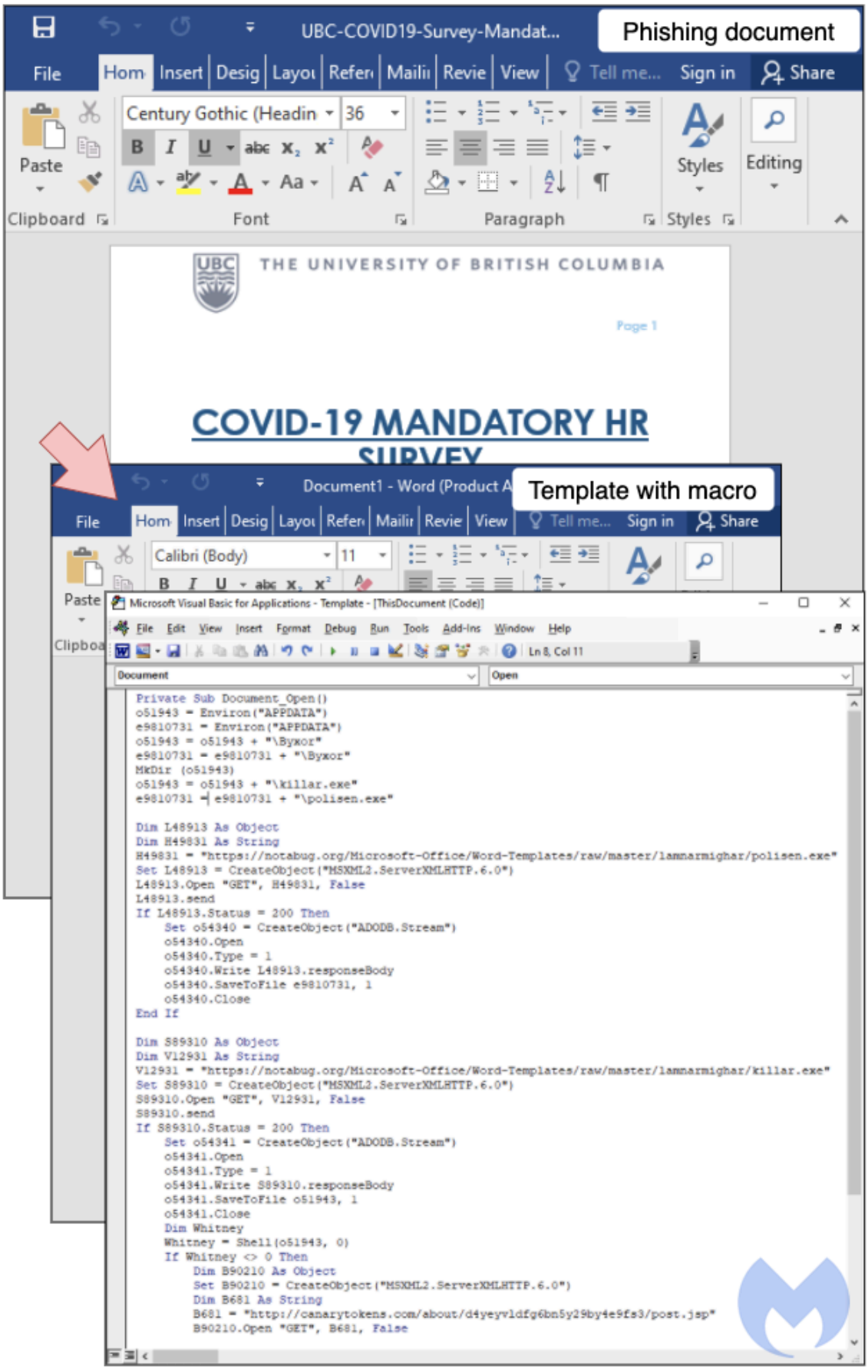
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

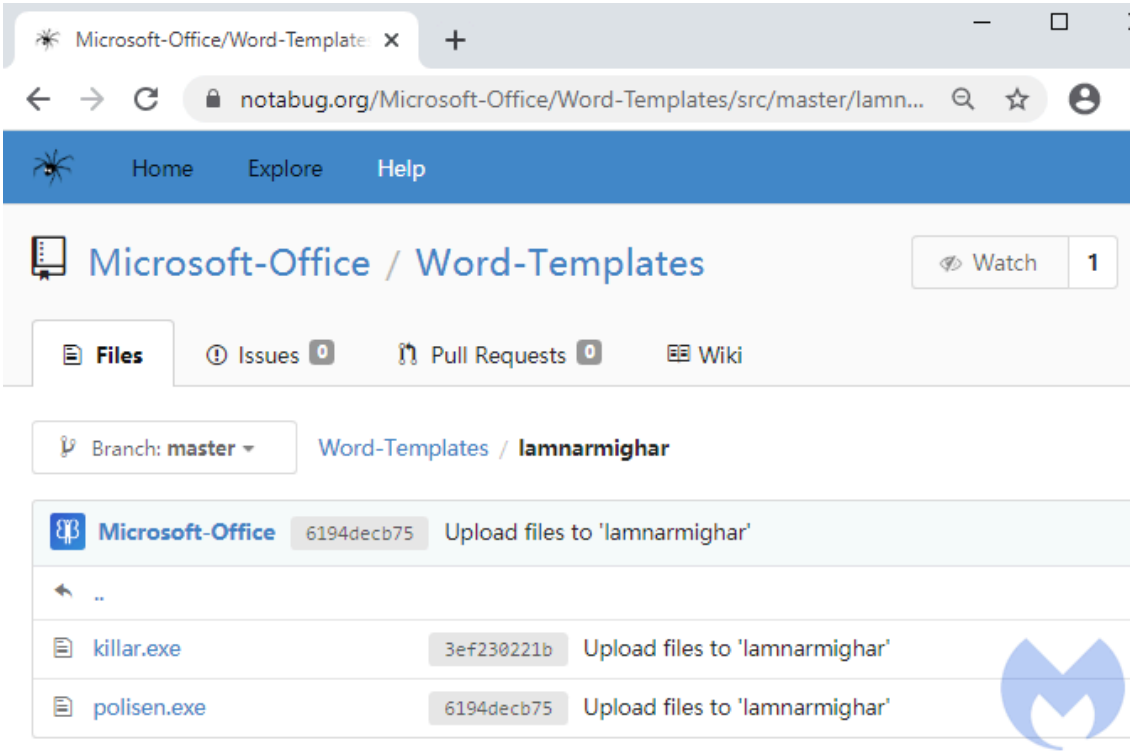
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



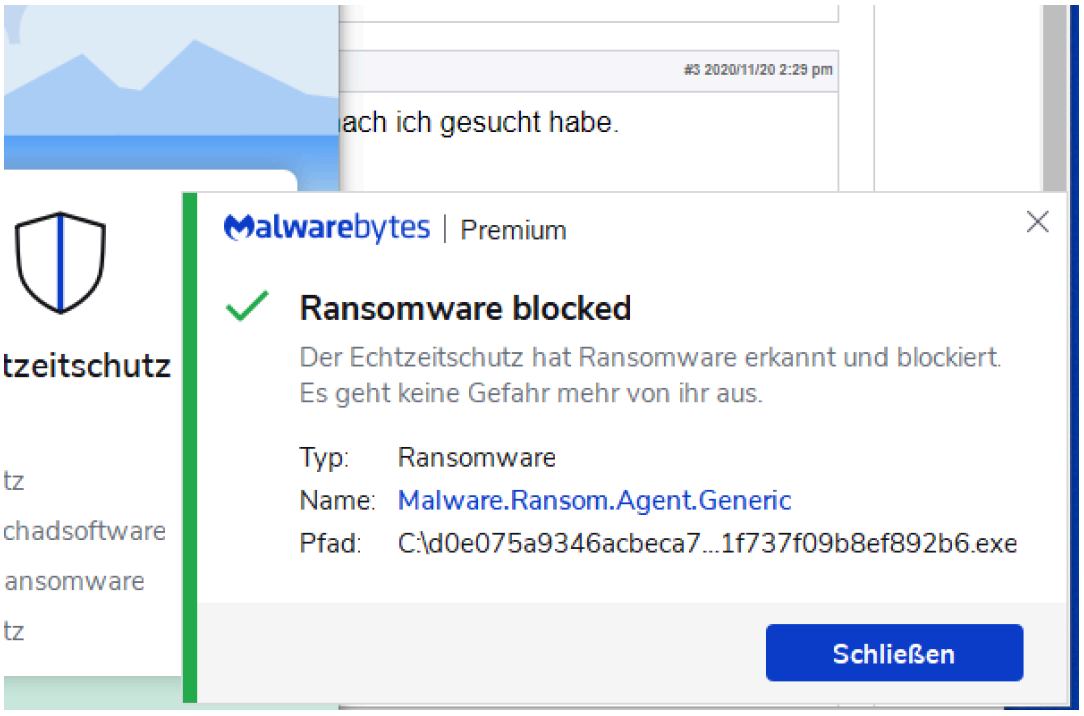
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

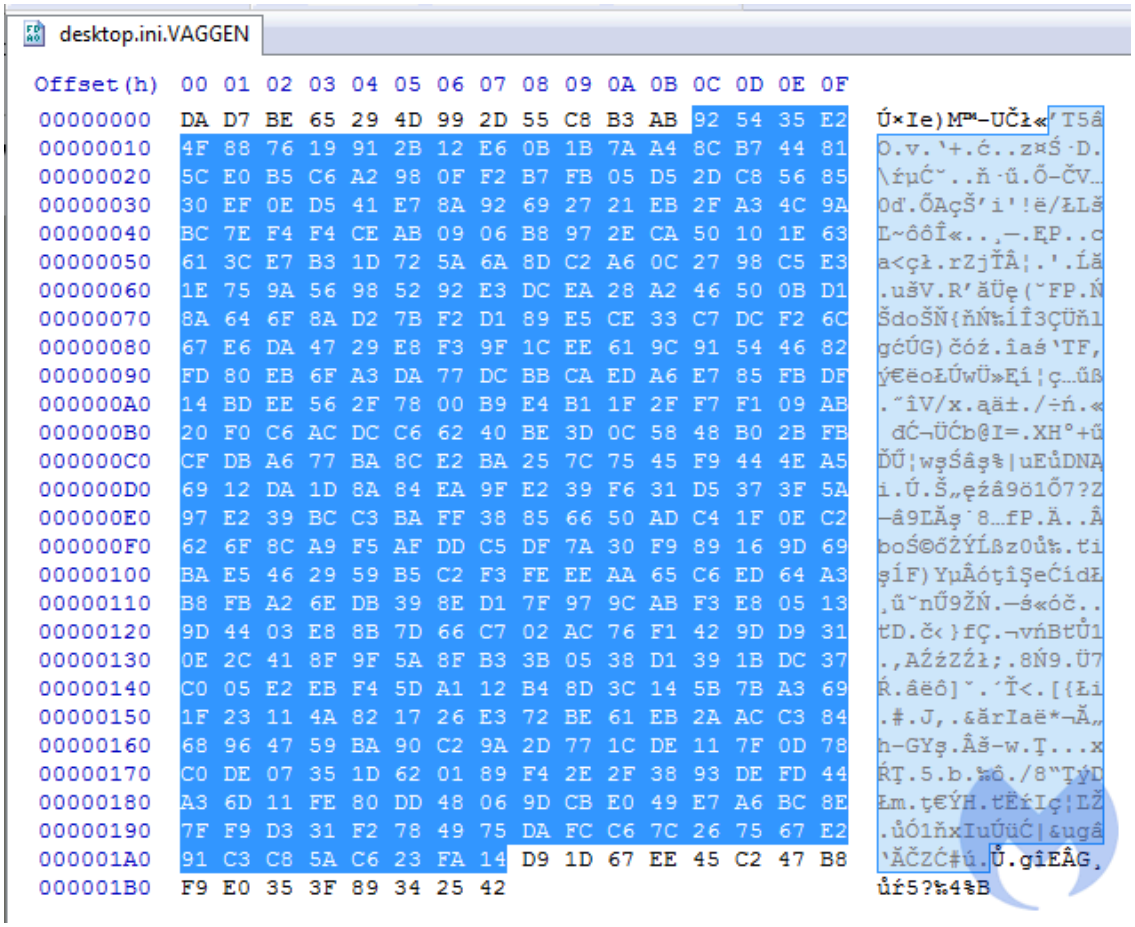
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

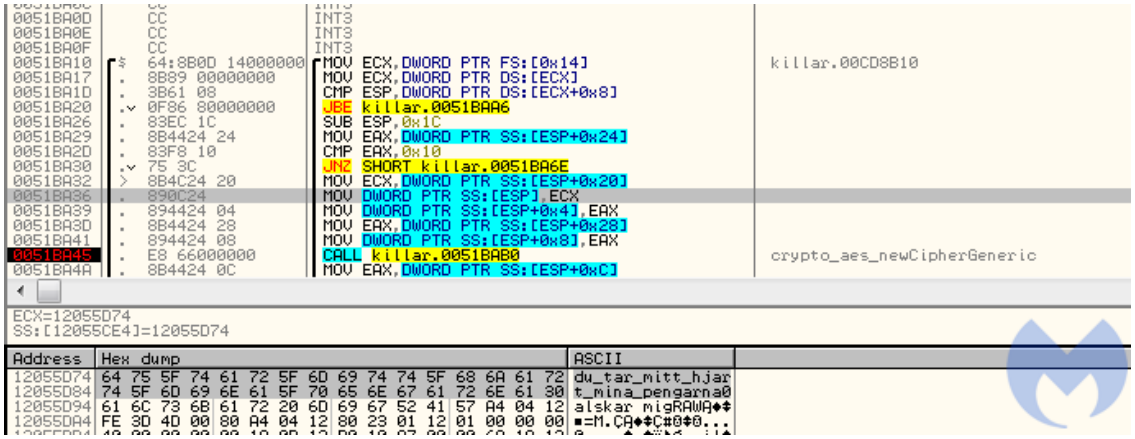


The content of the output file (with .VAGGEN extension) contains:

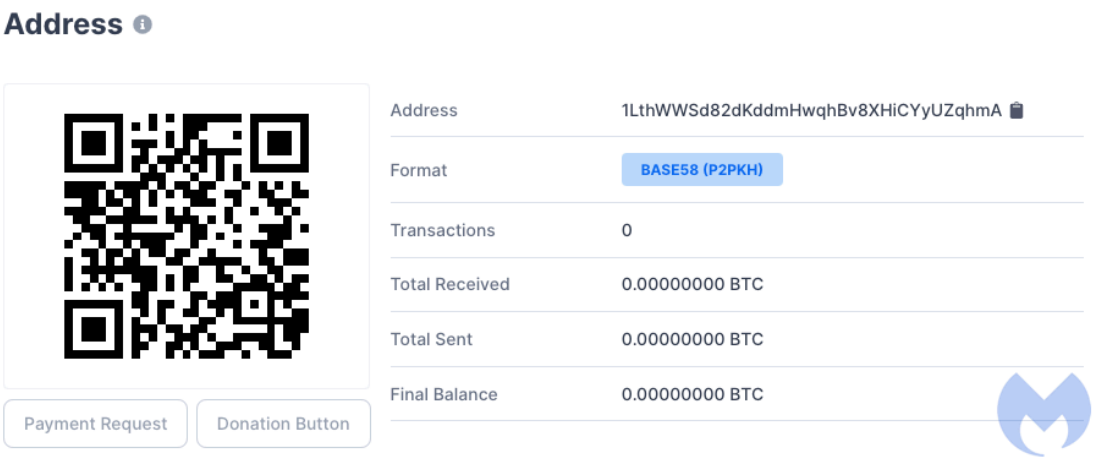
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

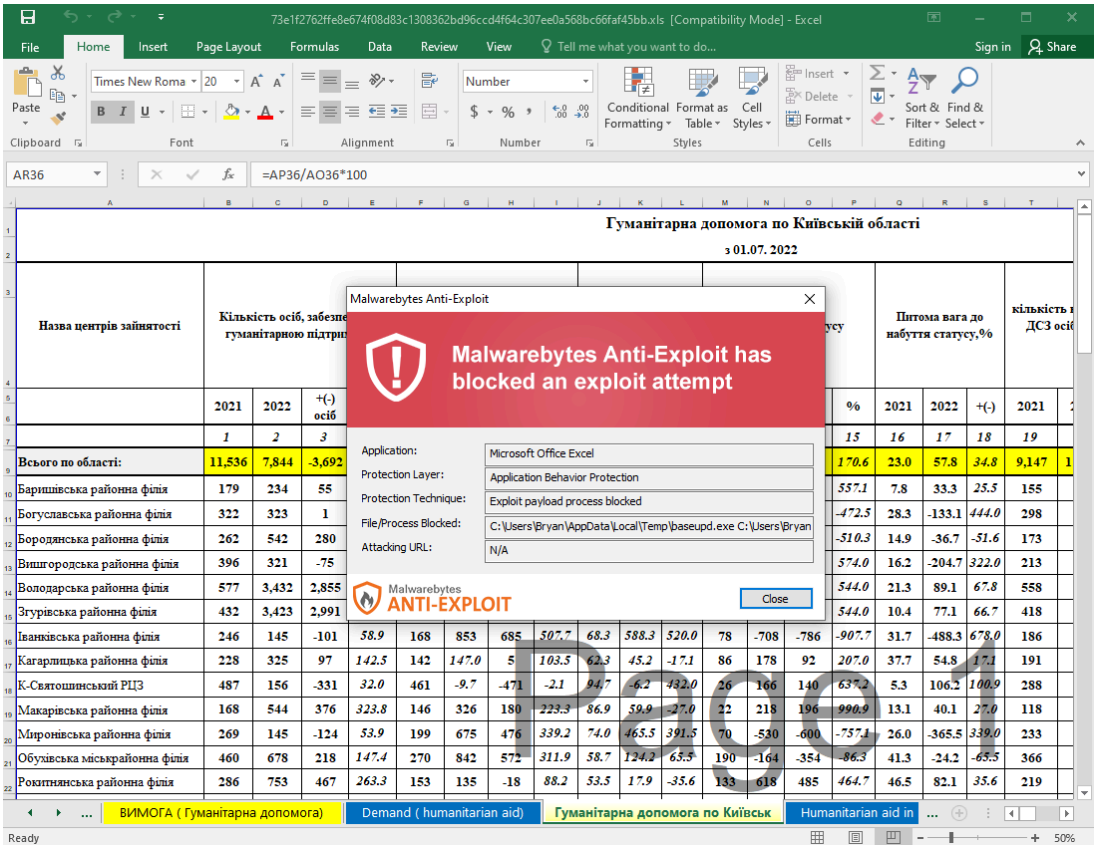
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

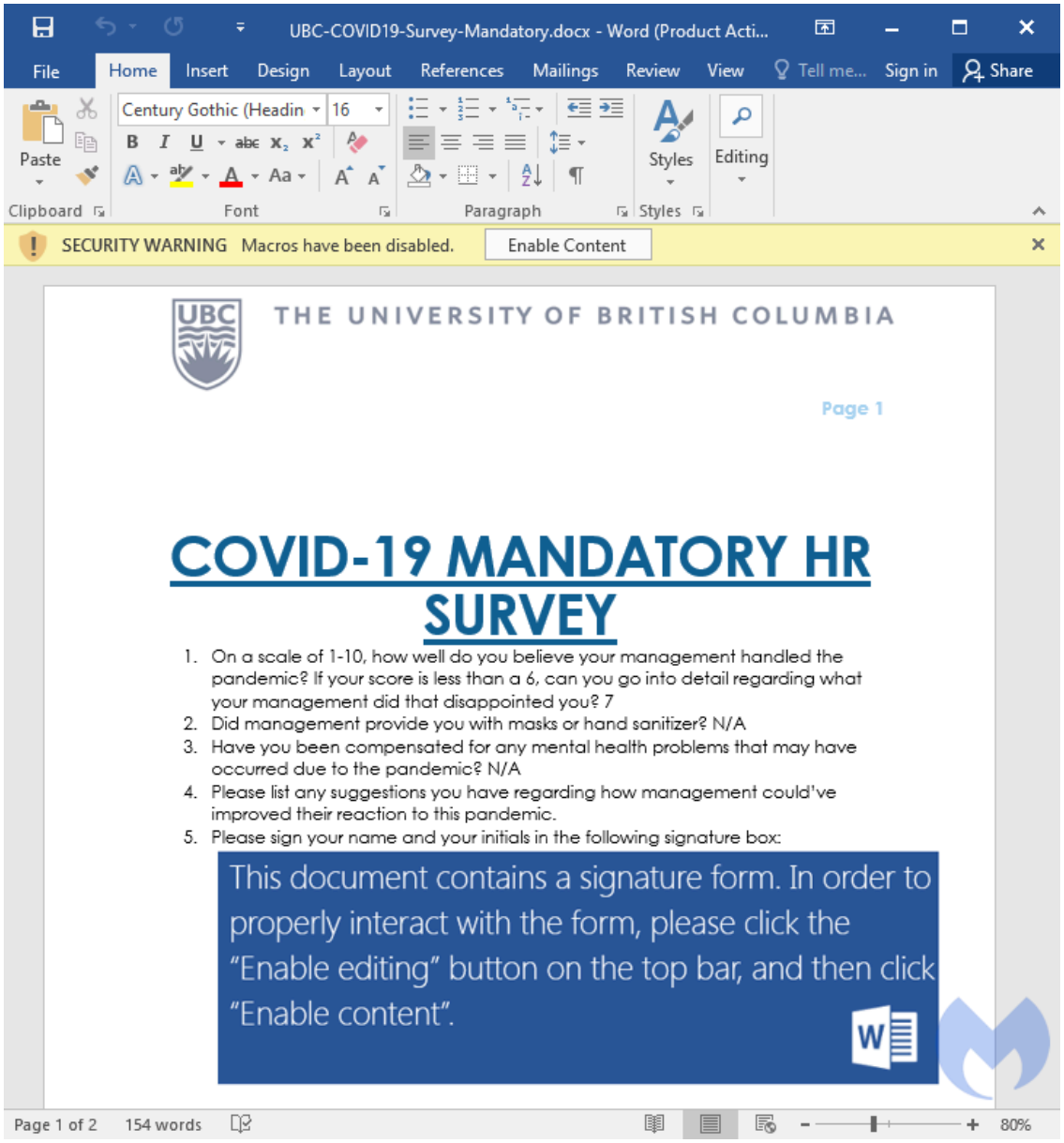
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

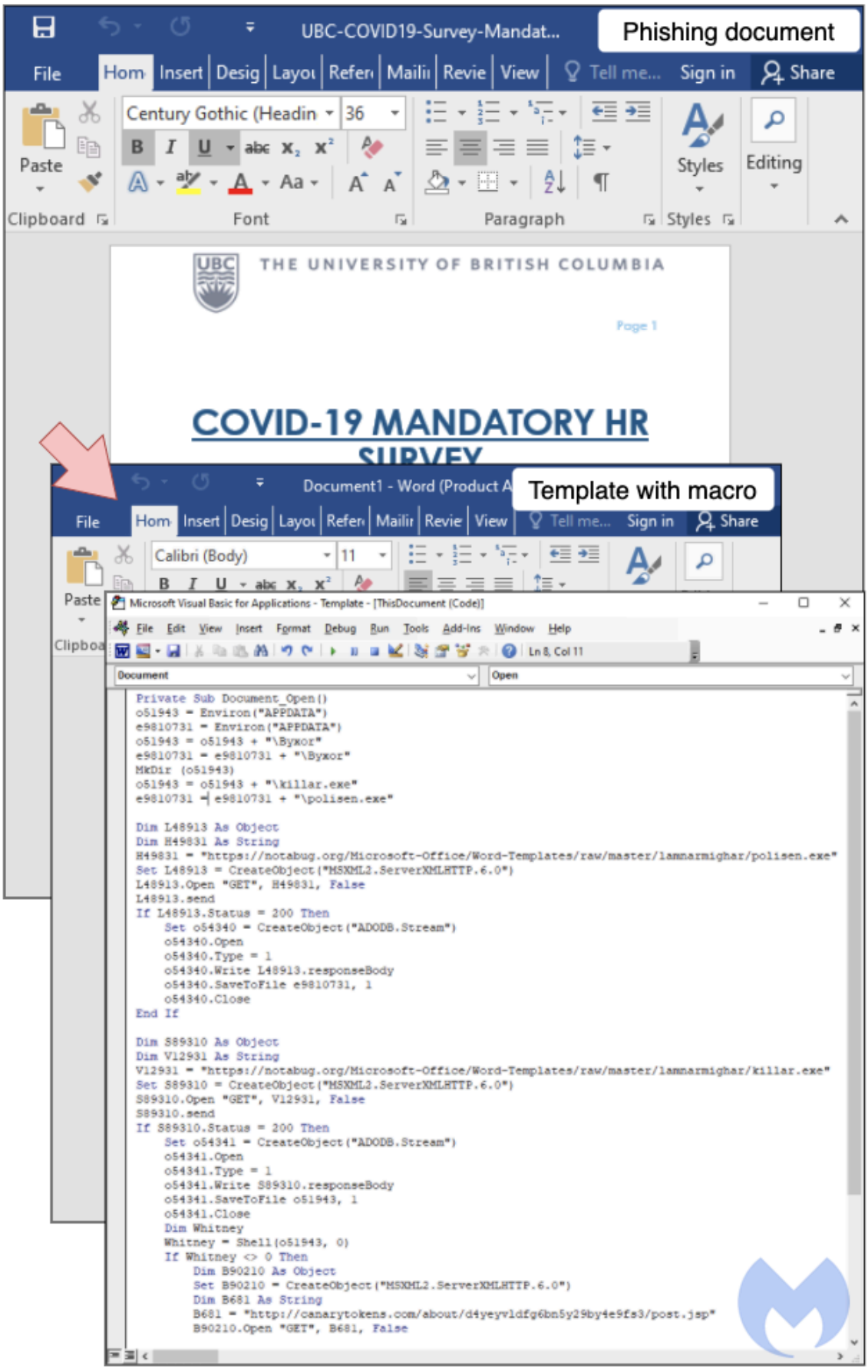
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

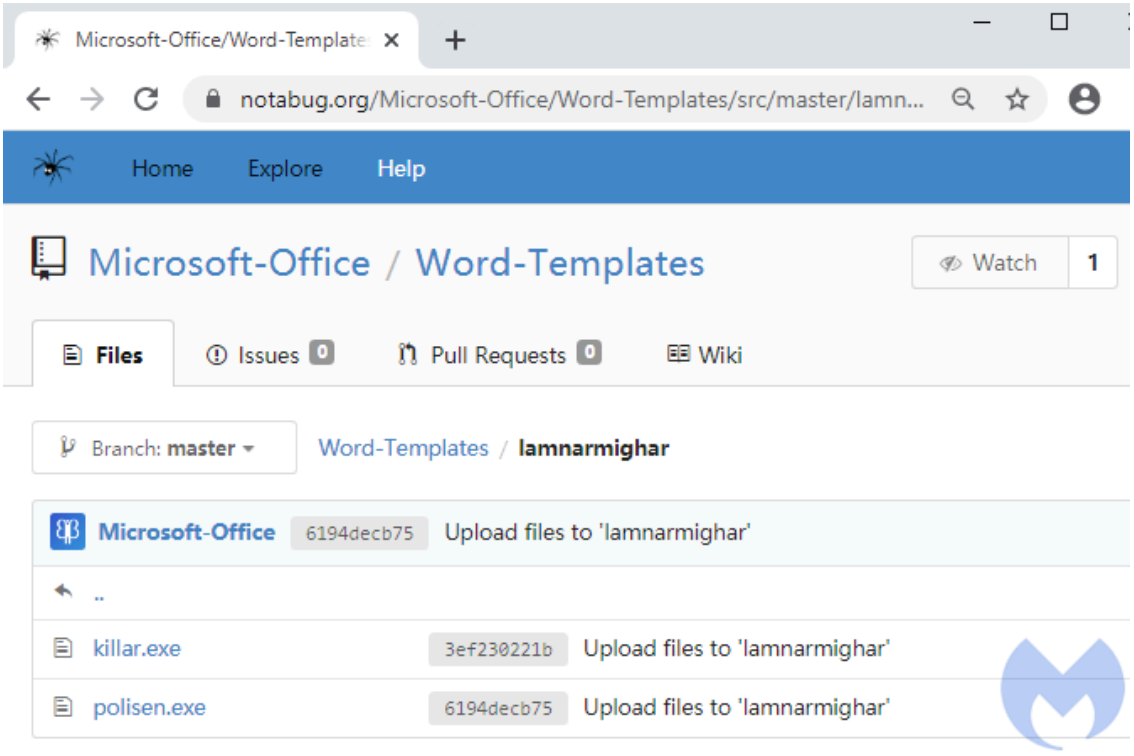
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



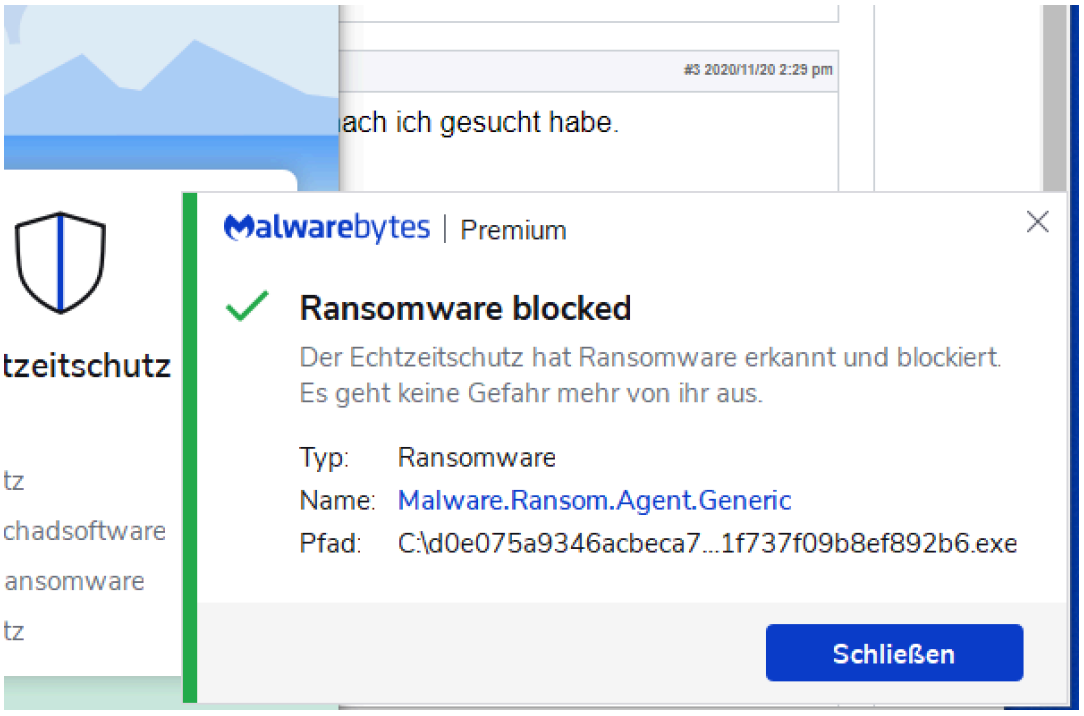
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

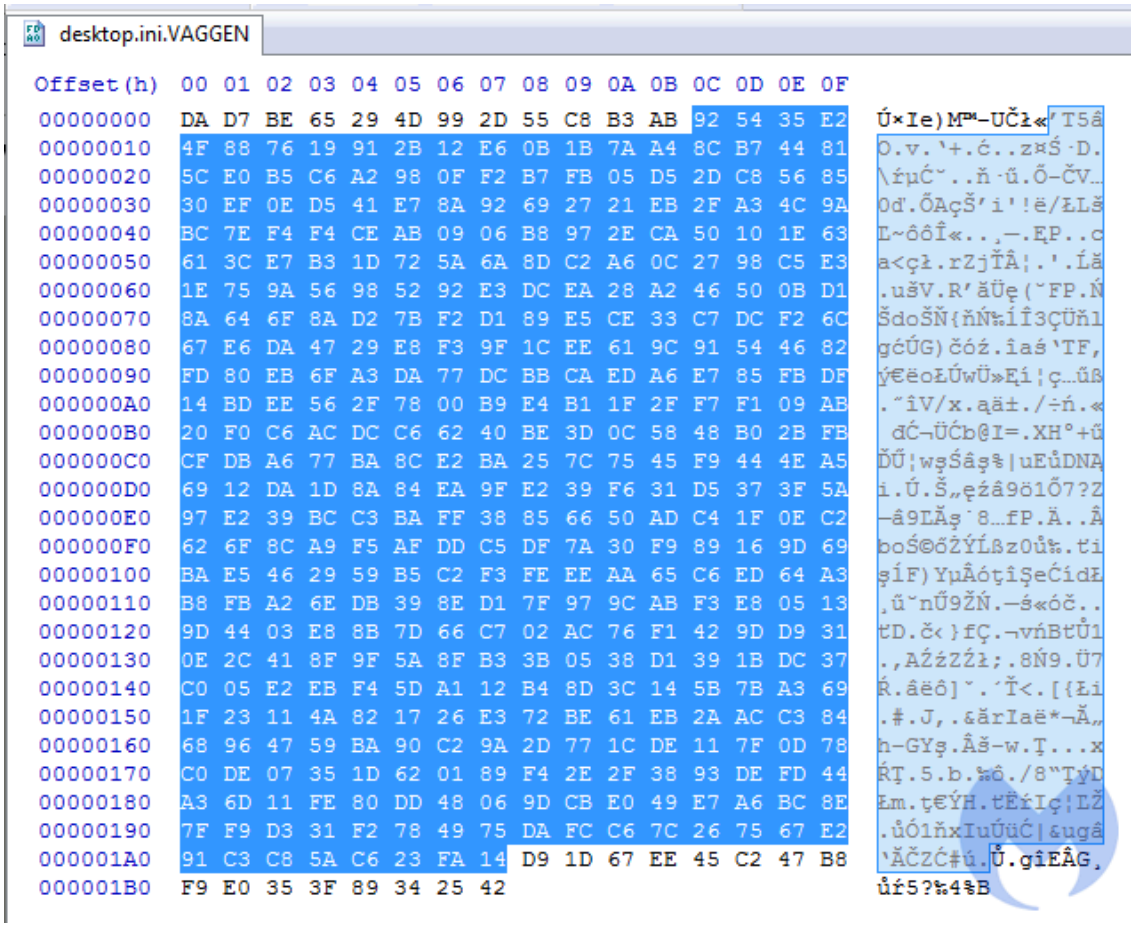
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

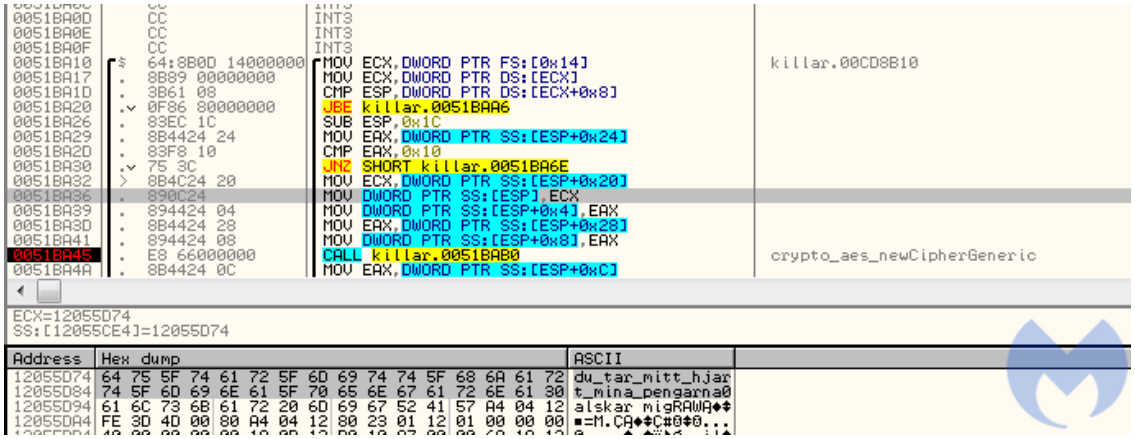


The content of the output file (with .VAGGEN extension) contains:

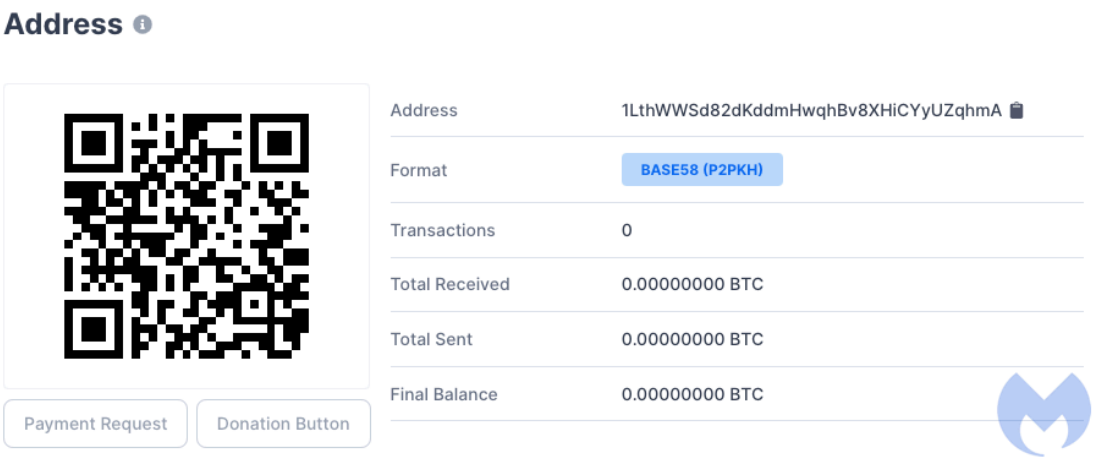
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

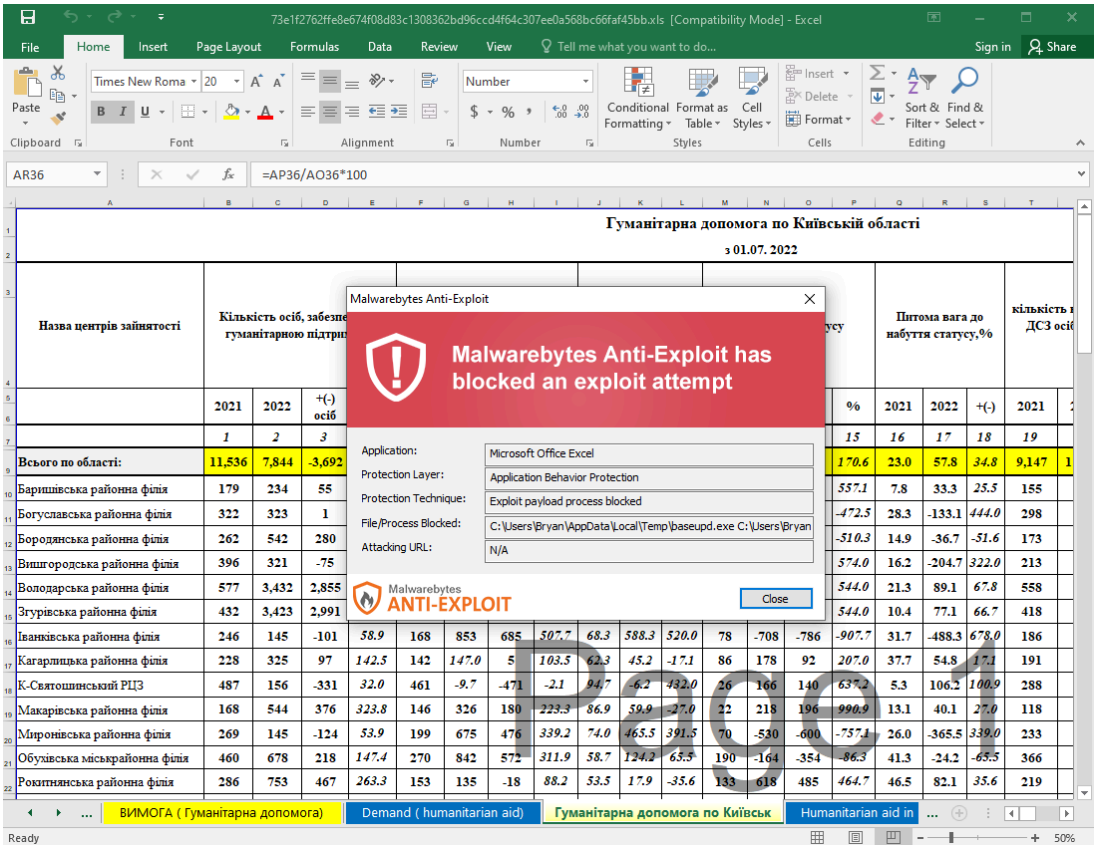
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

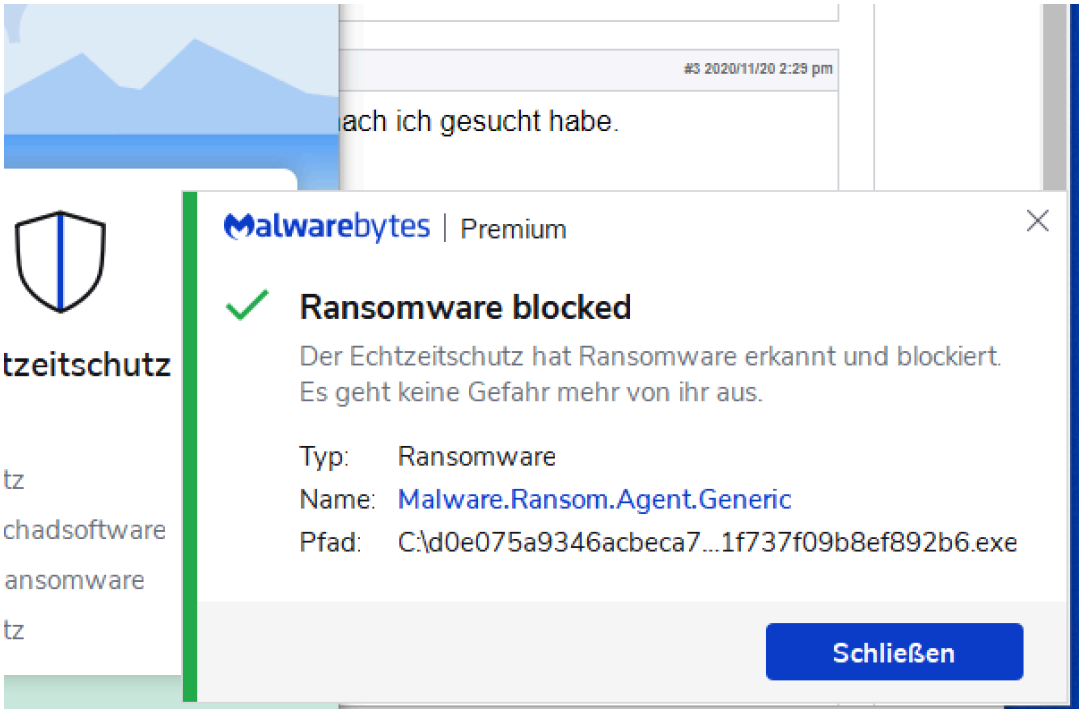
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

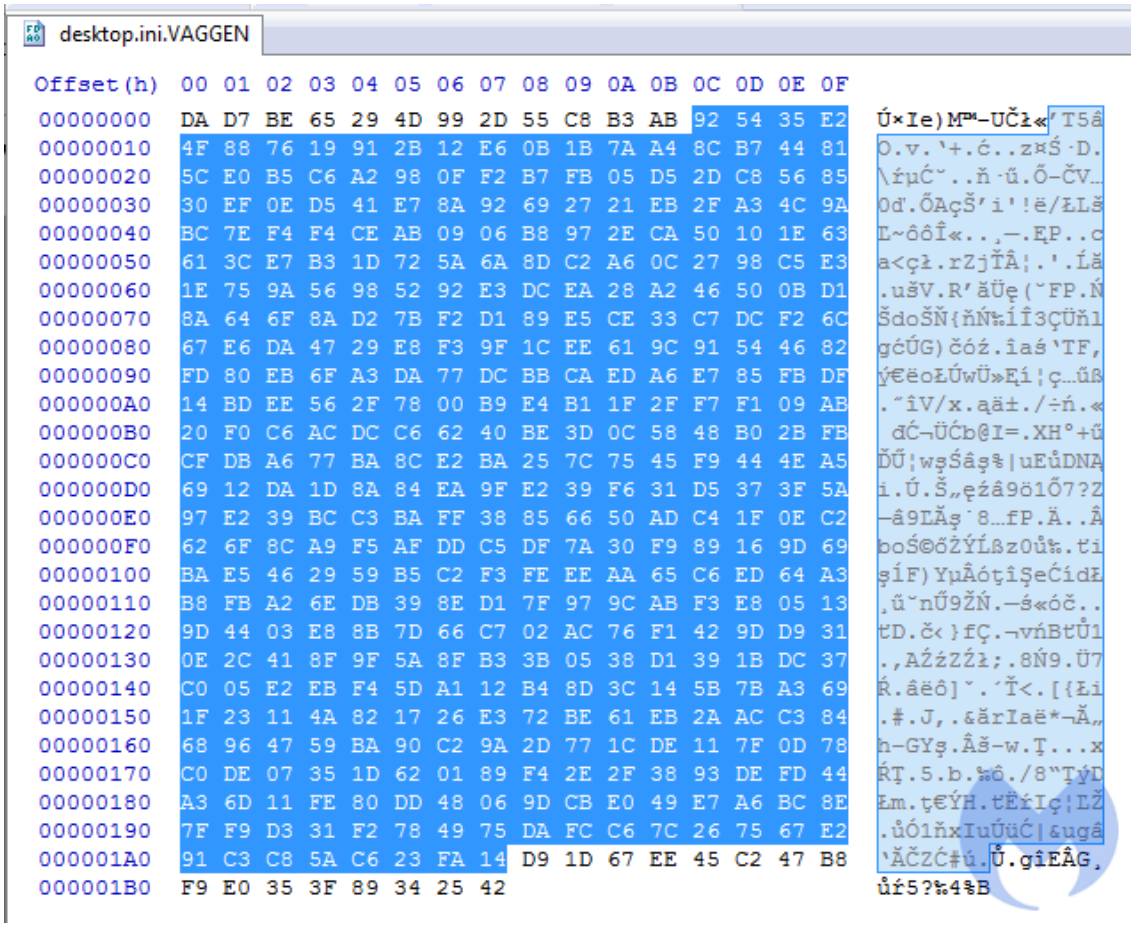
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

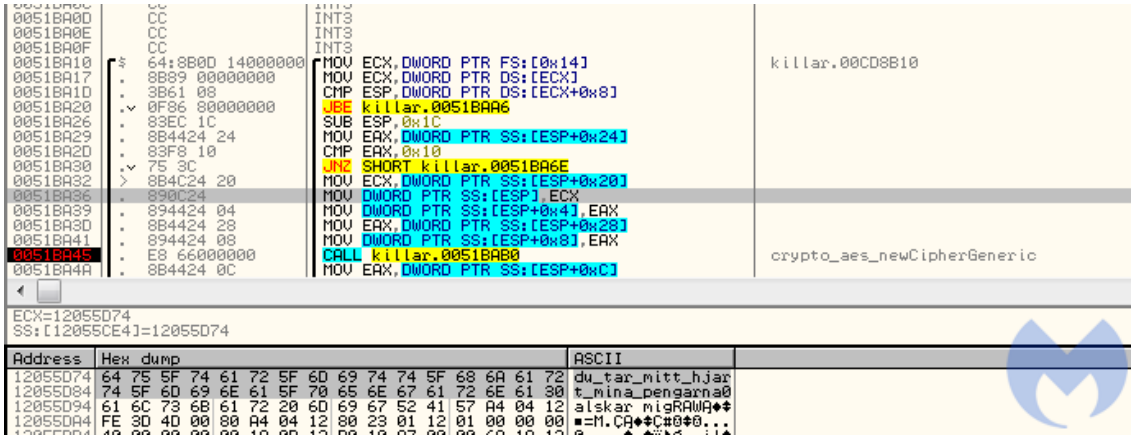


The content of the output file (with .VAGGEN extension) contains:

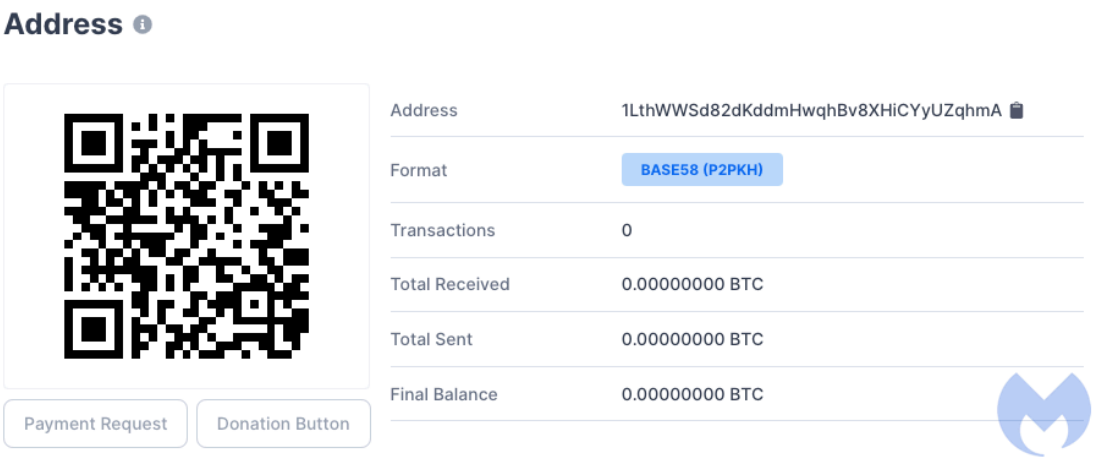
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

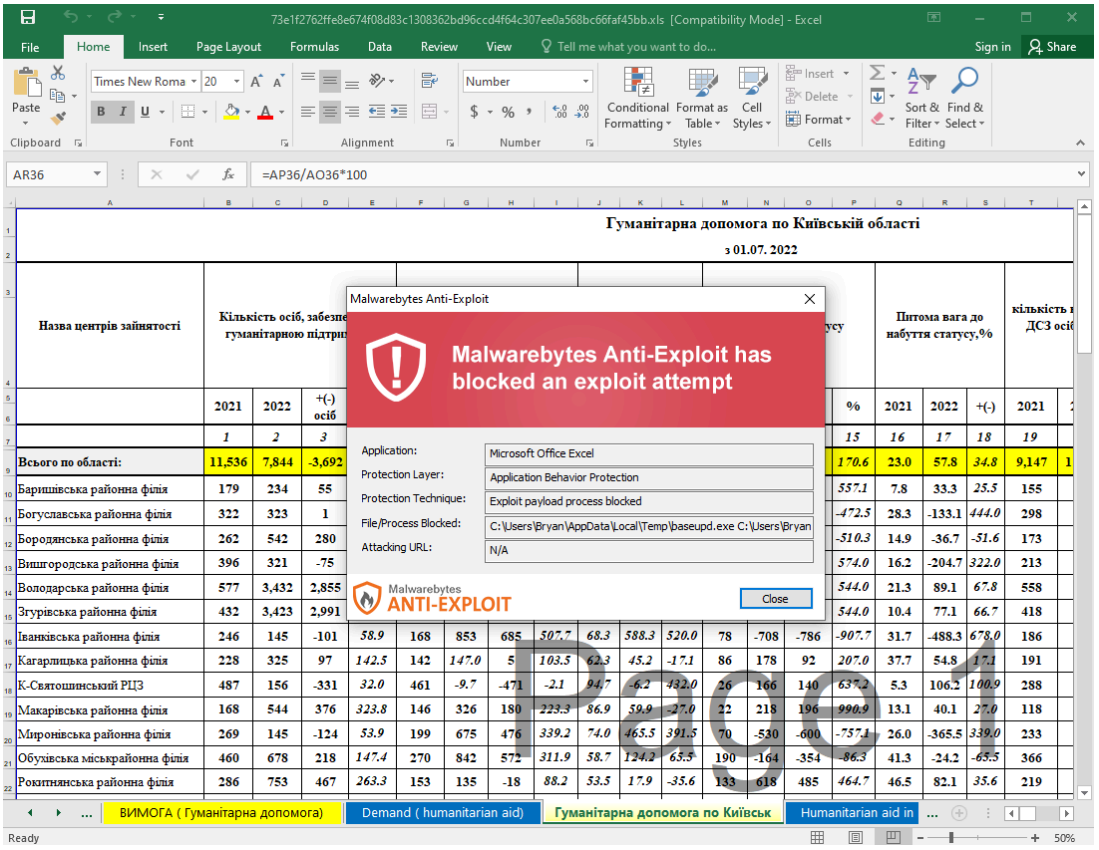
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

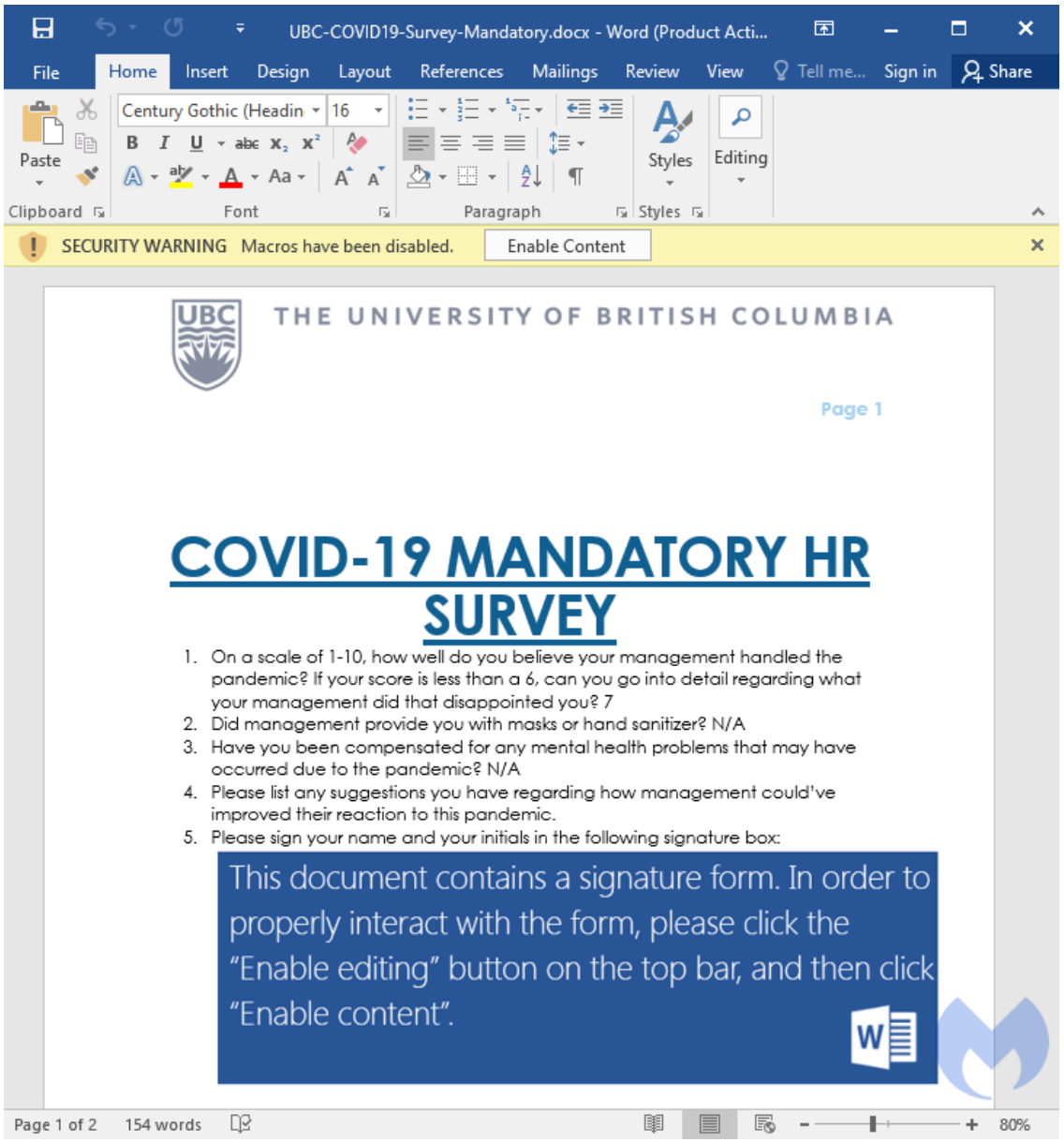
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

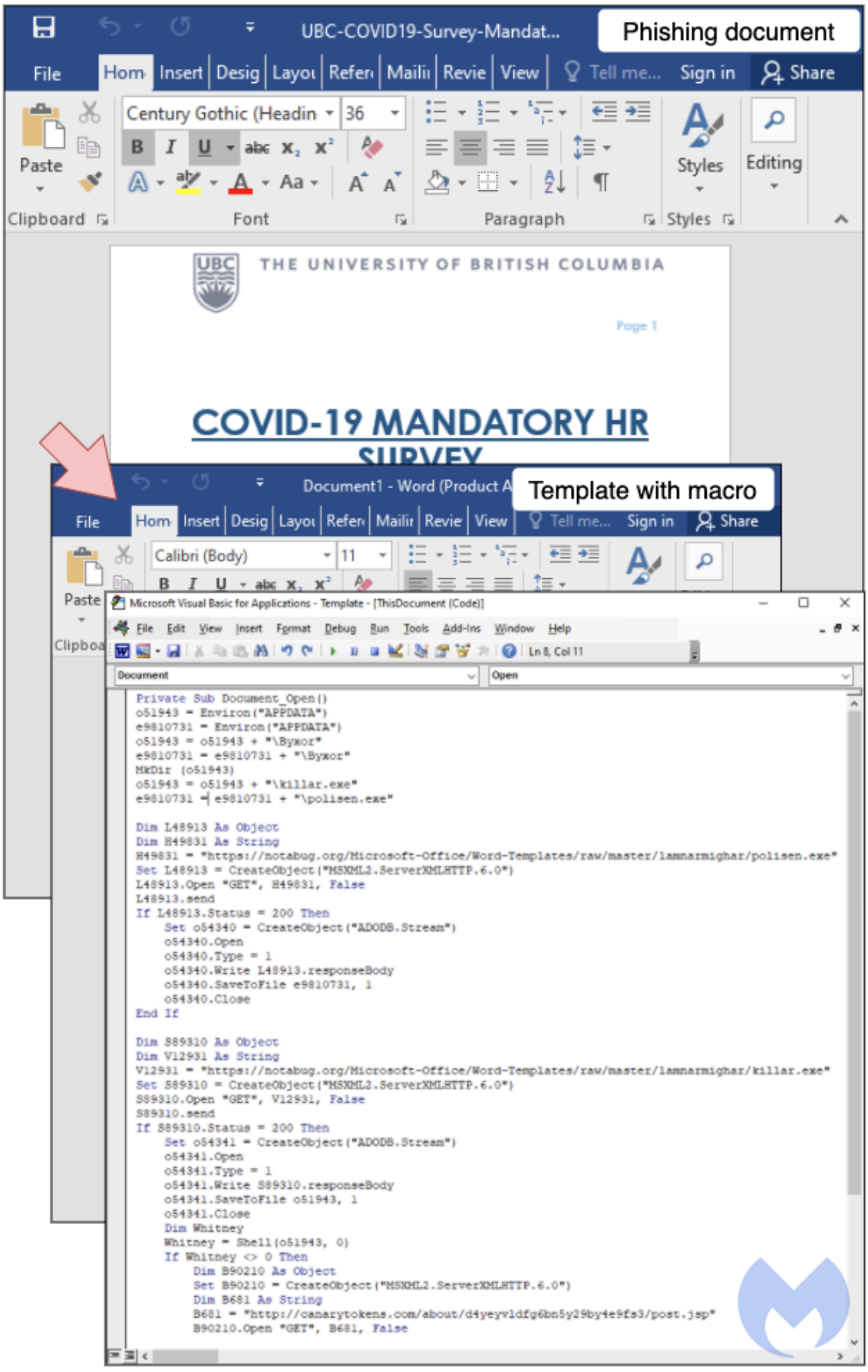
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

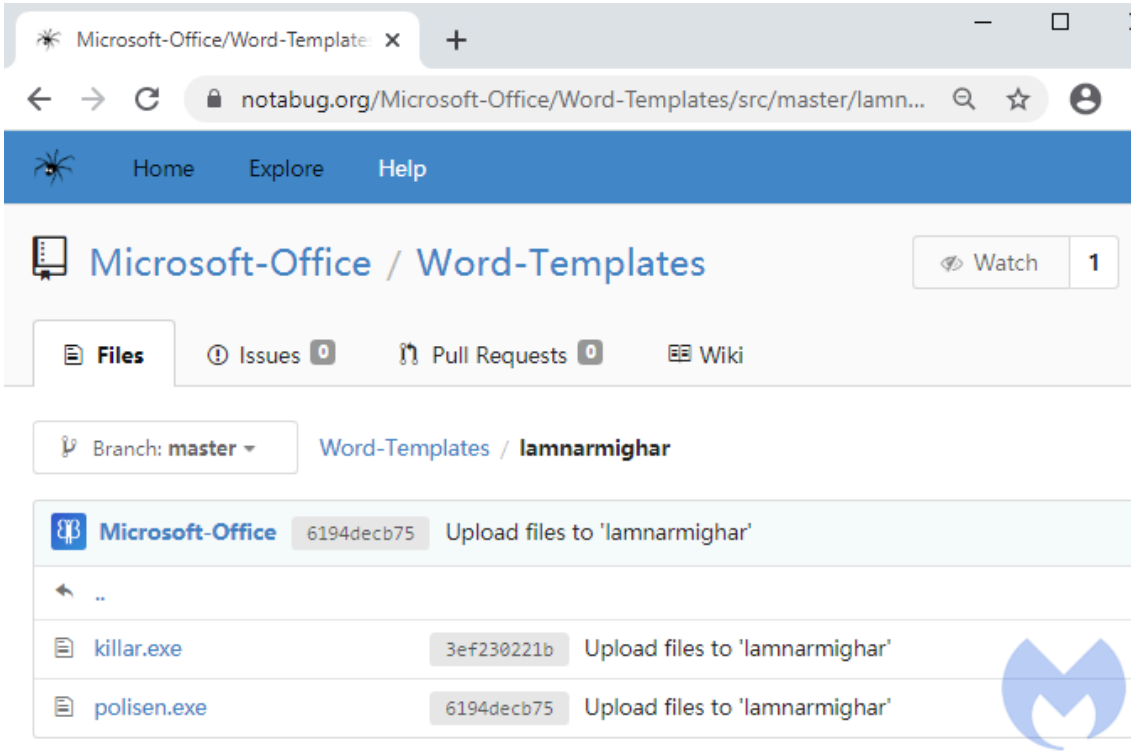
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



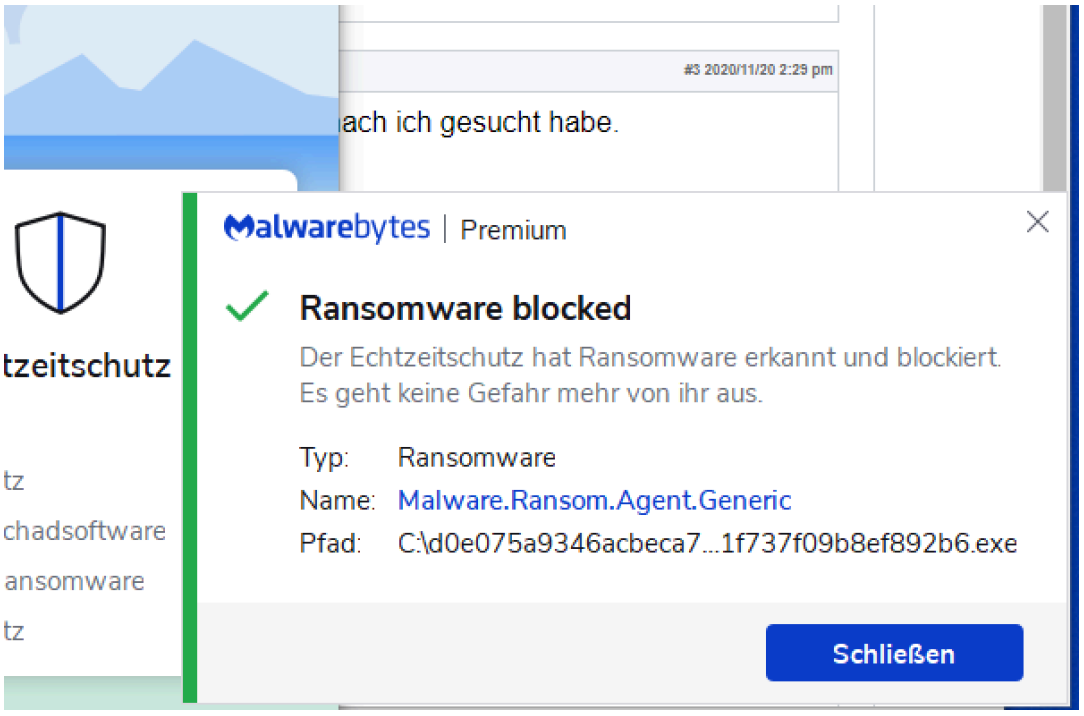
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

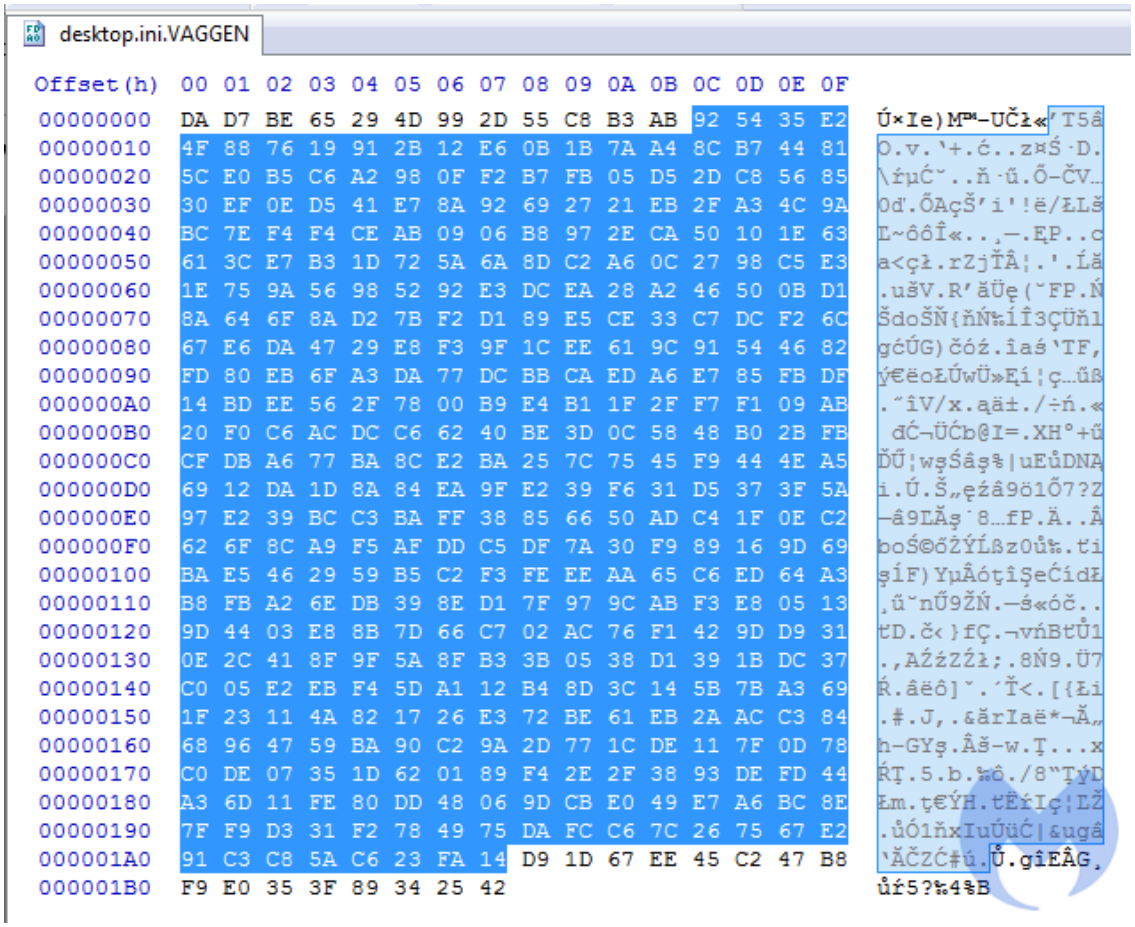
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

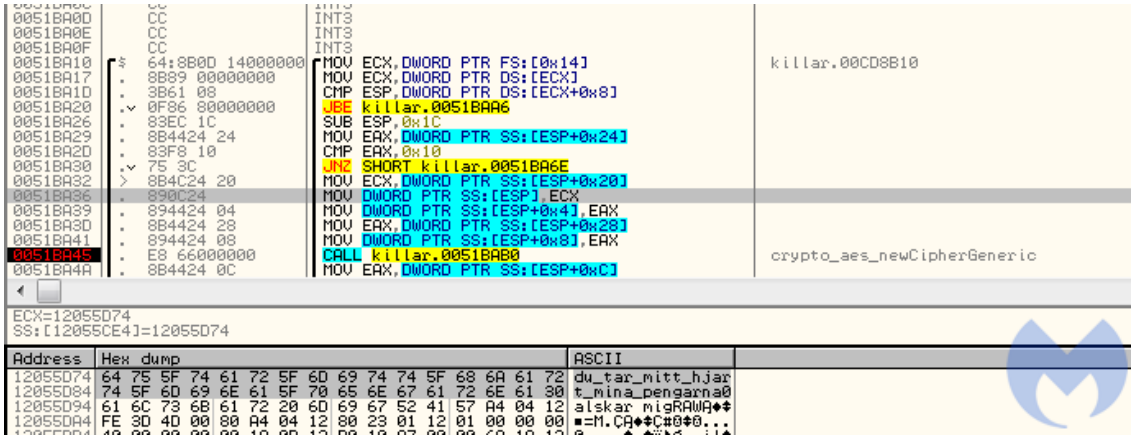


The content of the output file (with .VAGGEN extension) contains:

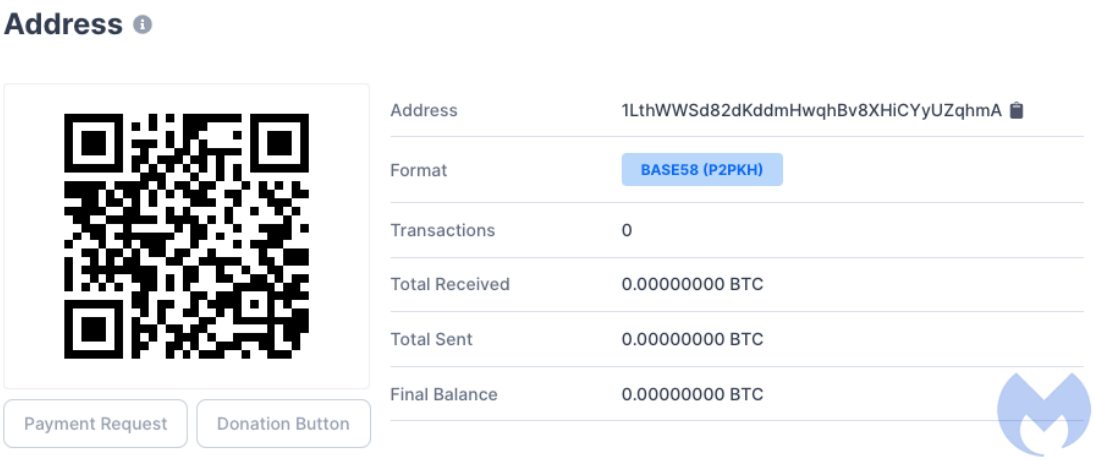
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

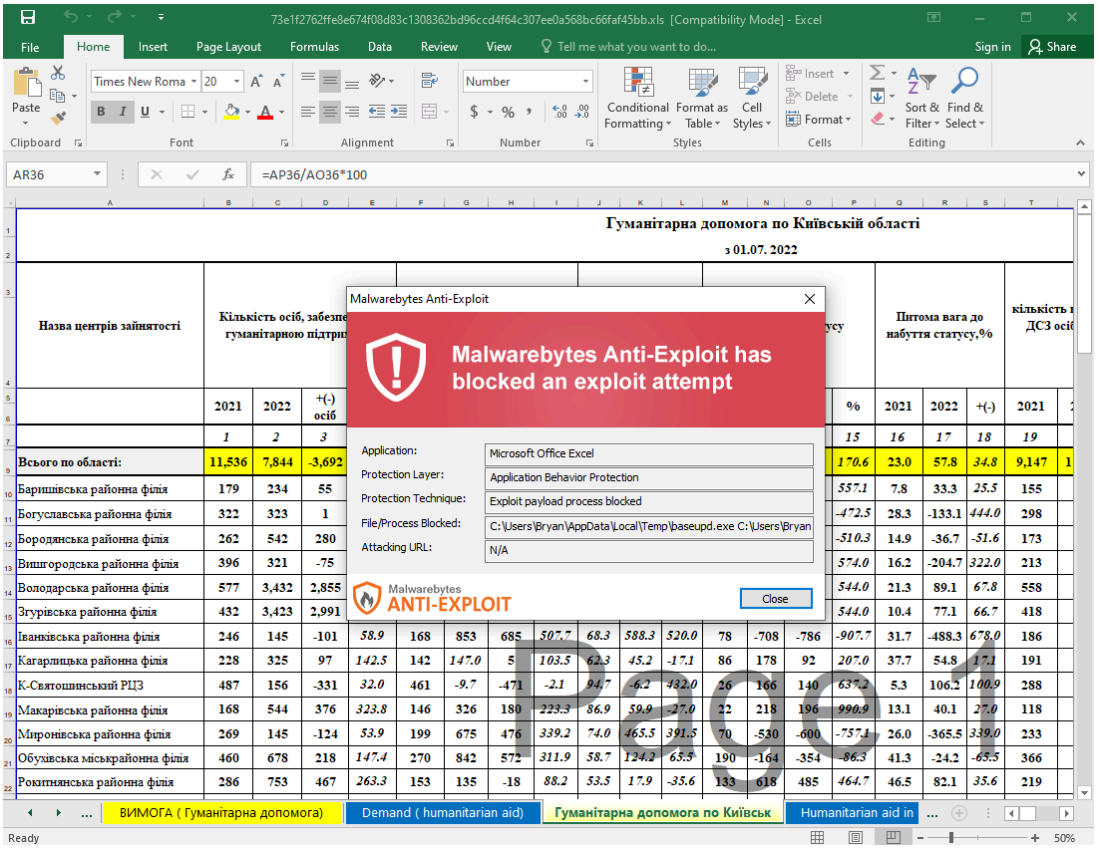
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

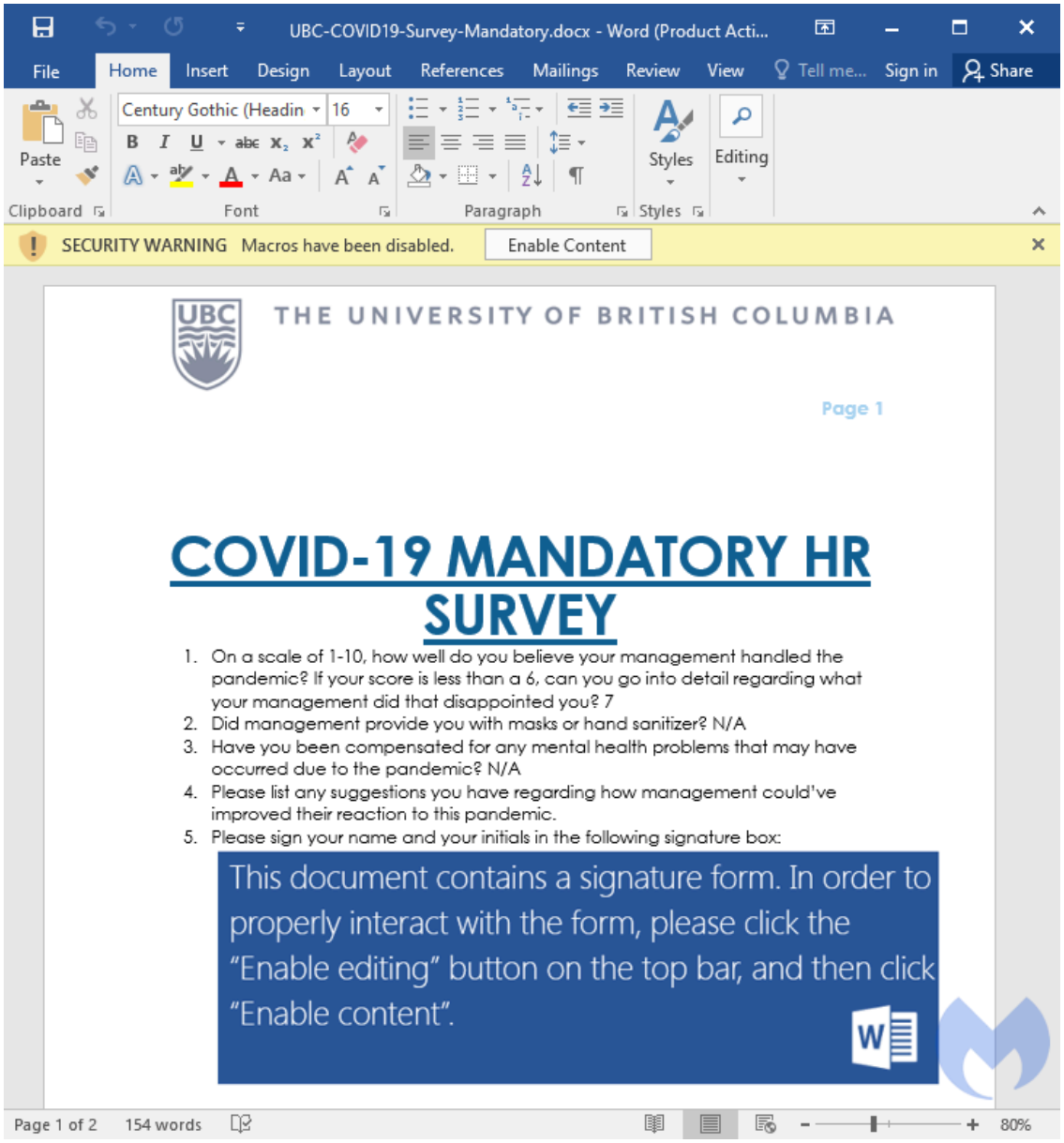
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

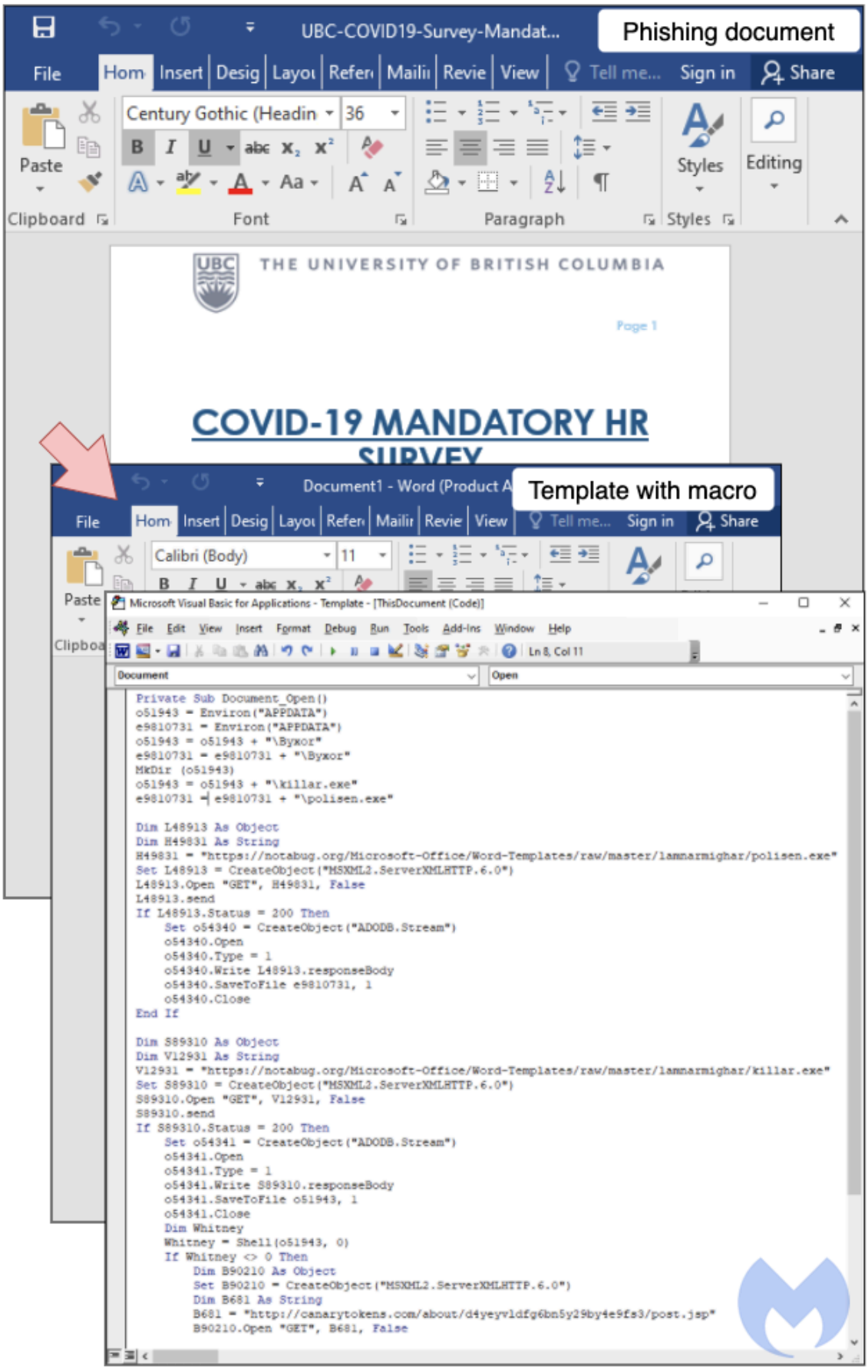
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

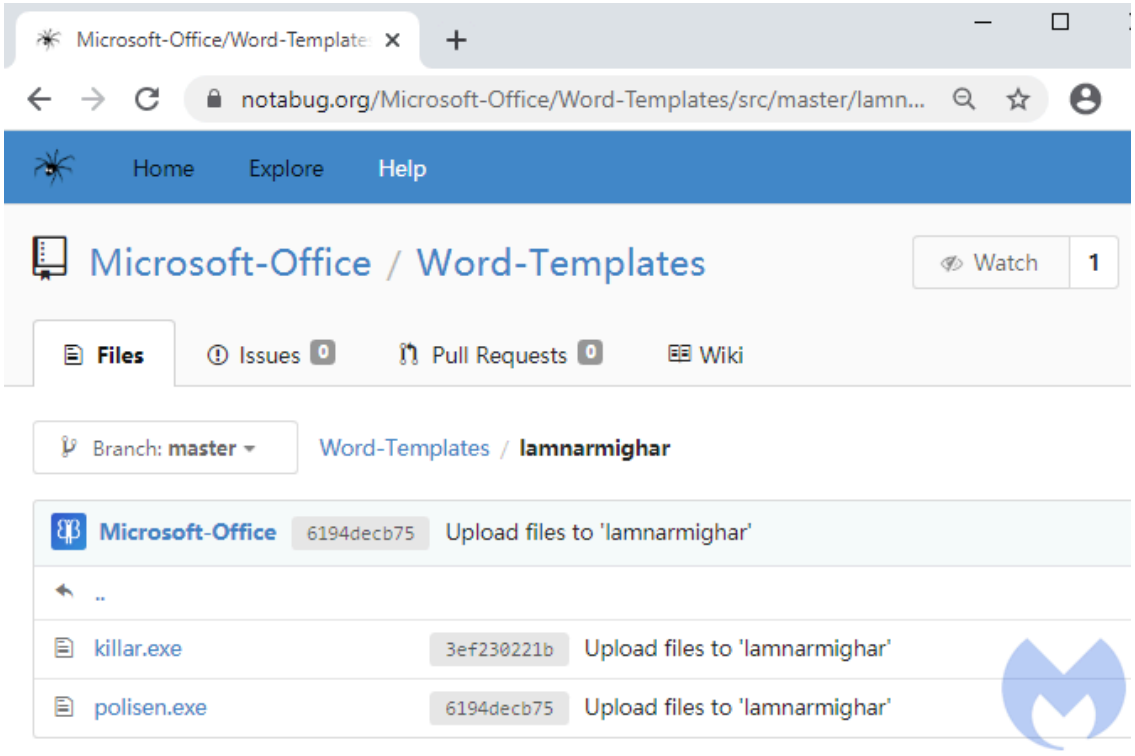
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



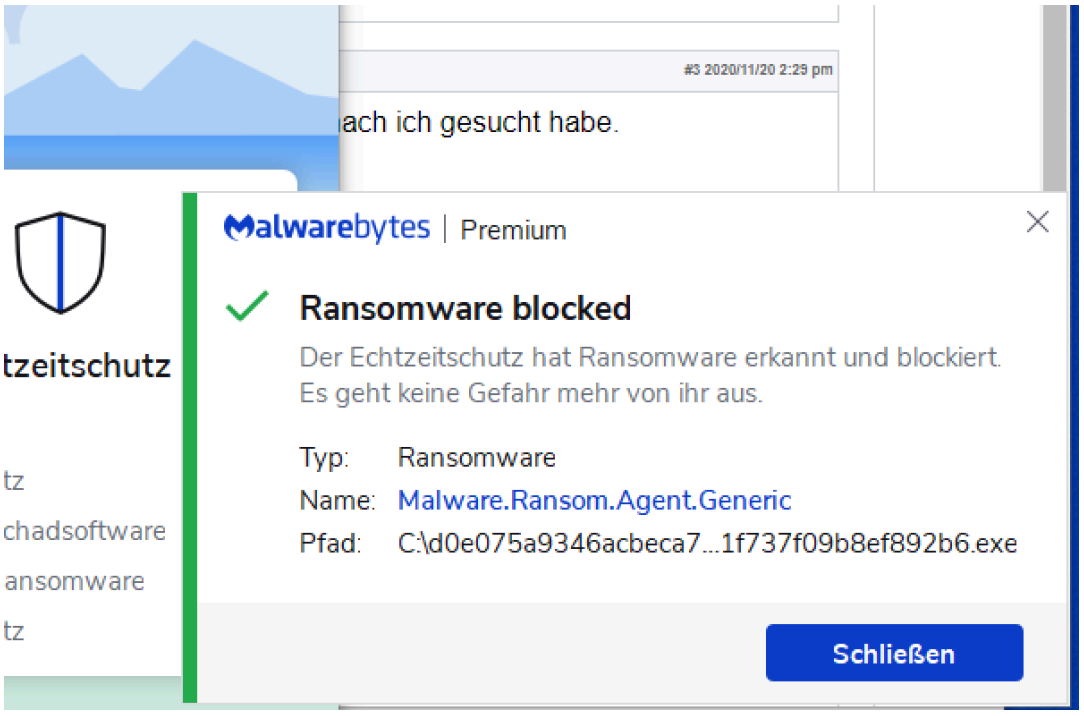
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

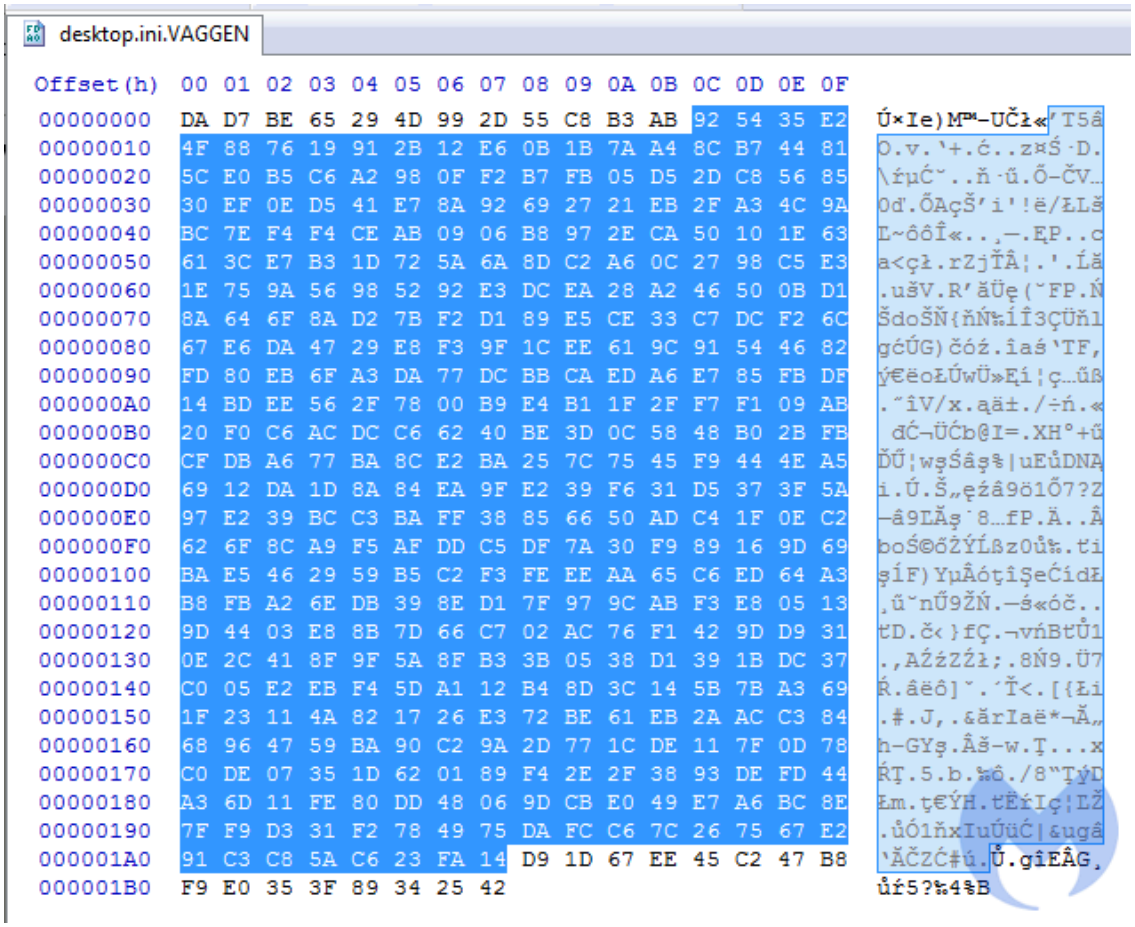
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

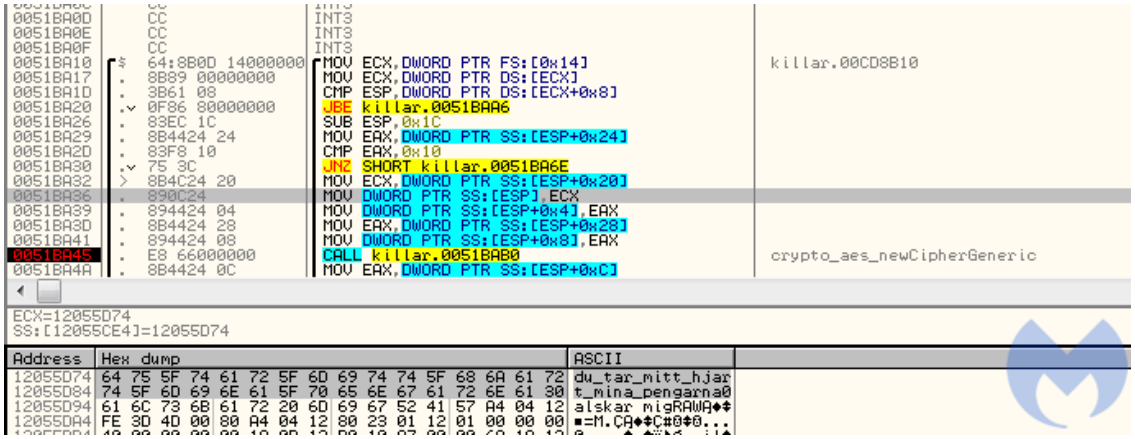


The content of the output file (with .VAGGEN extension) contains:

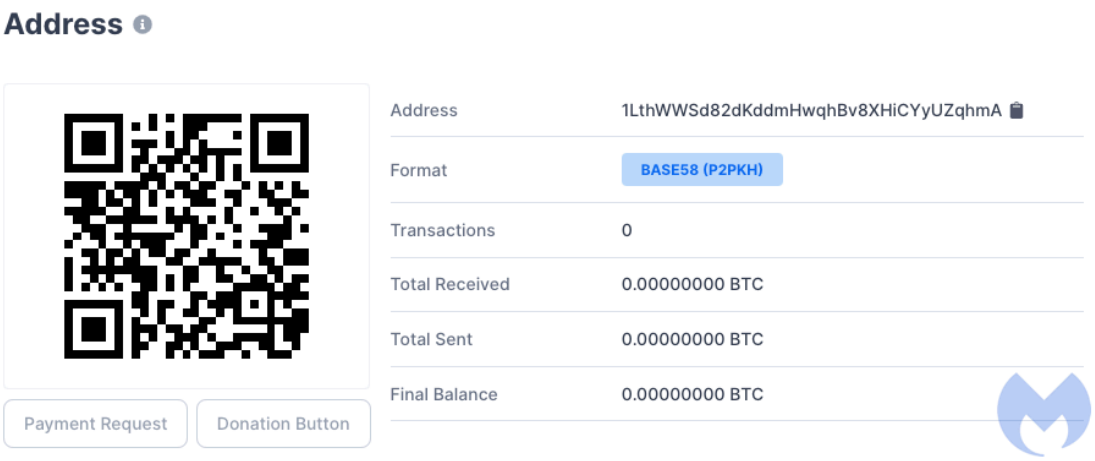
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

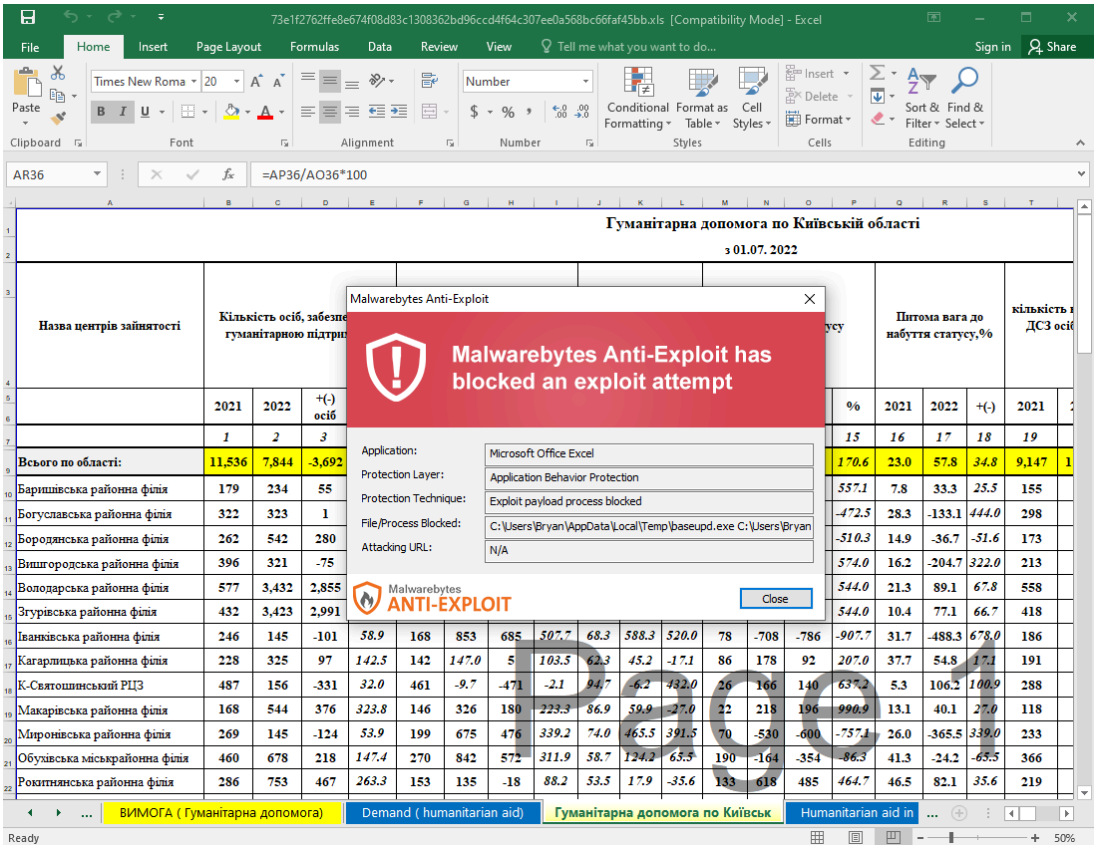
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

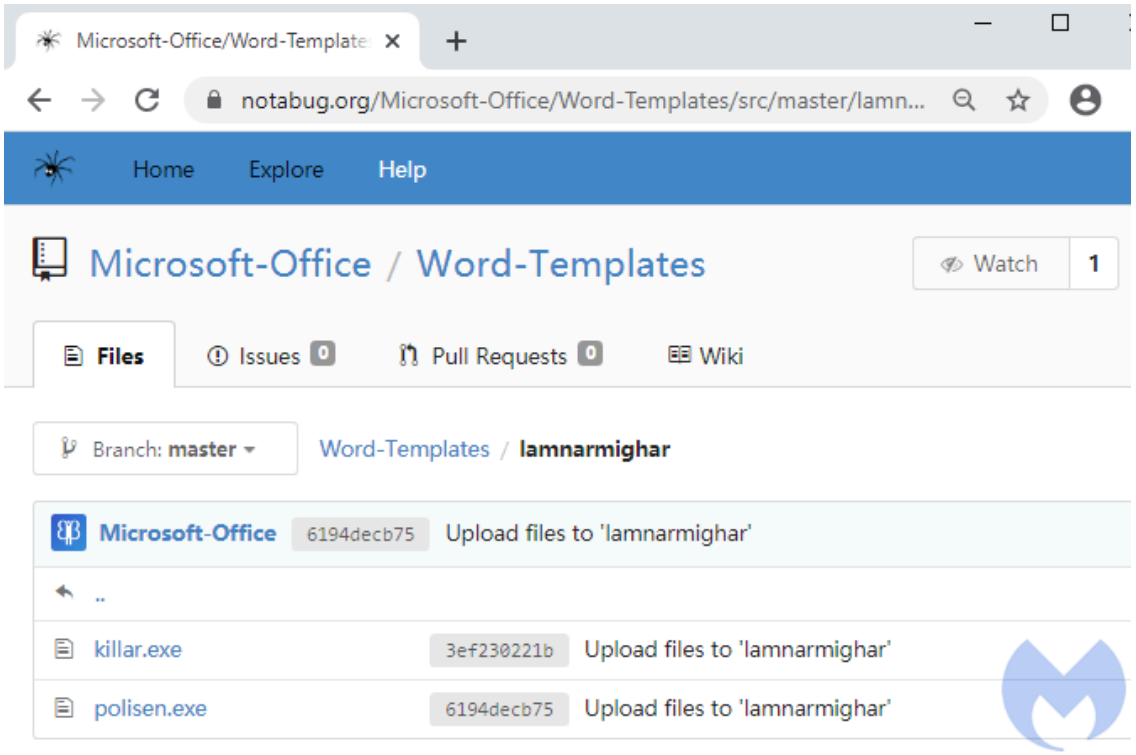
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



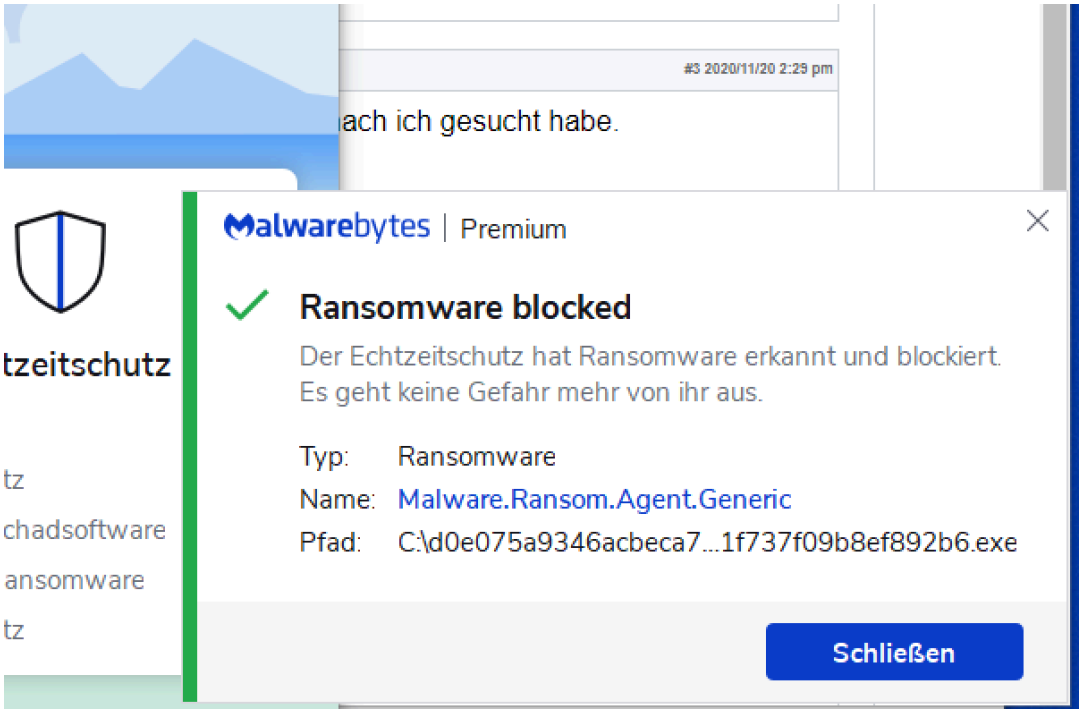
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

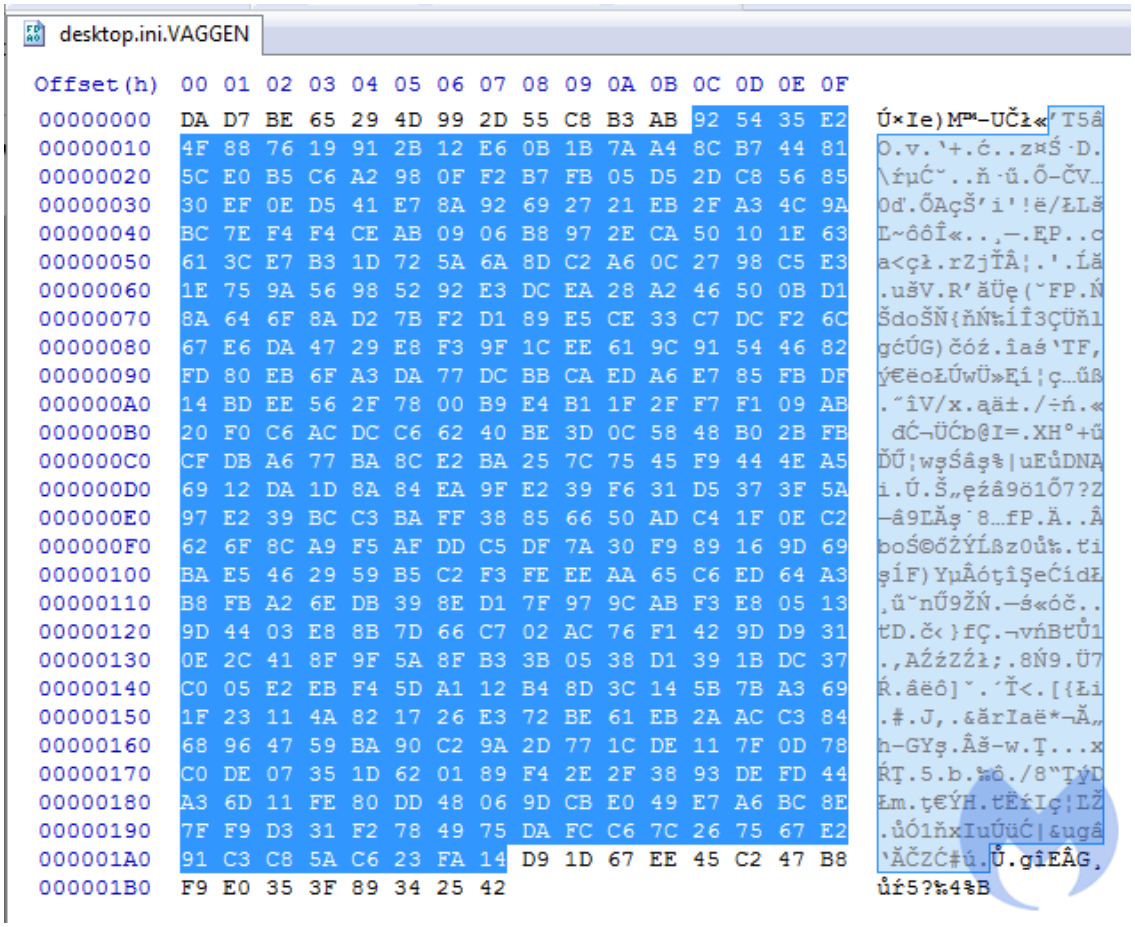
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

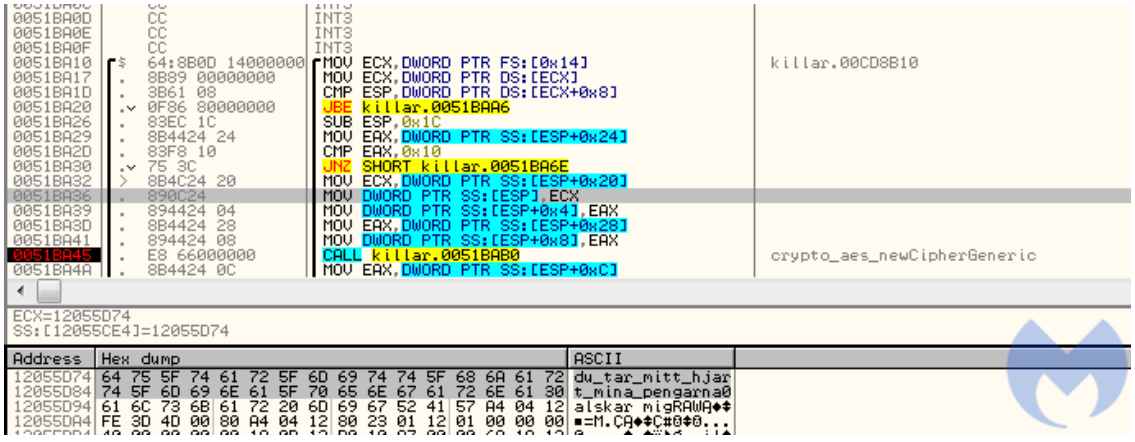


The content of the output file (with .VAGGEN extension) contains:

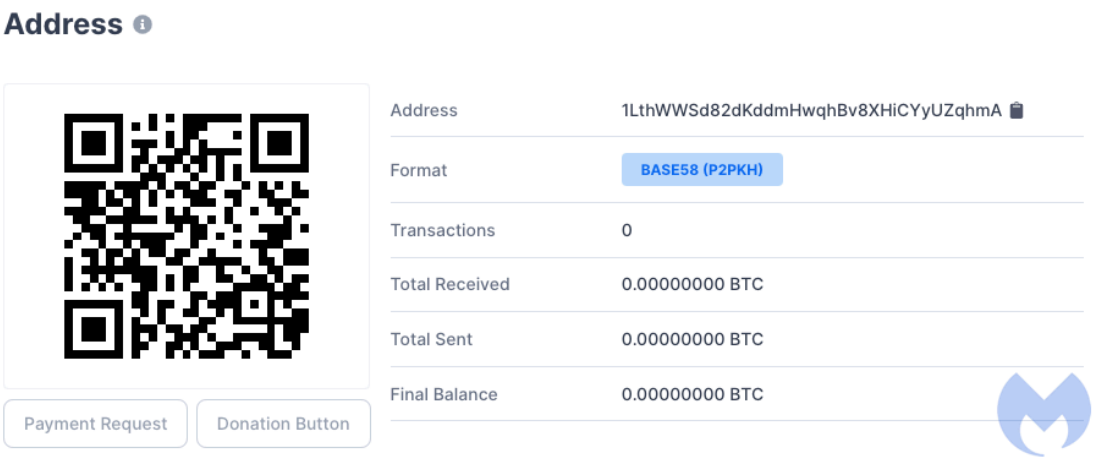
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

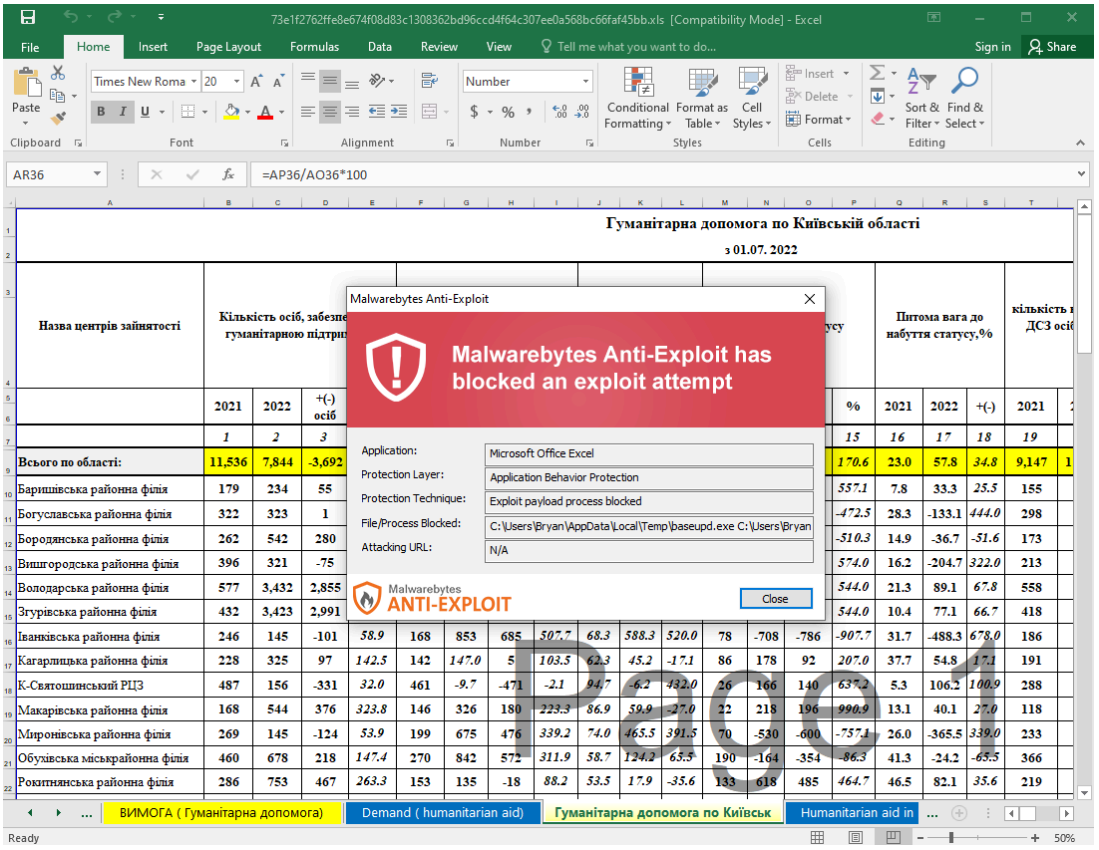
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

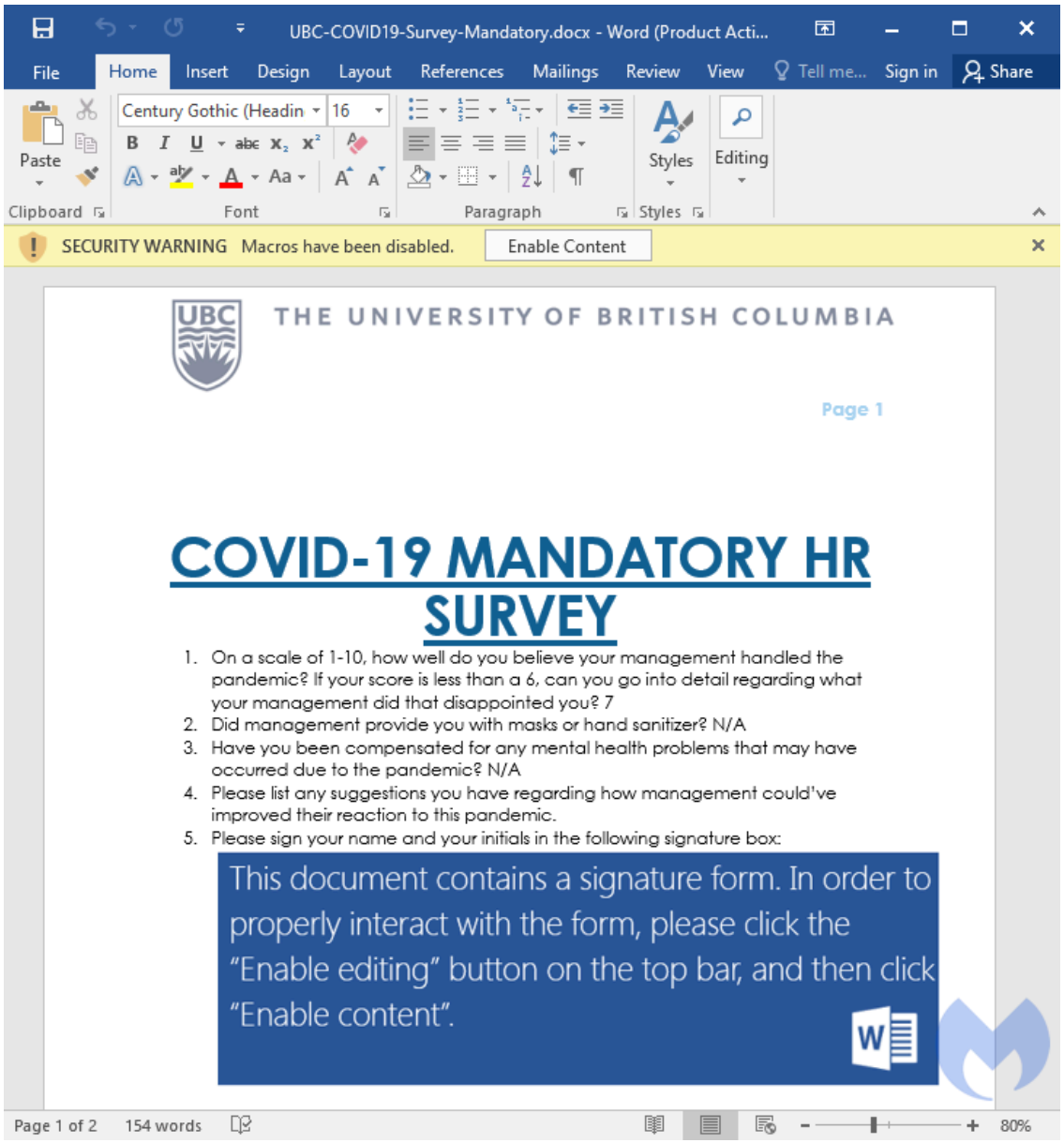
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

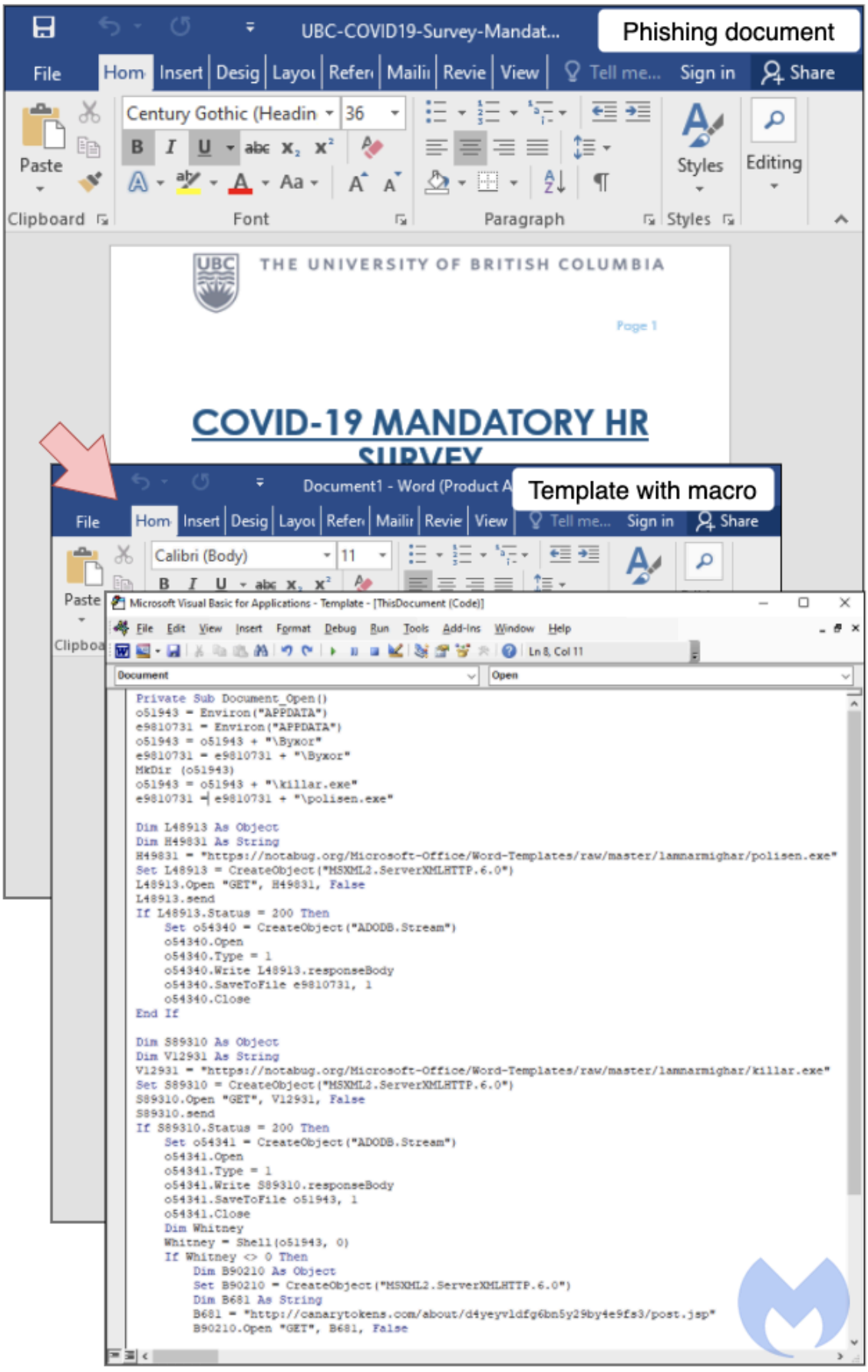
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

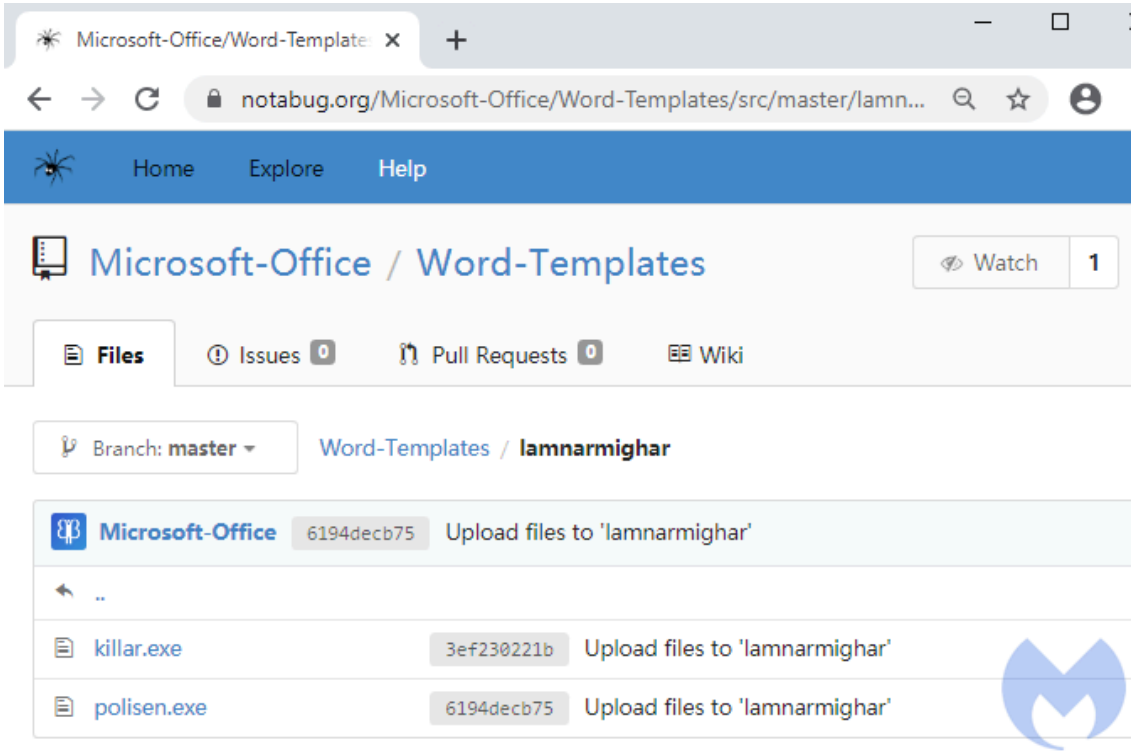
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



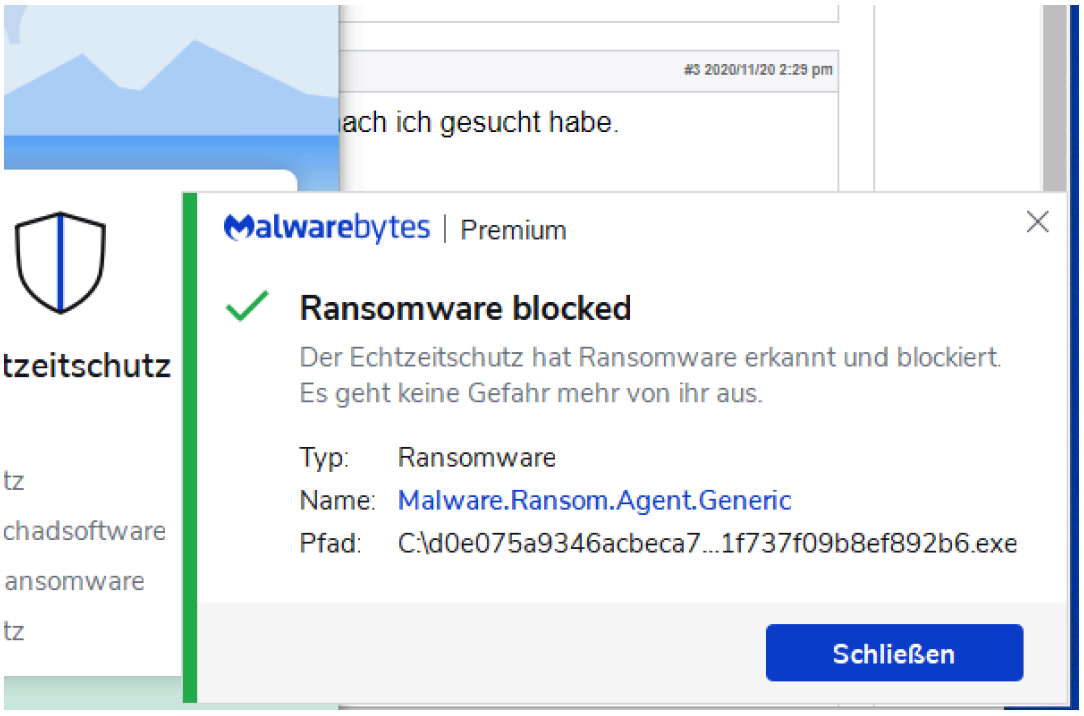
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

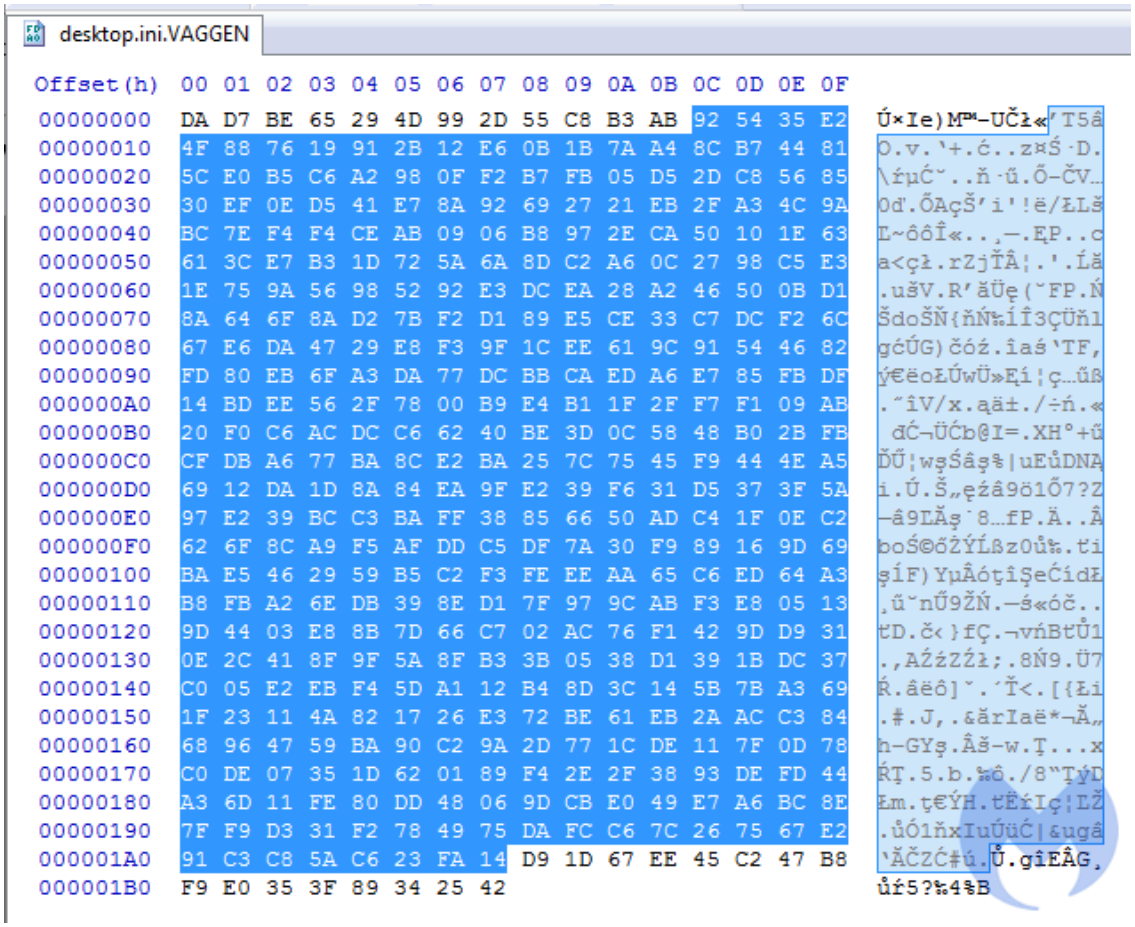
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

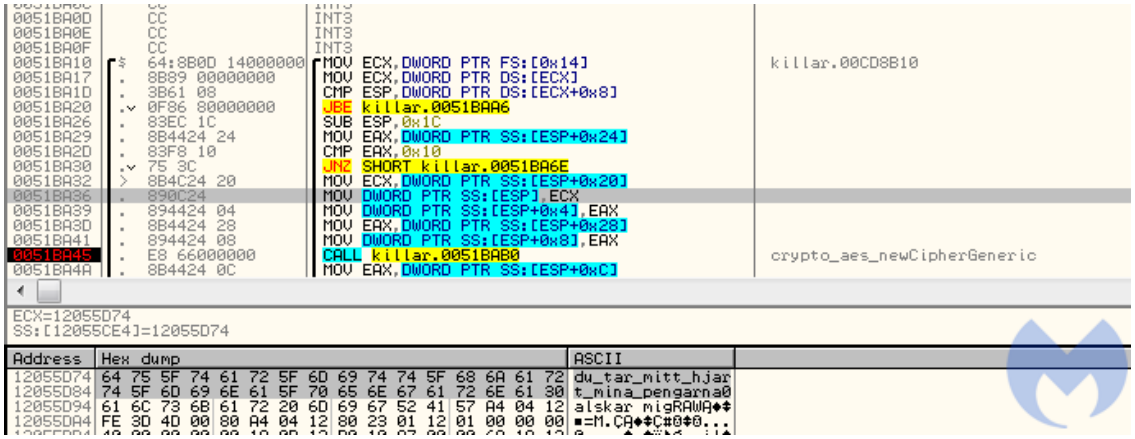


The content of the output file (with .VAGGEN extension) contains:

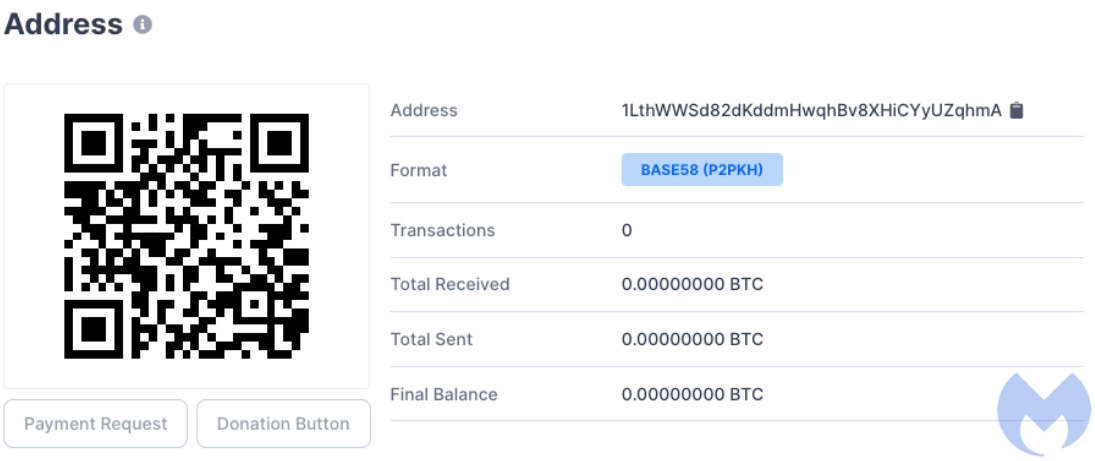
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

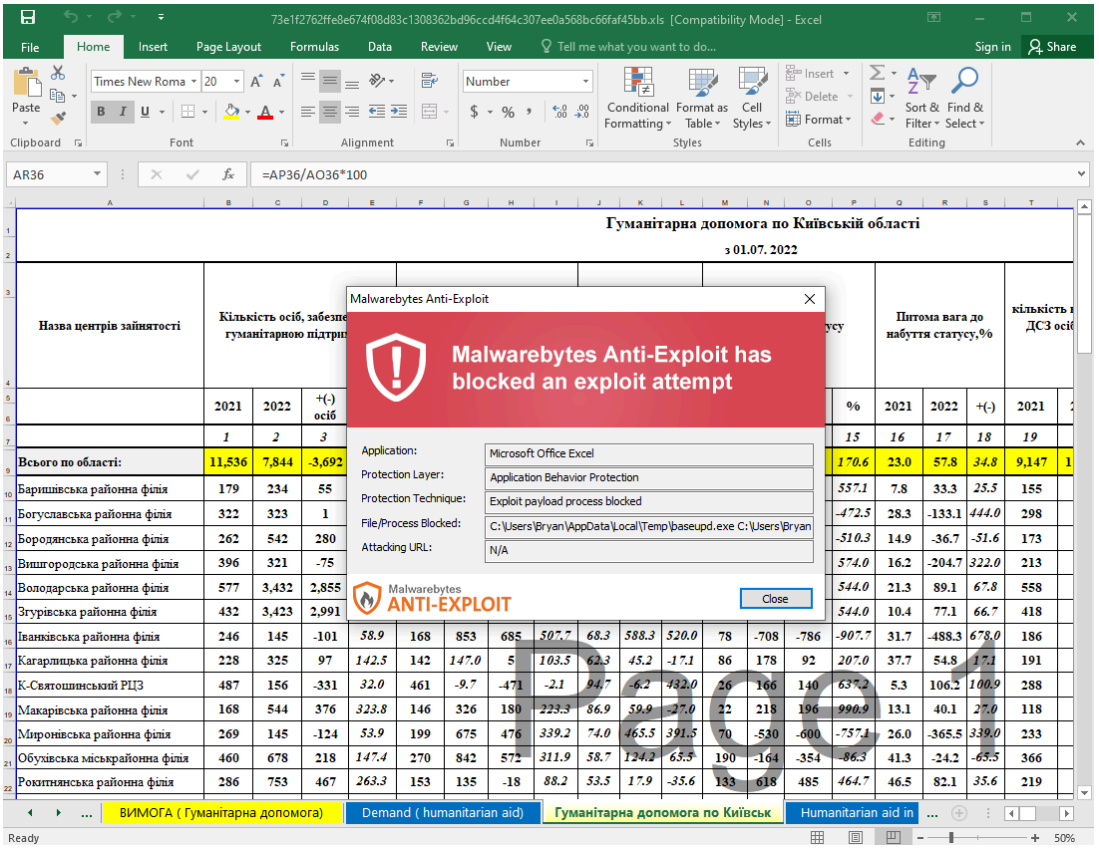
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

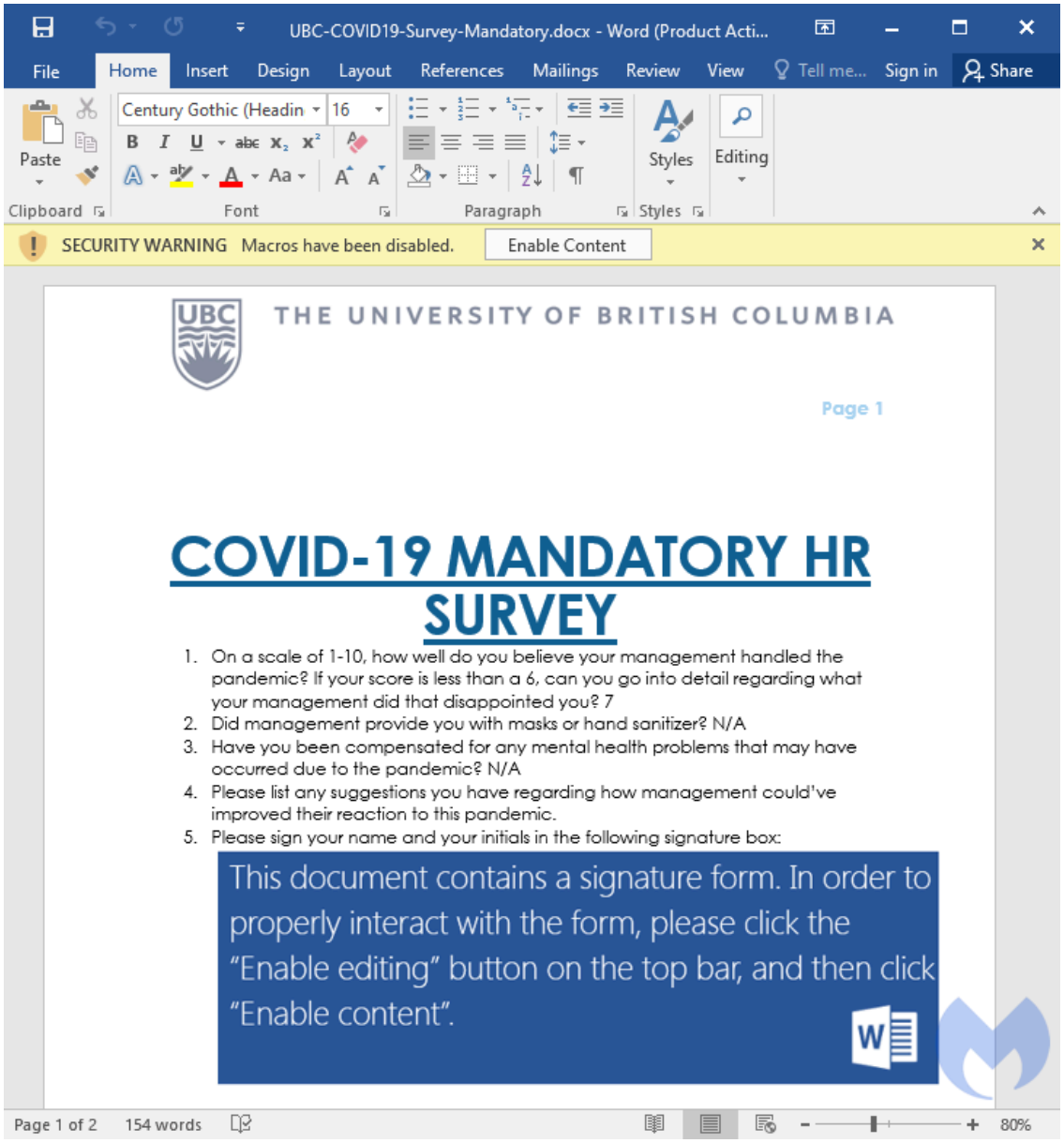
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

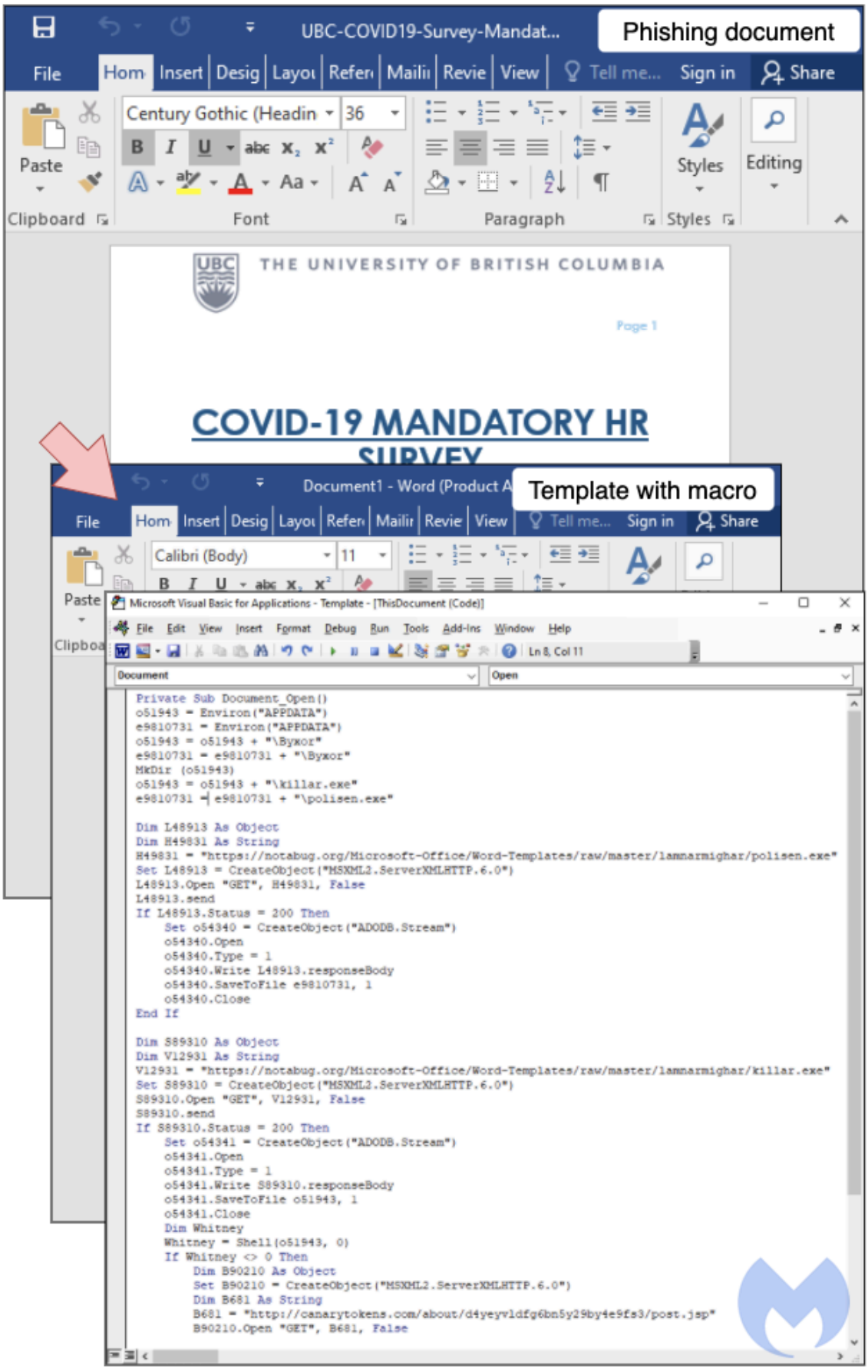
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

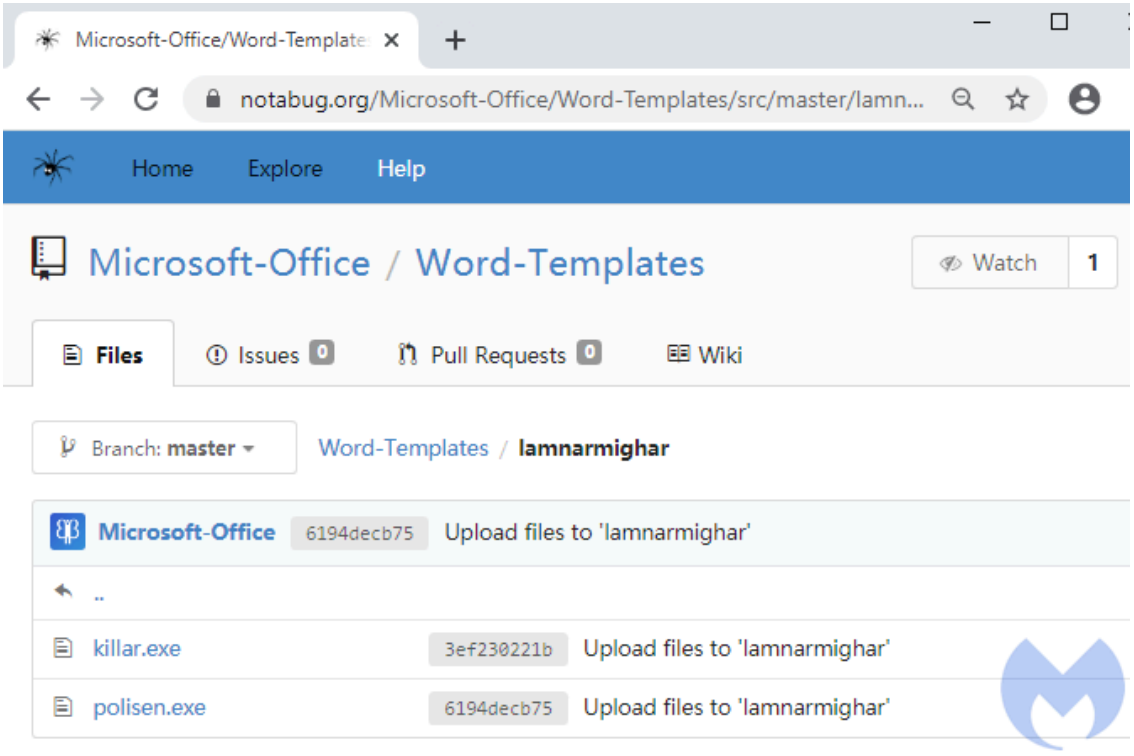
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



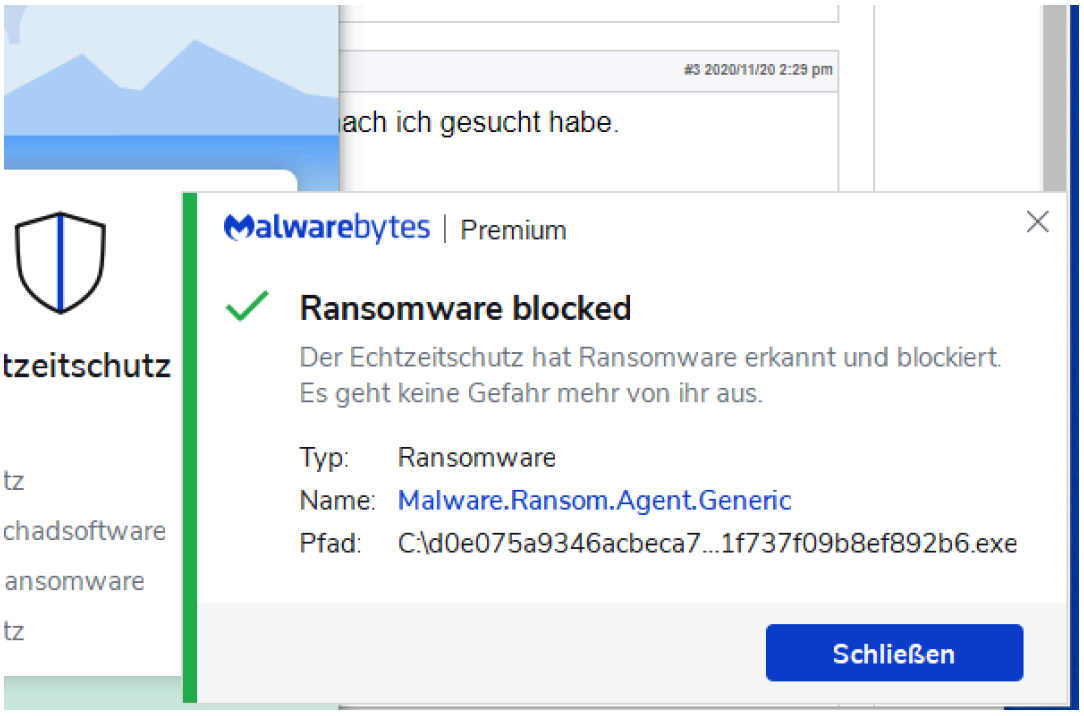
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

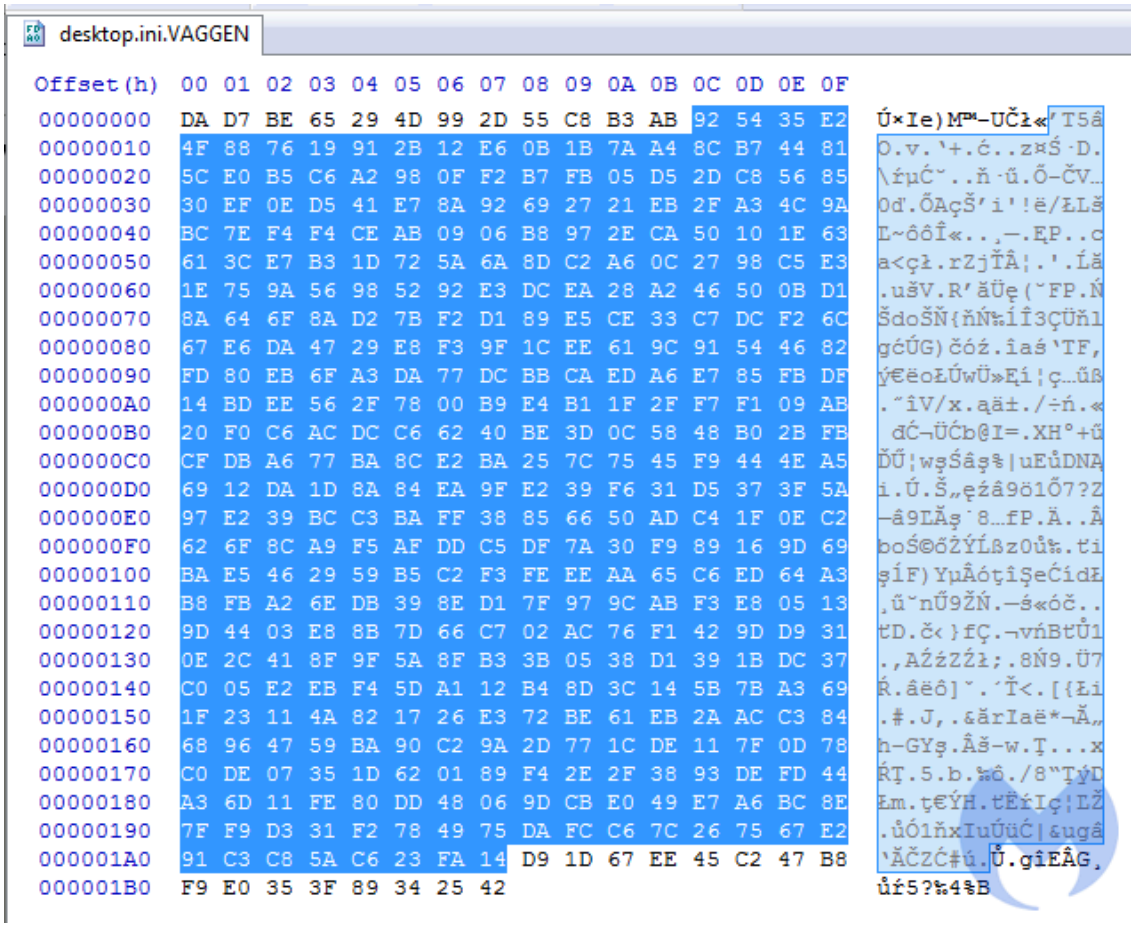
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

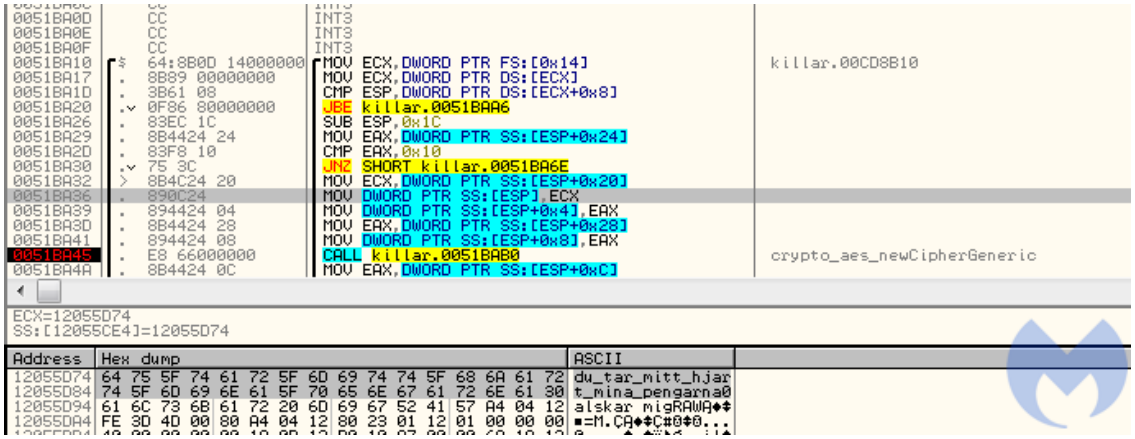


The content of the output file (with .VAGGEN extension) contains:

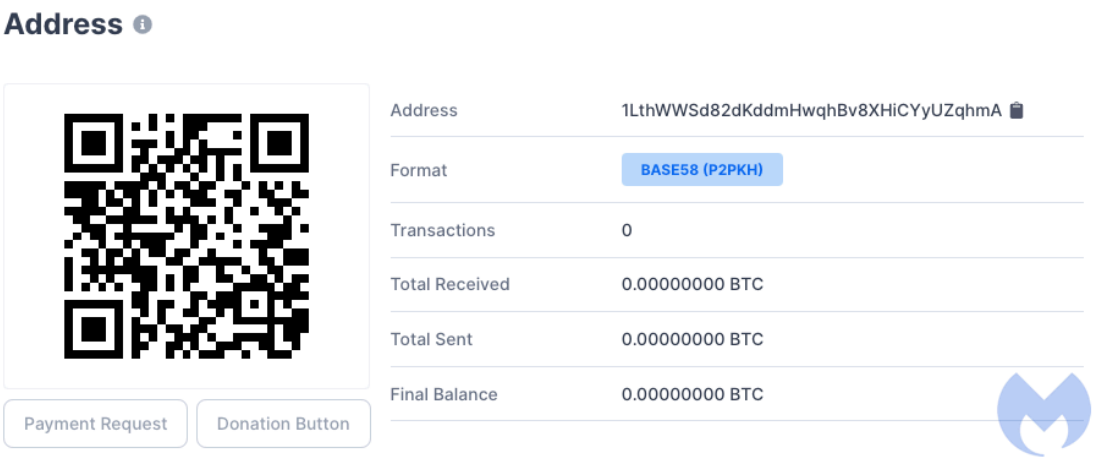
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

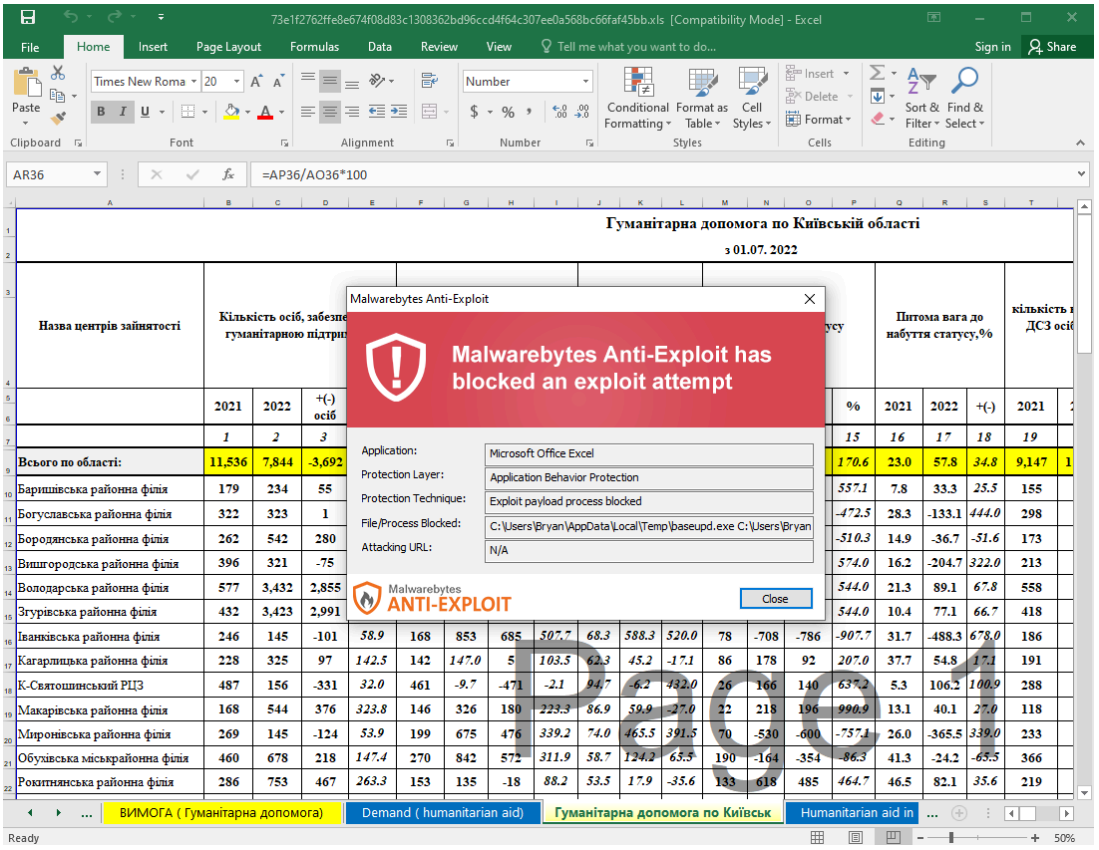
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

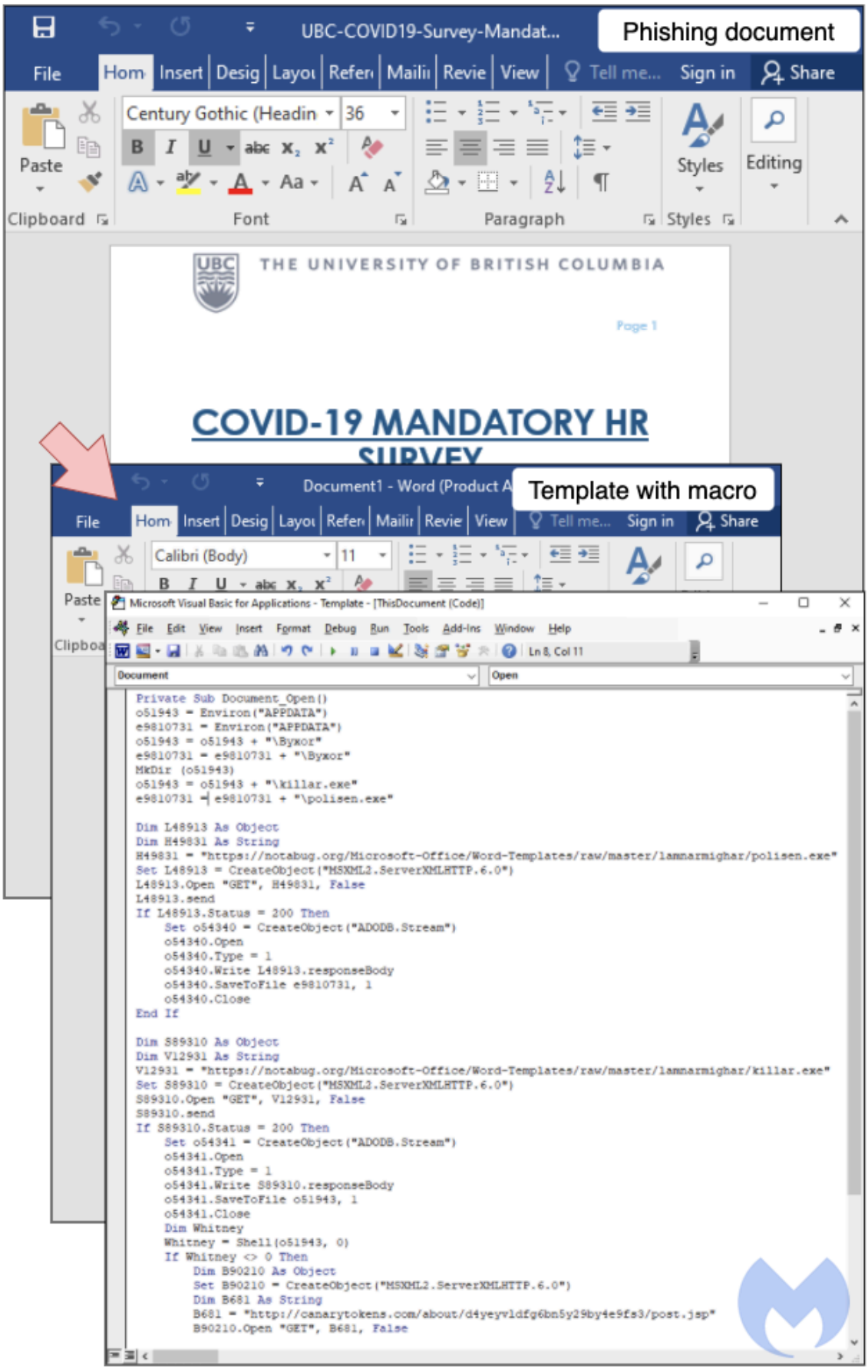
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

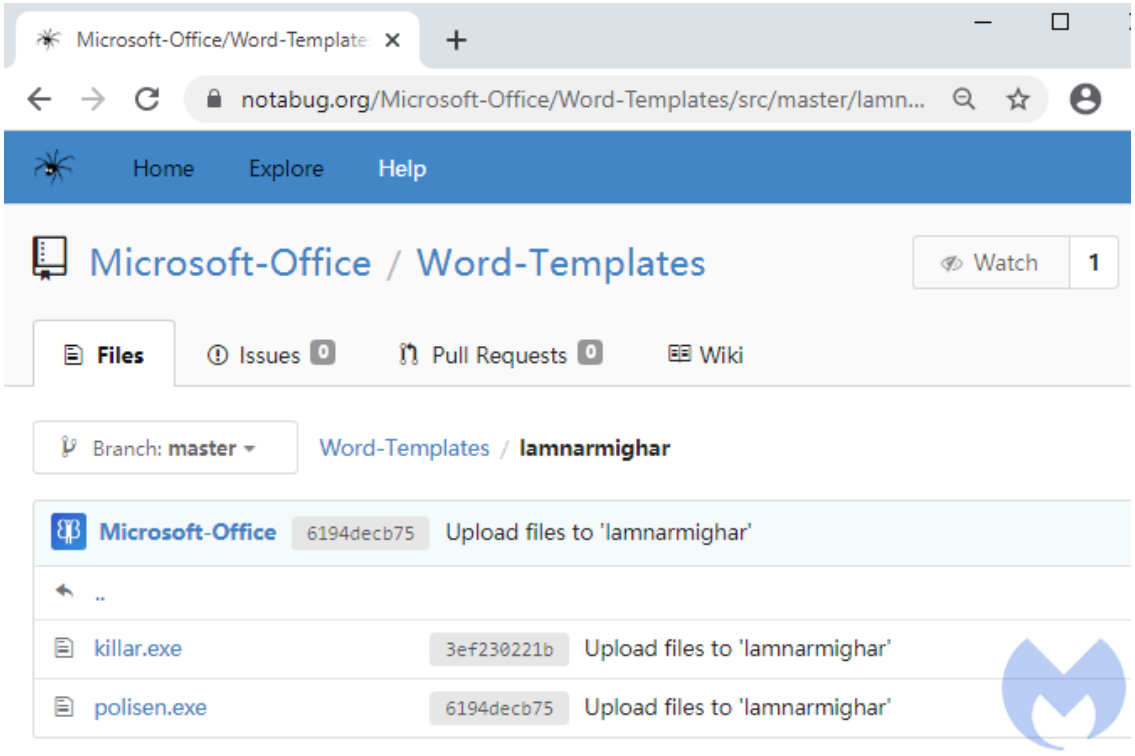
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



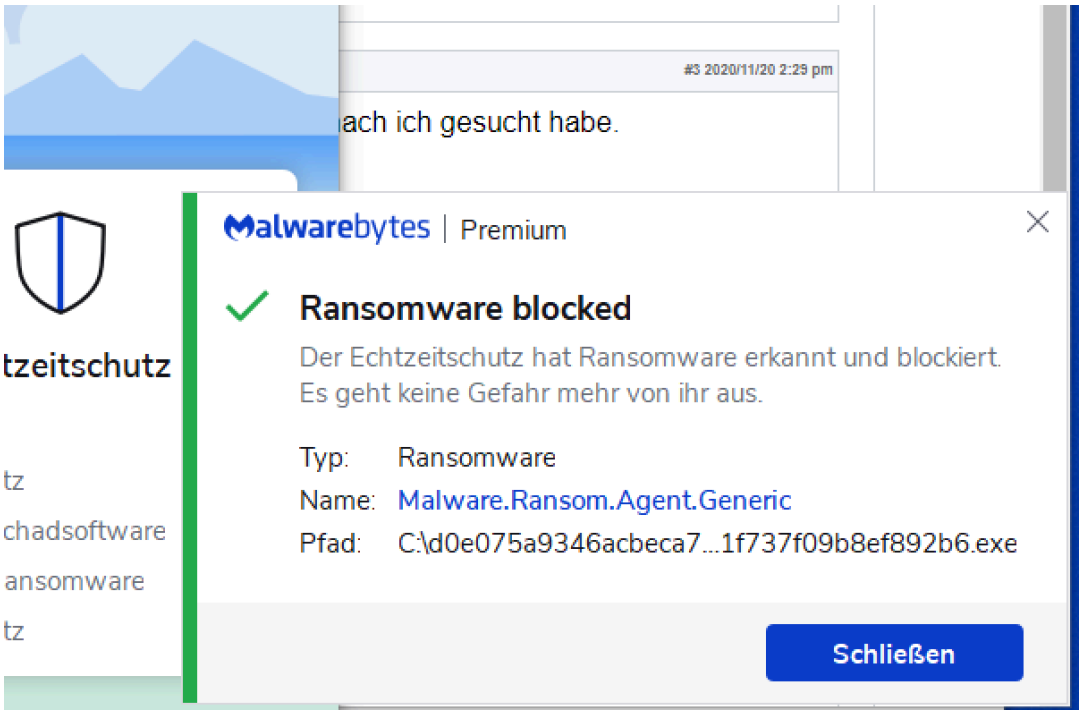
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

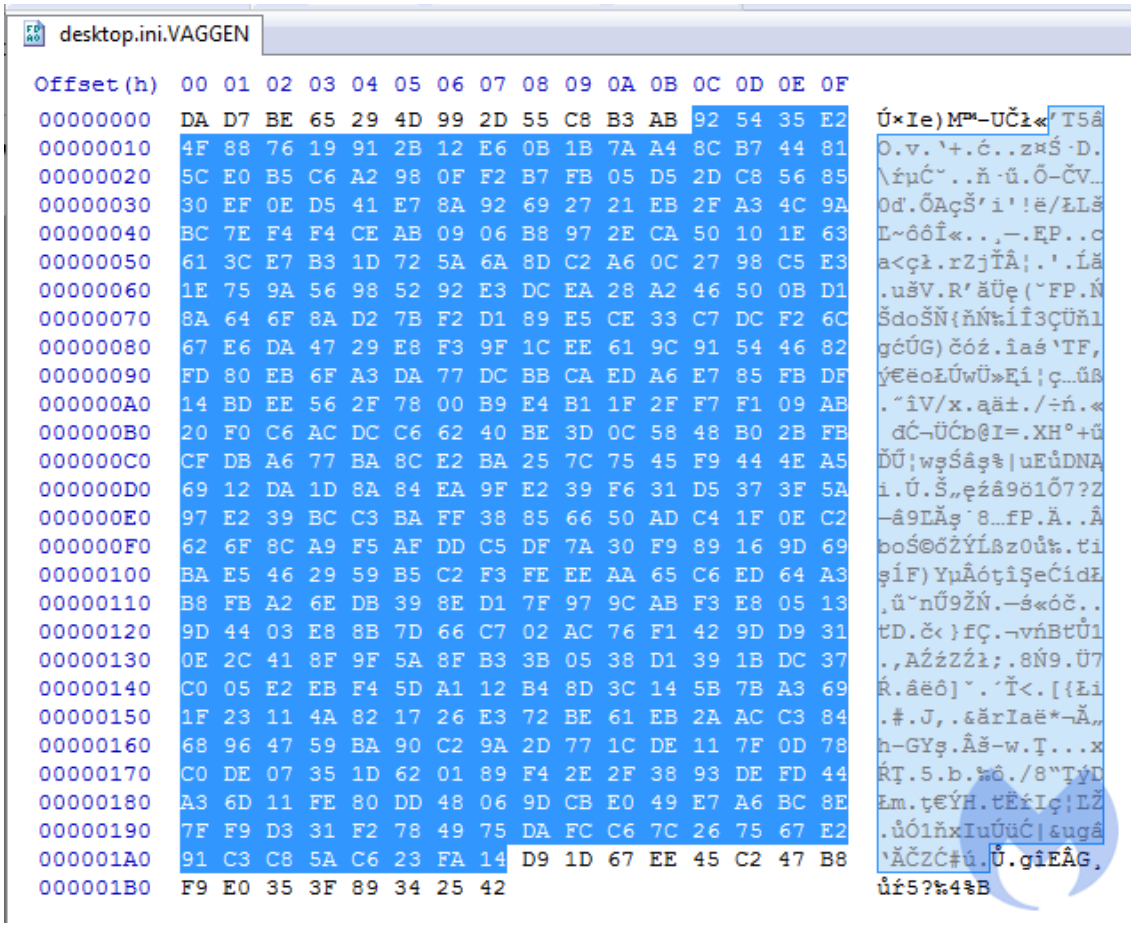
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

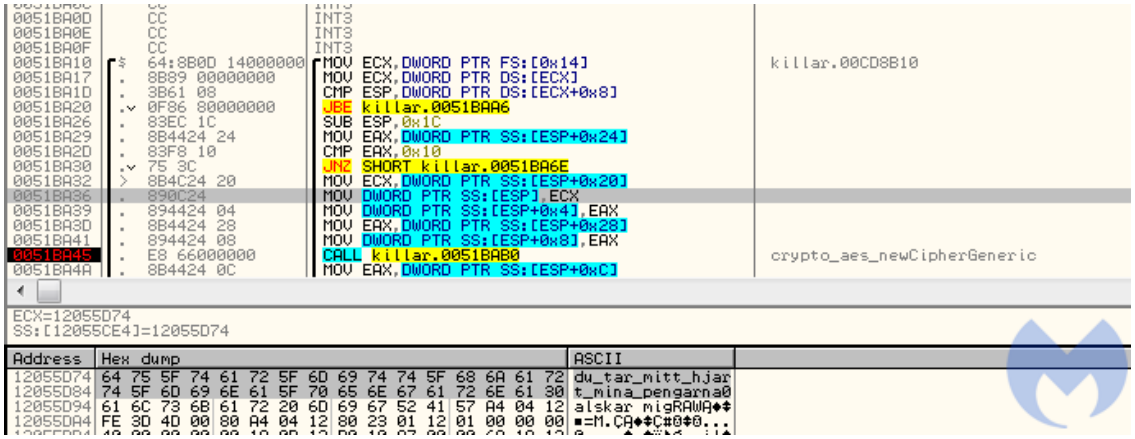


The content of the output file (with .VAGGEN extension) contains:

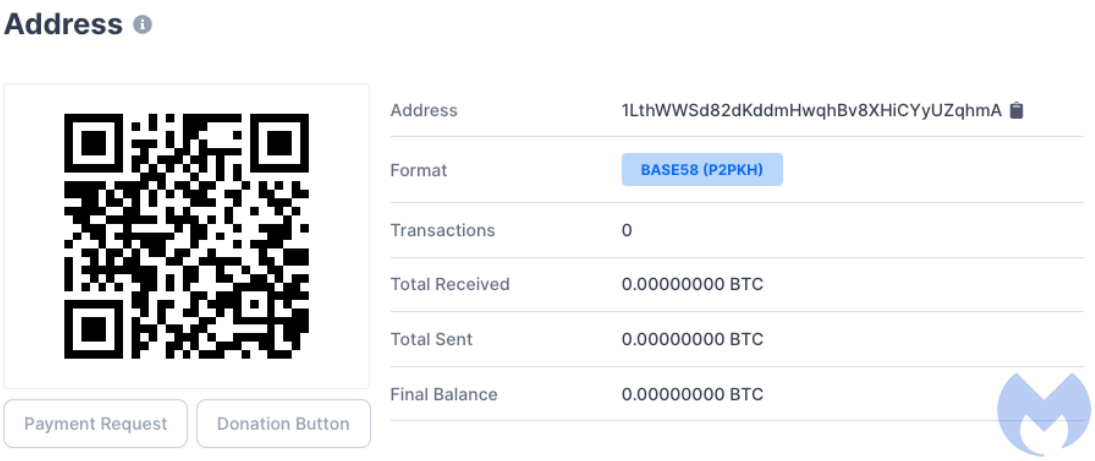
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

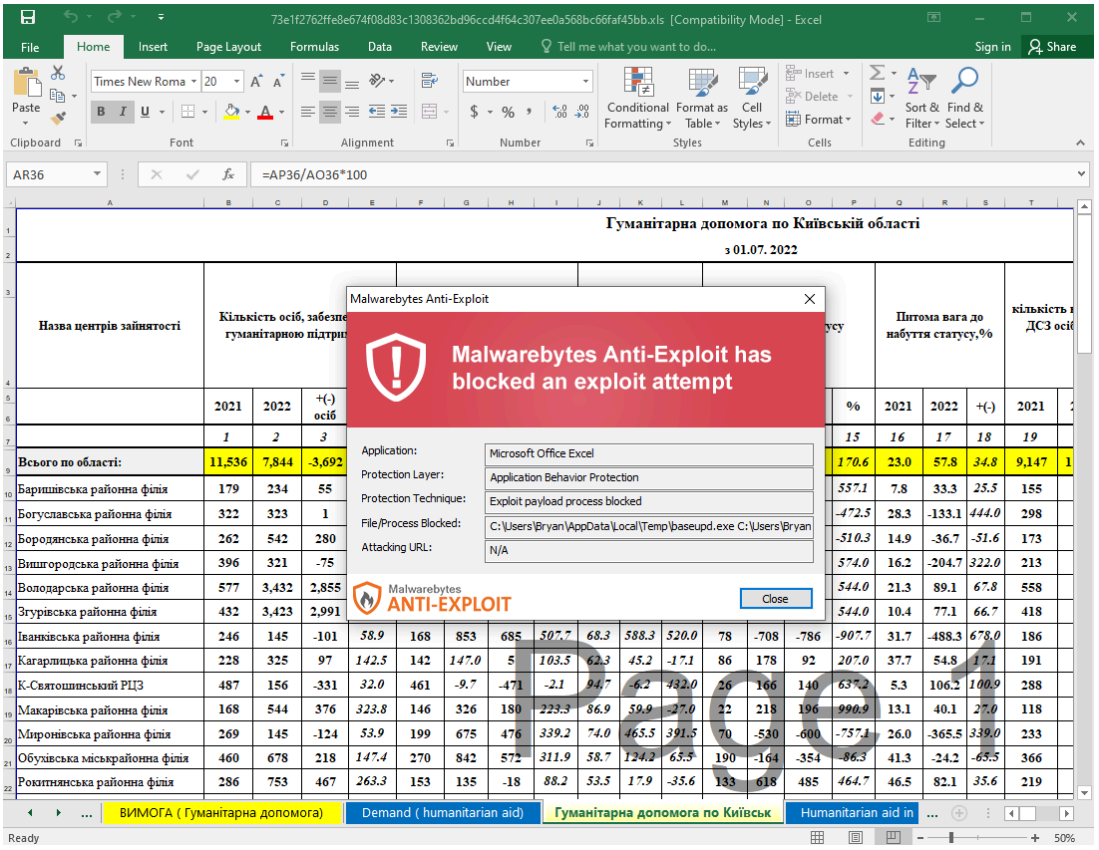
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

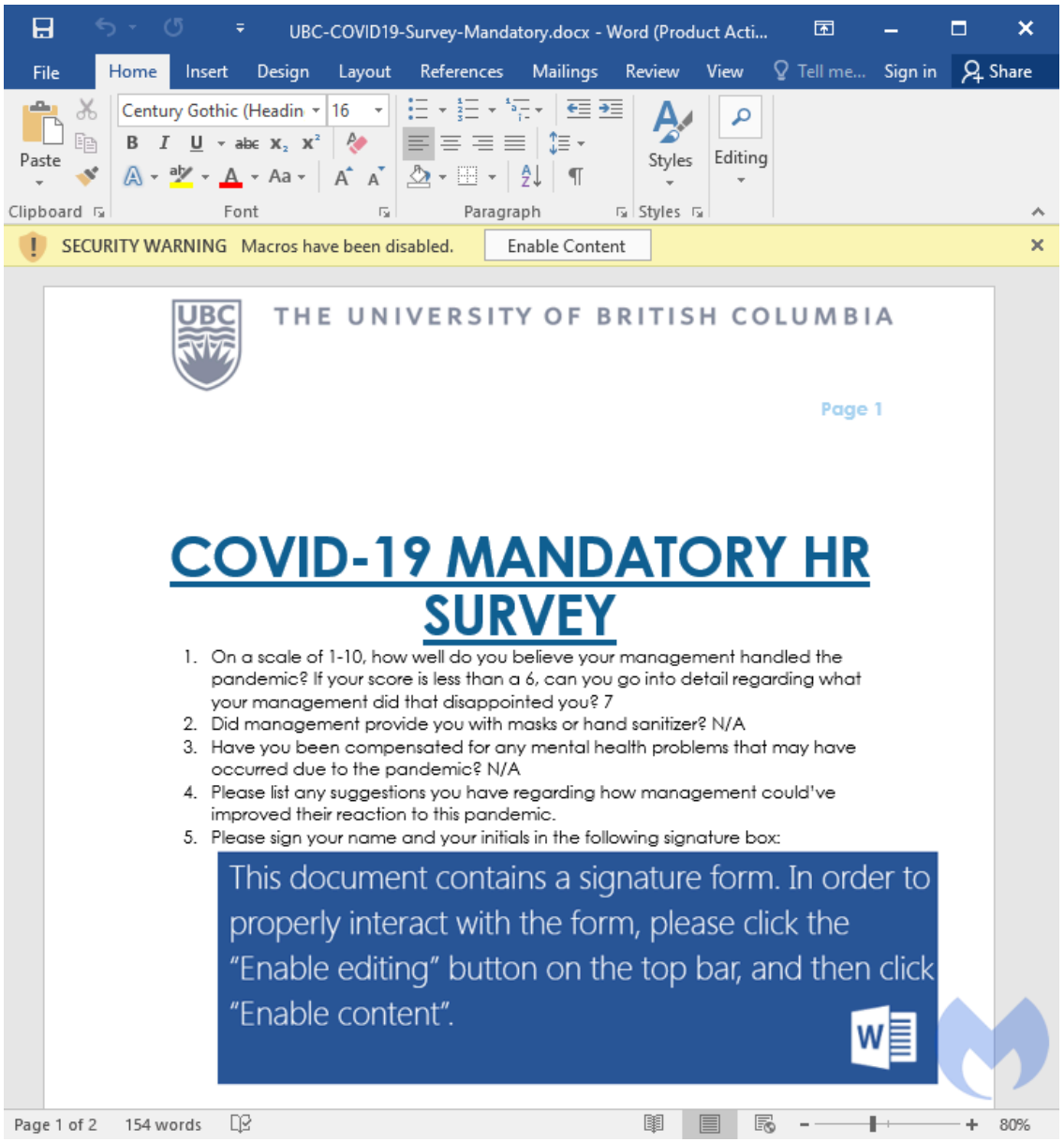
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

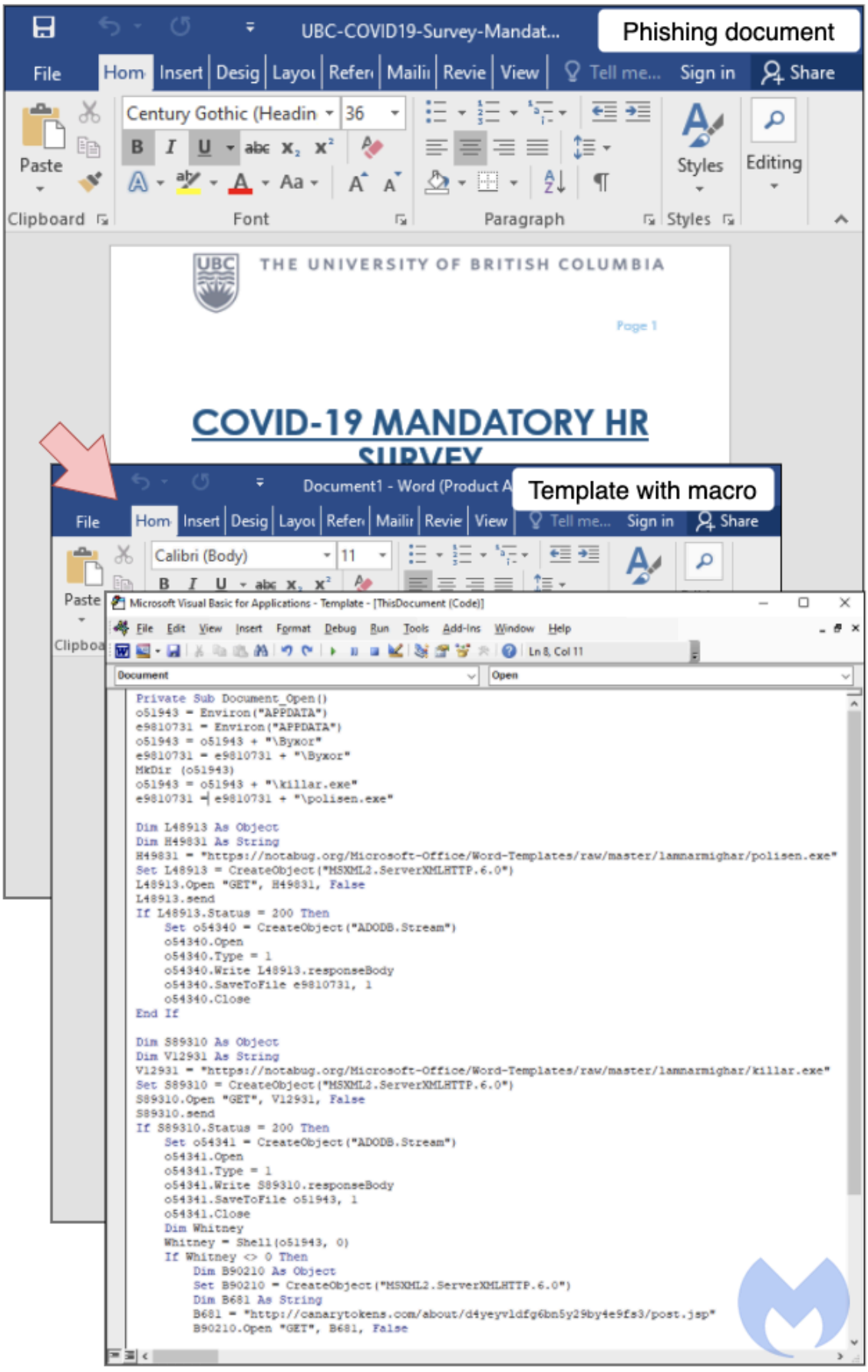
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

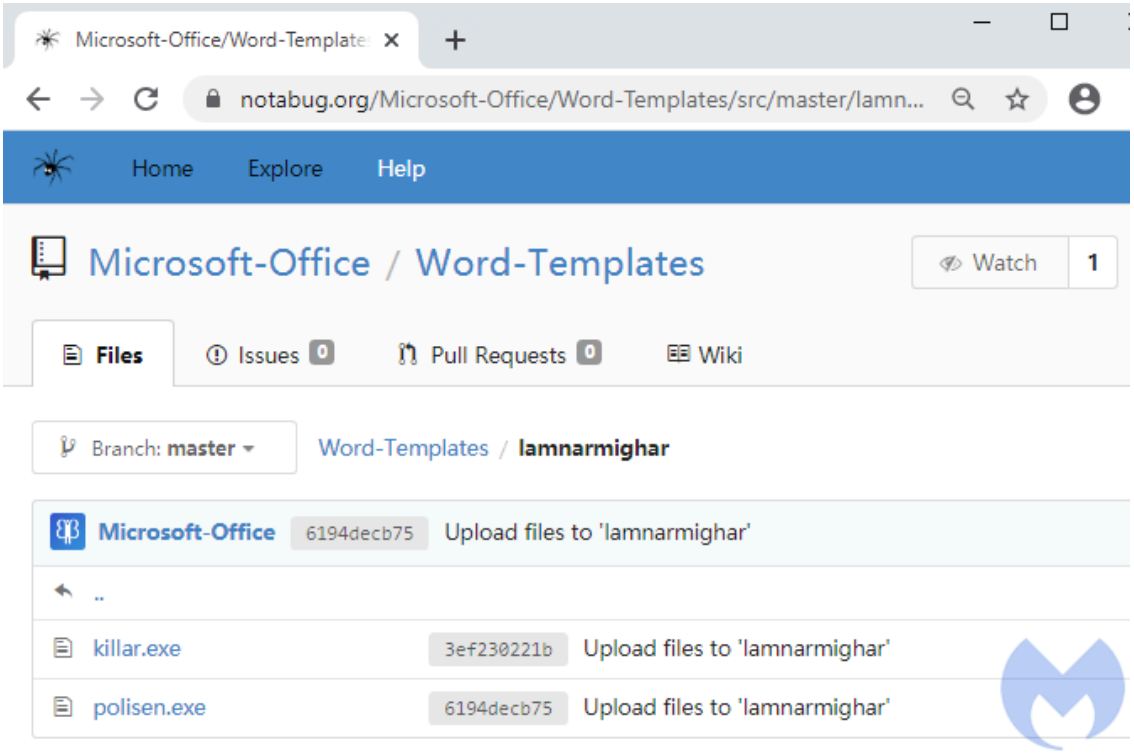
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



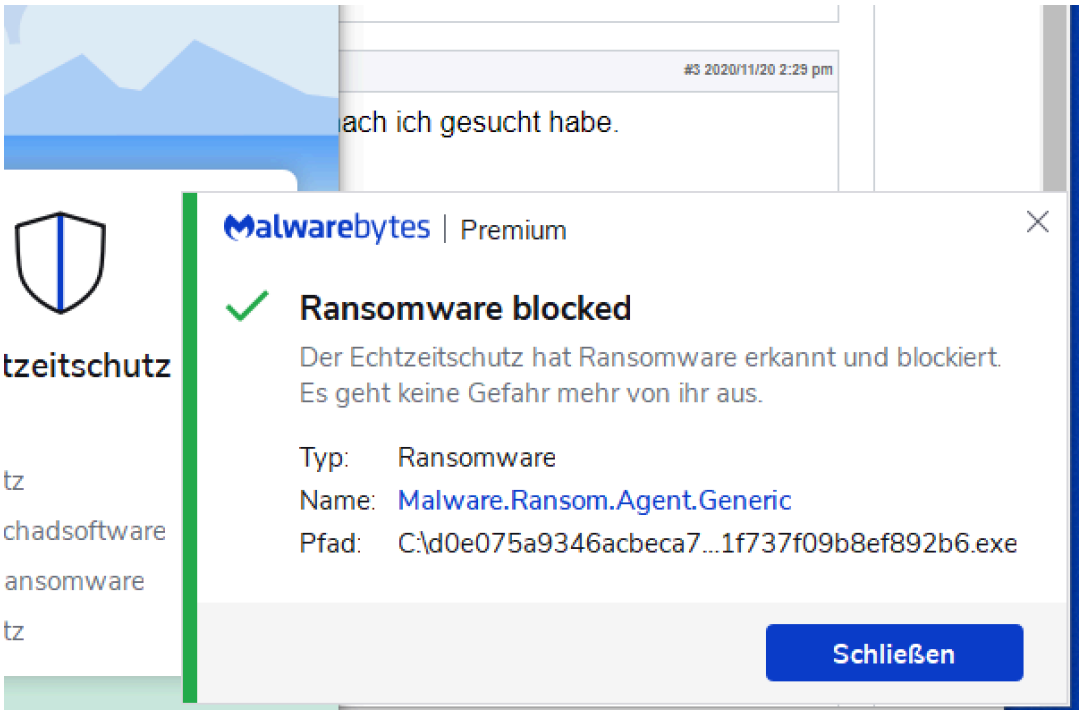
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

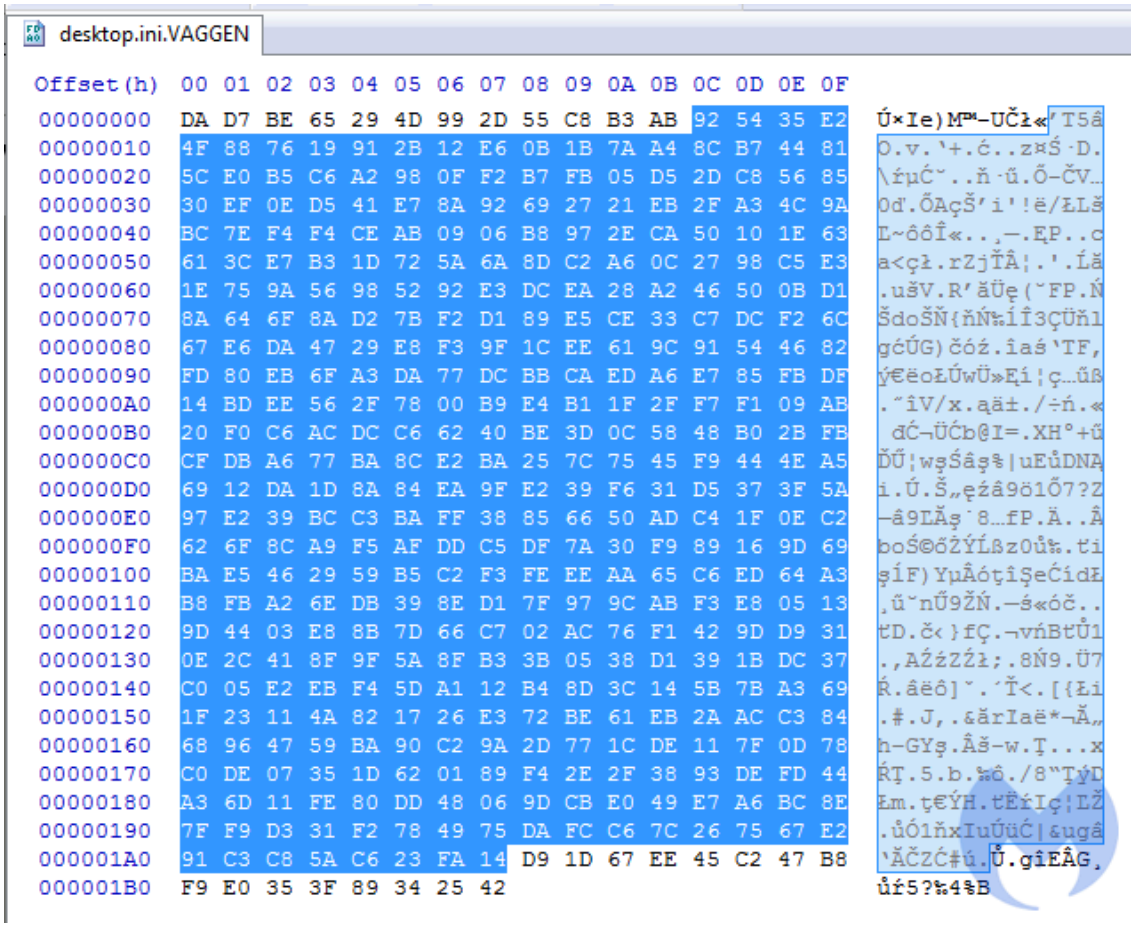
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

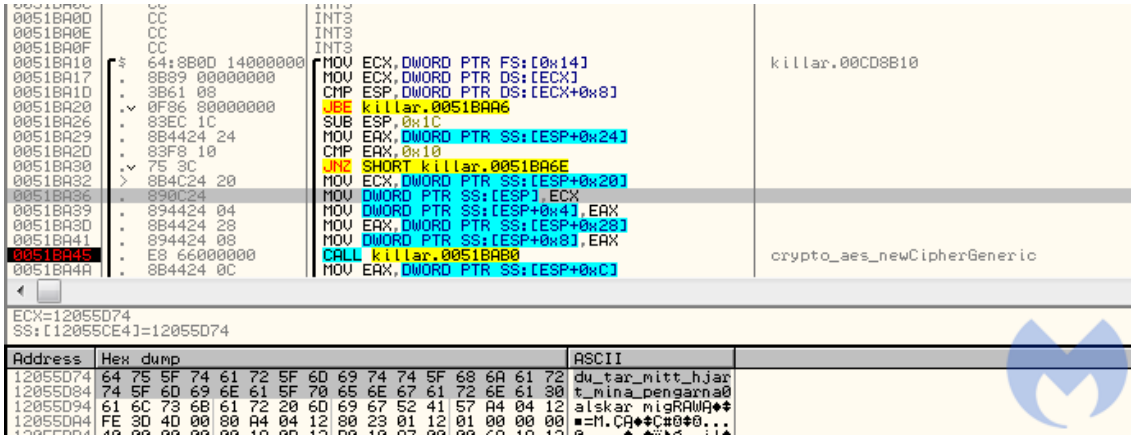


The content of the output file (with .VAGGEN extension) contains:

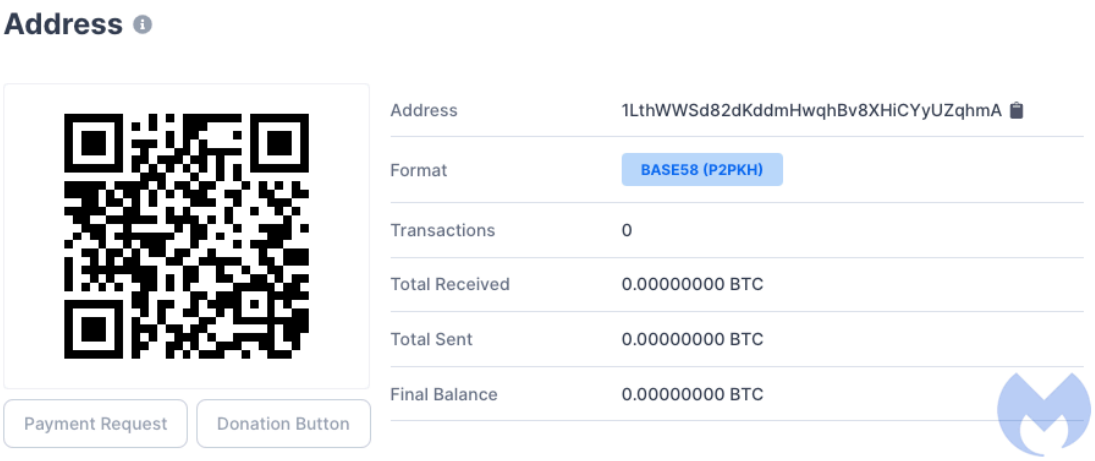
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

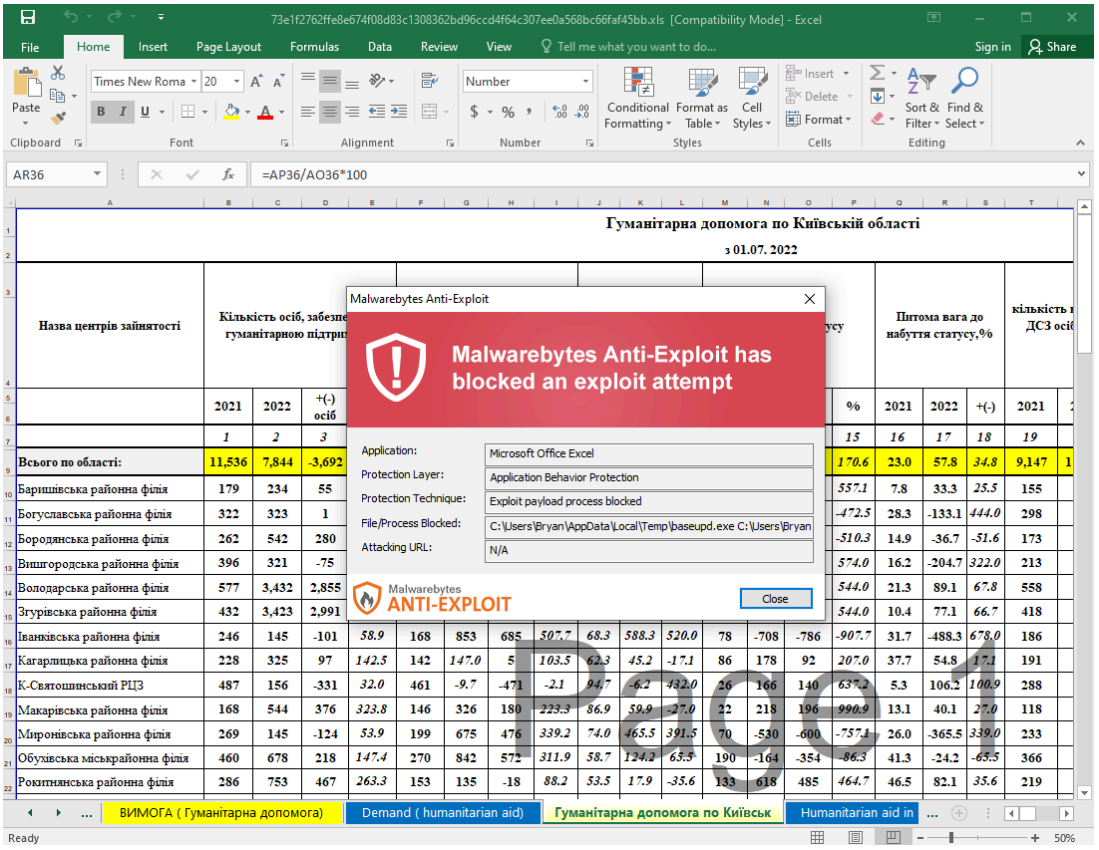
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

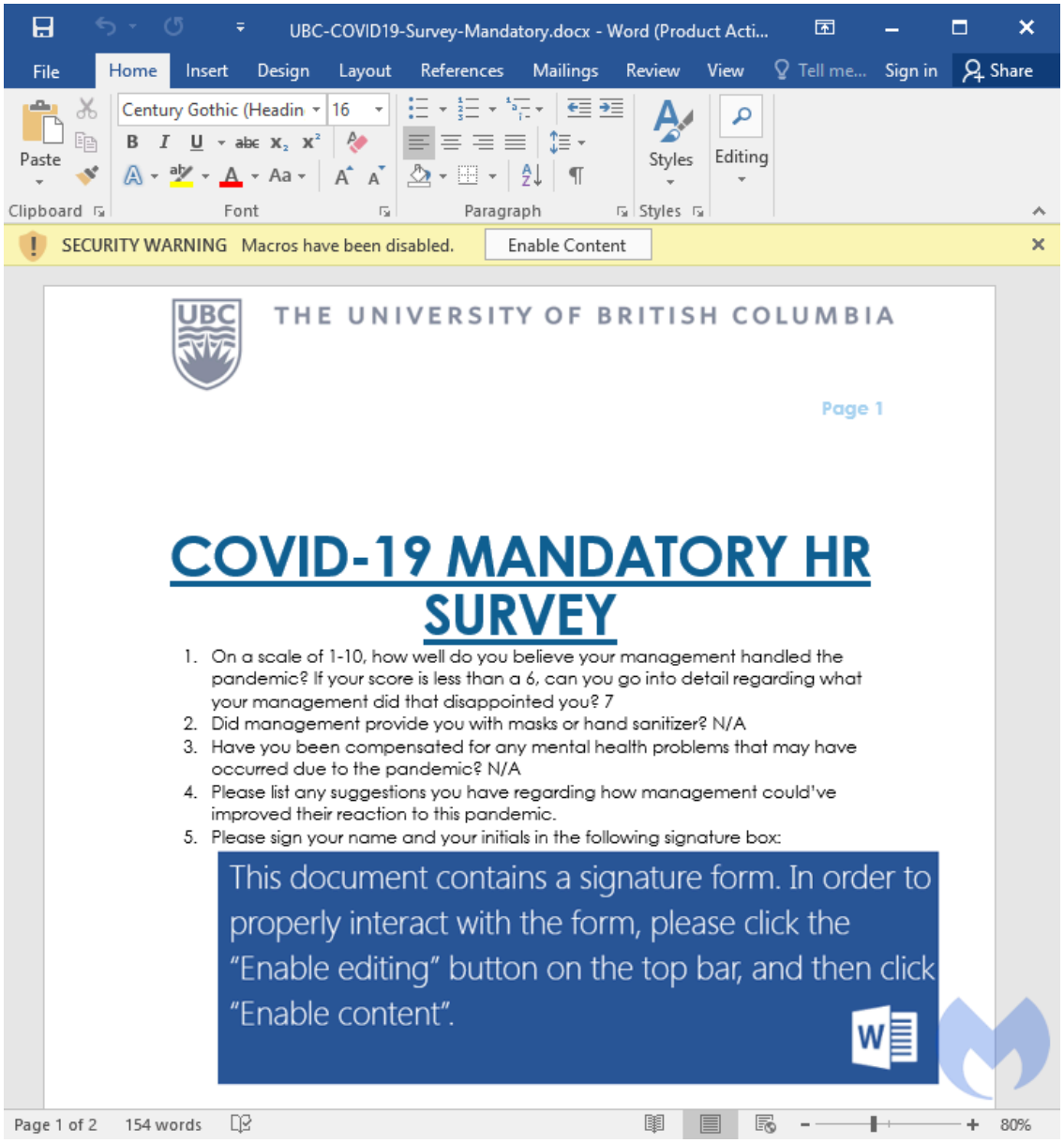
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

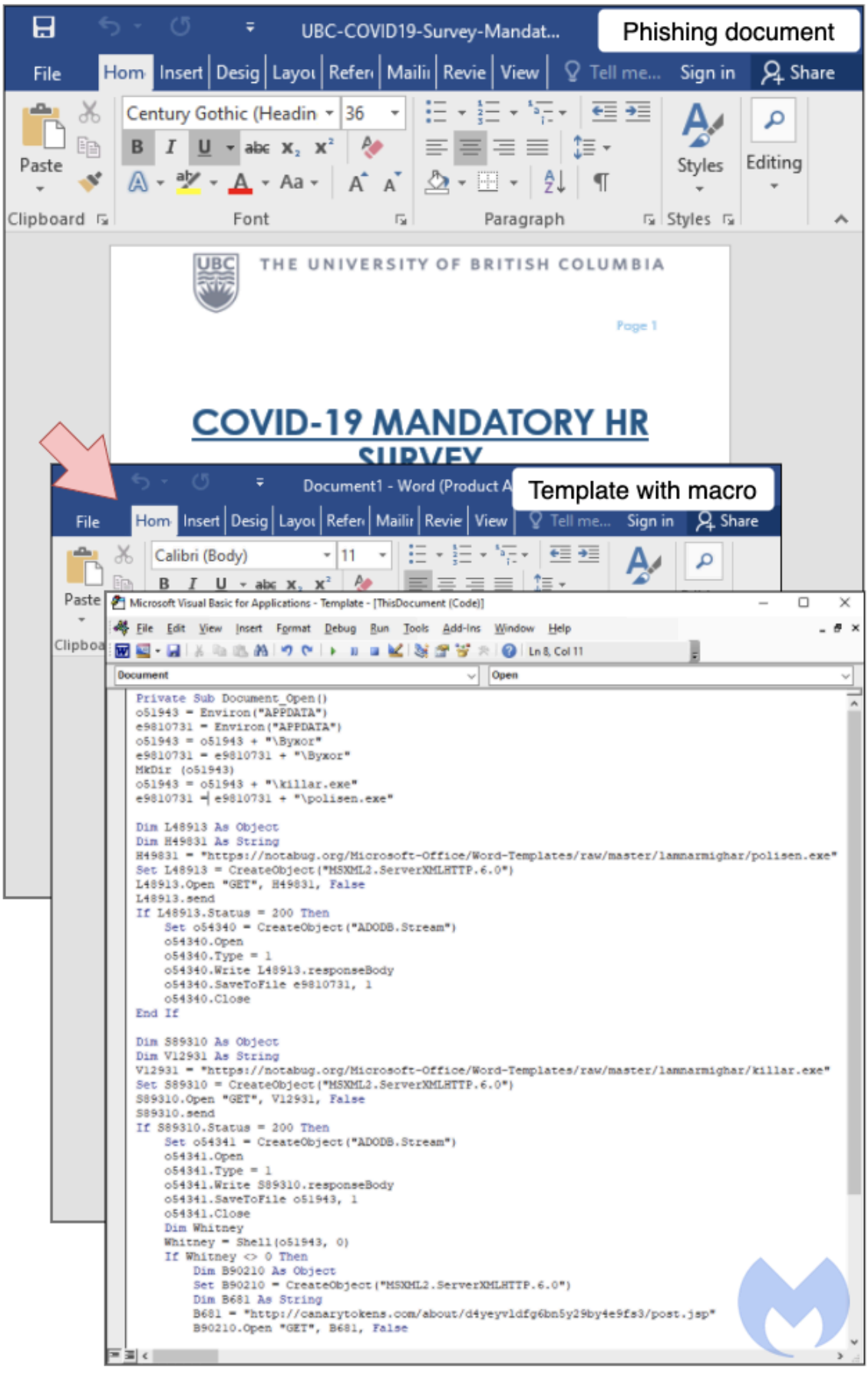
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

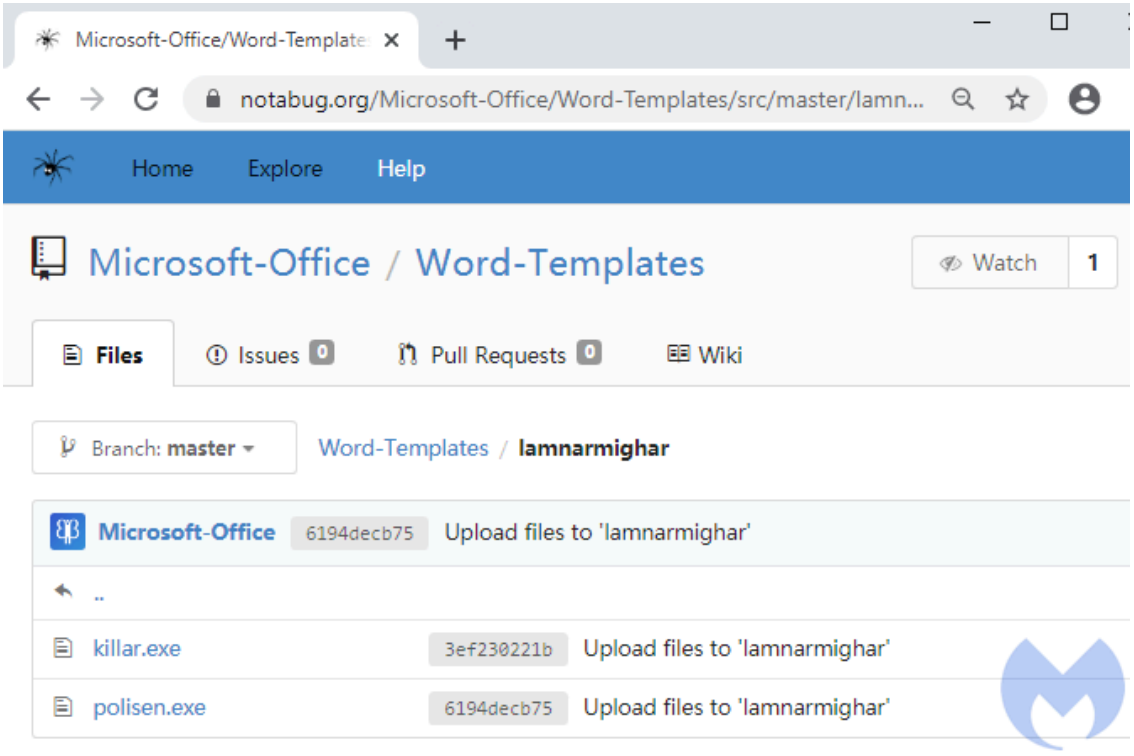
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



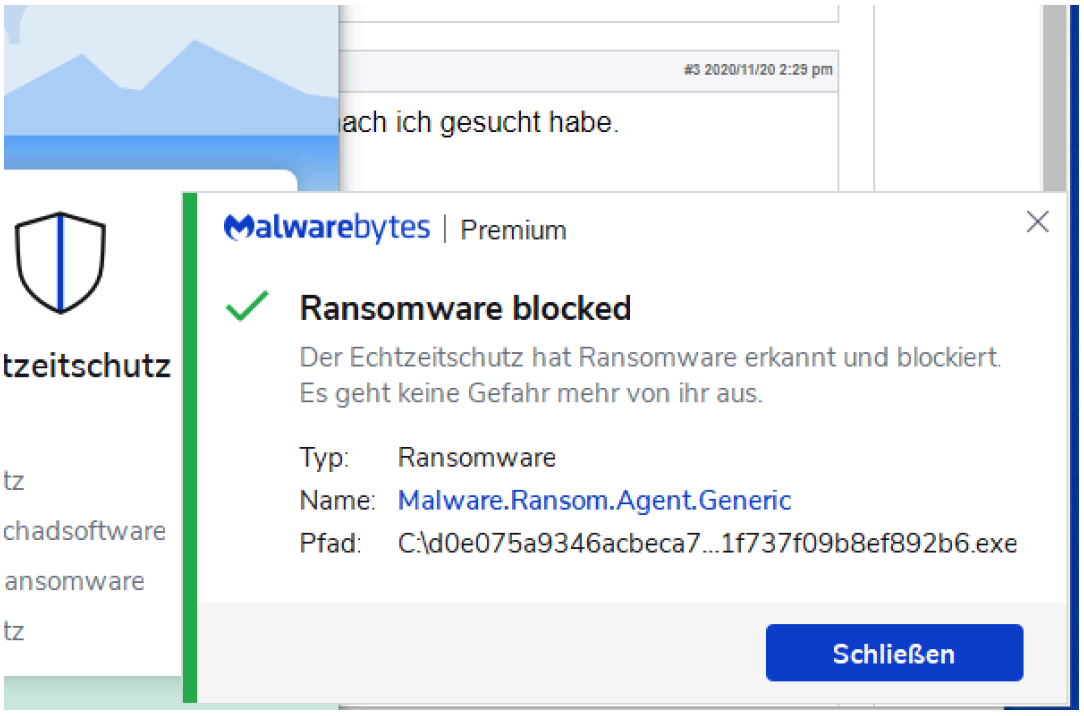
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

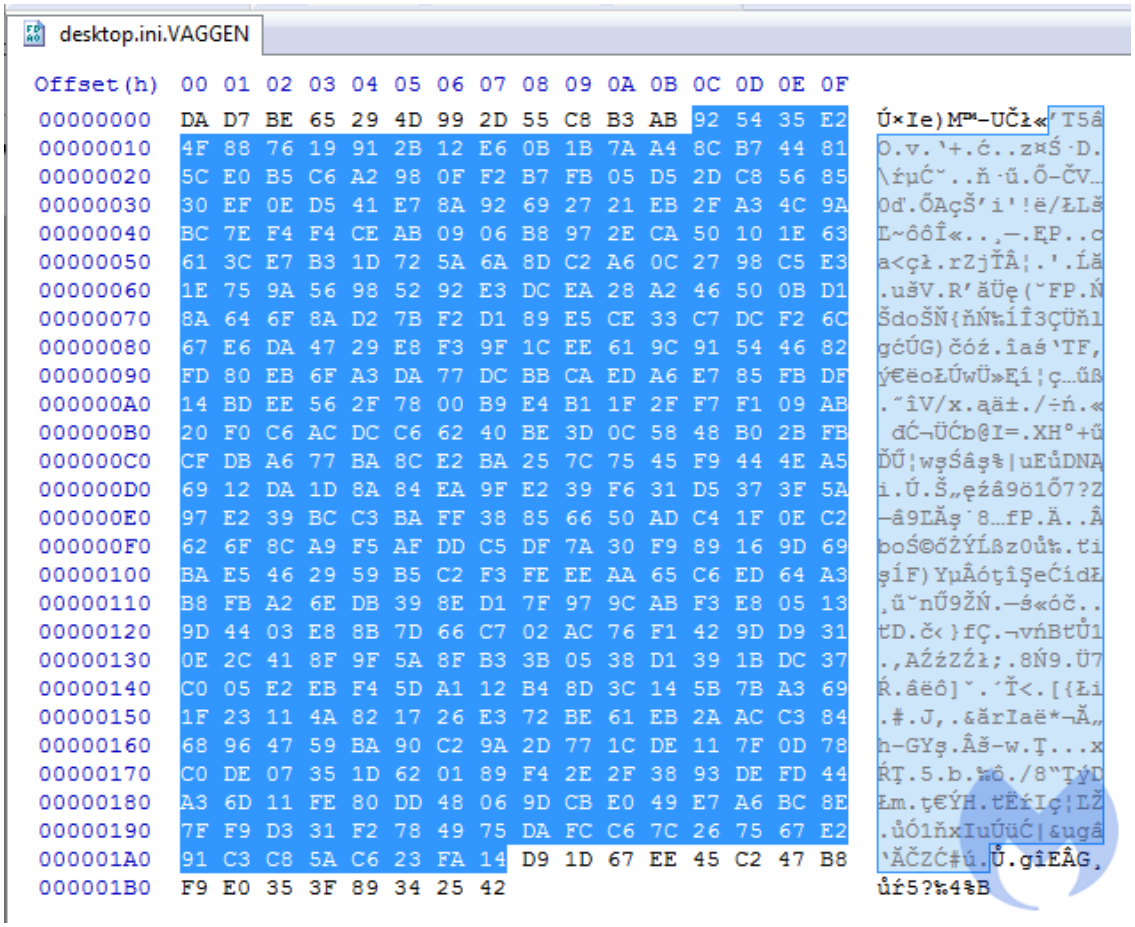
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

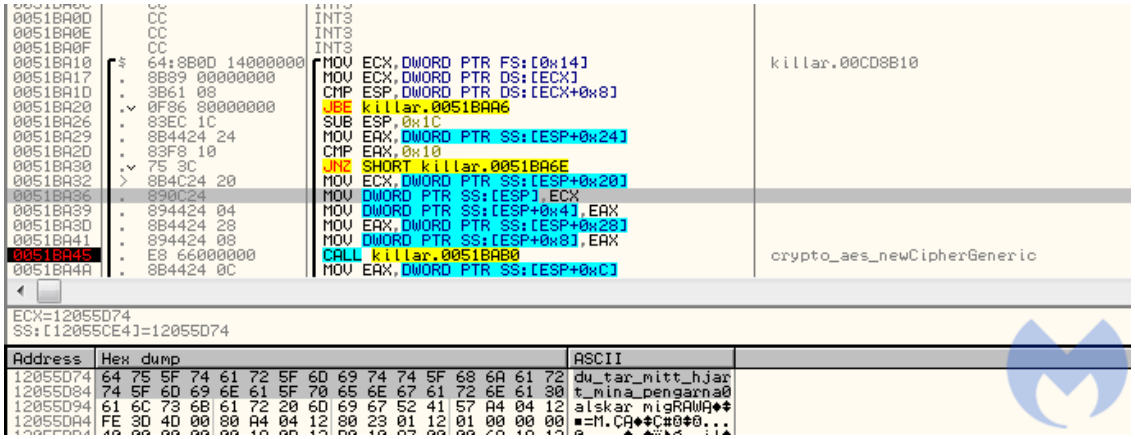


The content of the output file (with .VAGGEN extension) contains:

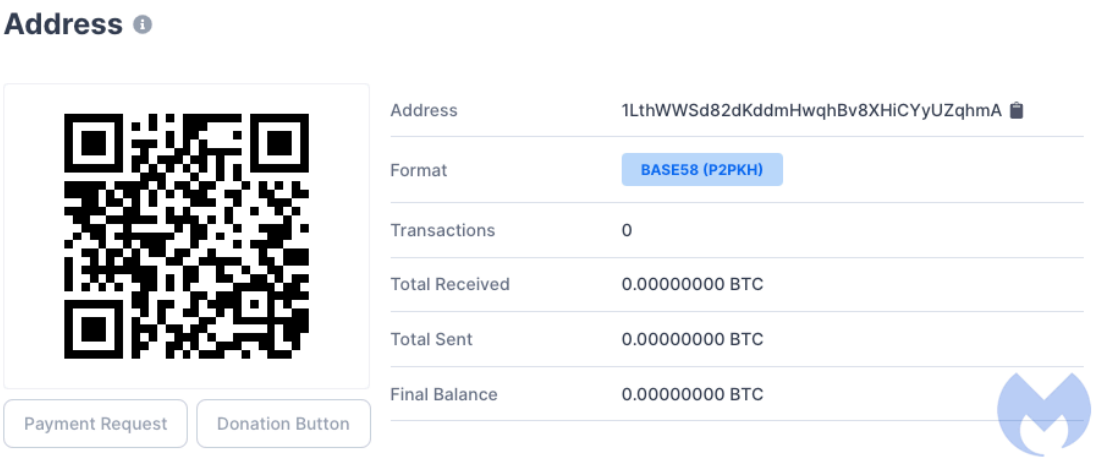
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

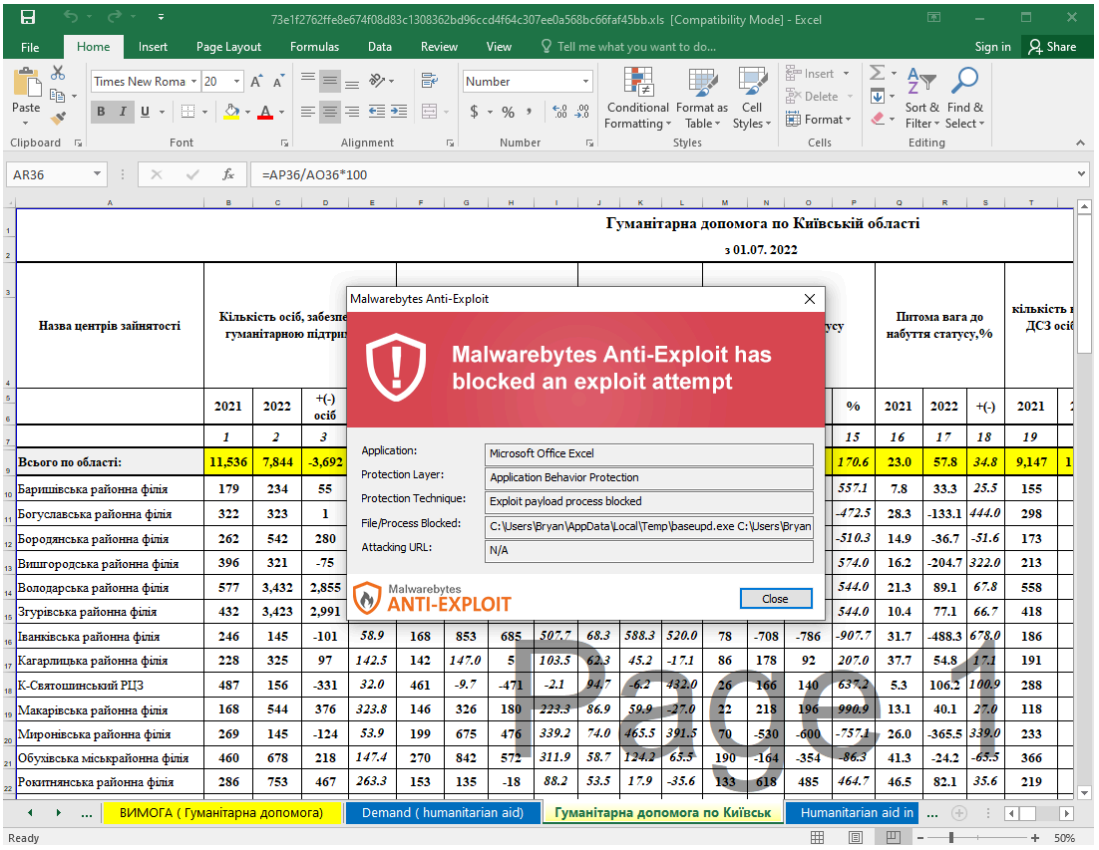
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

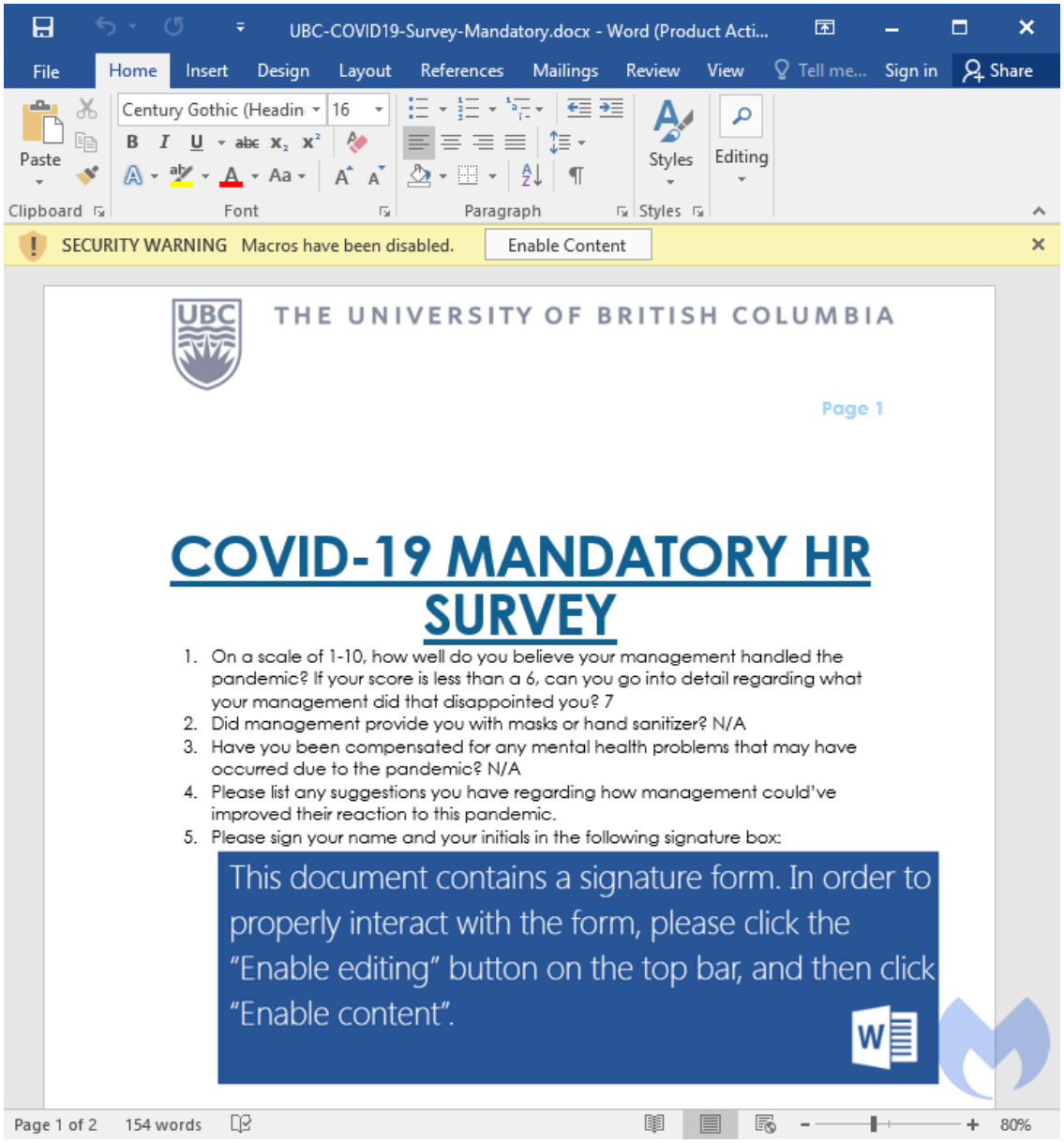
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

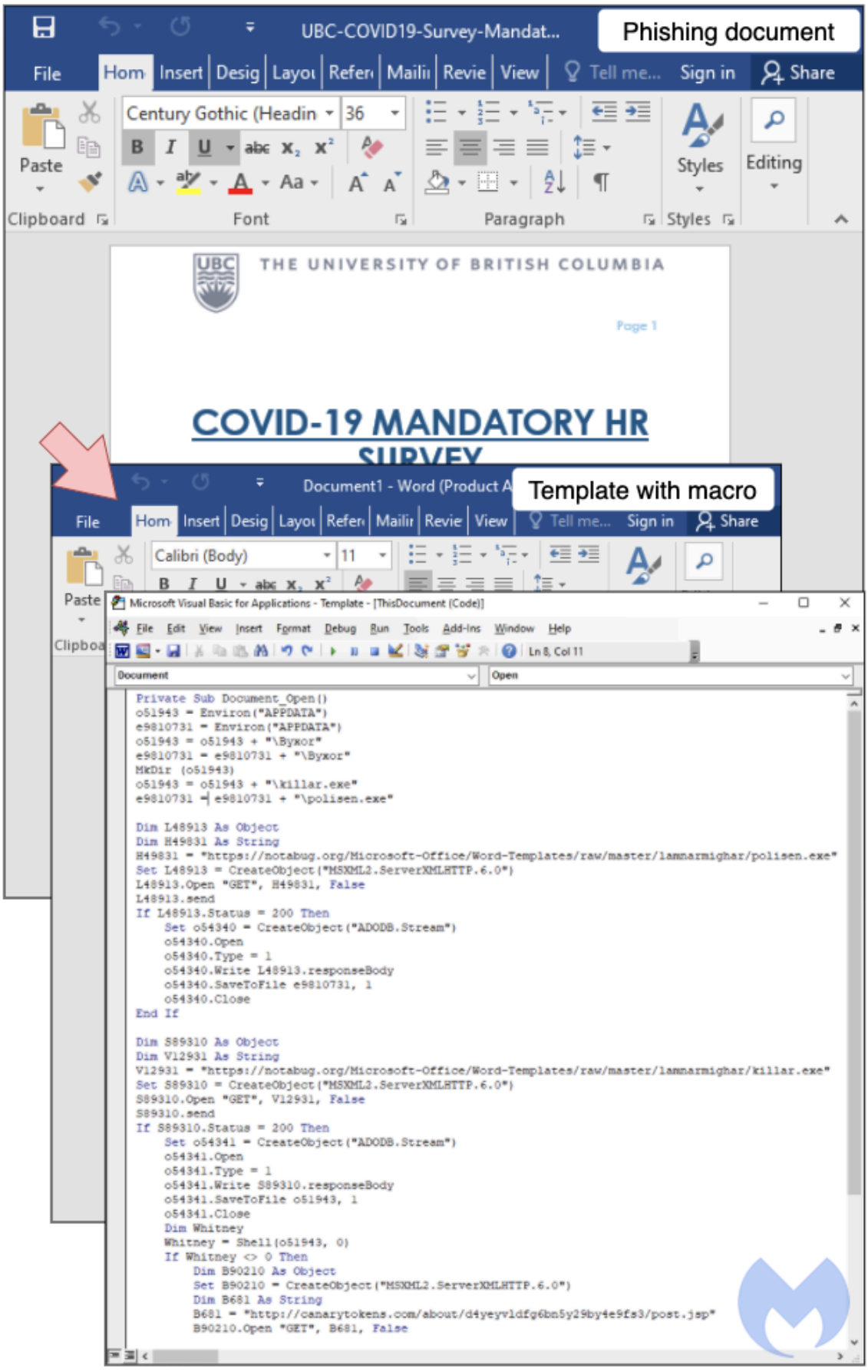
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

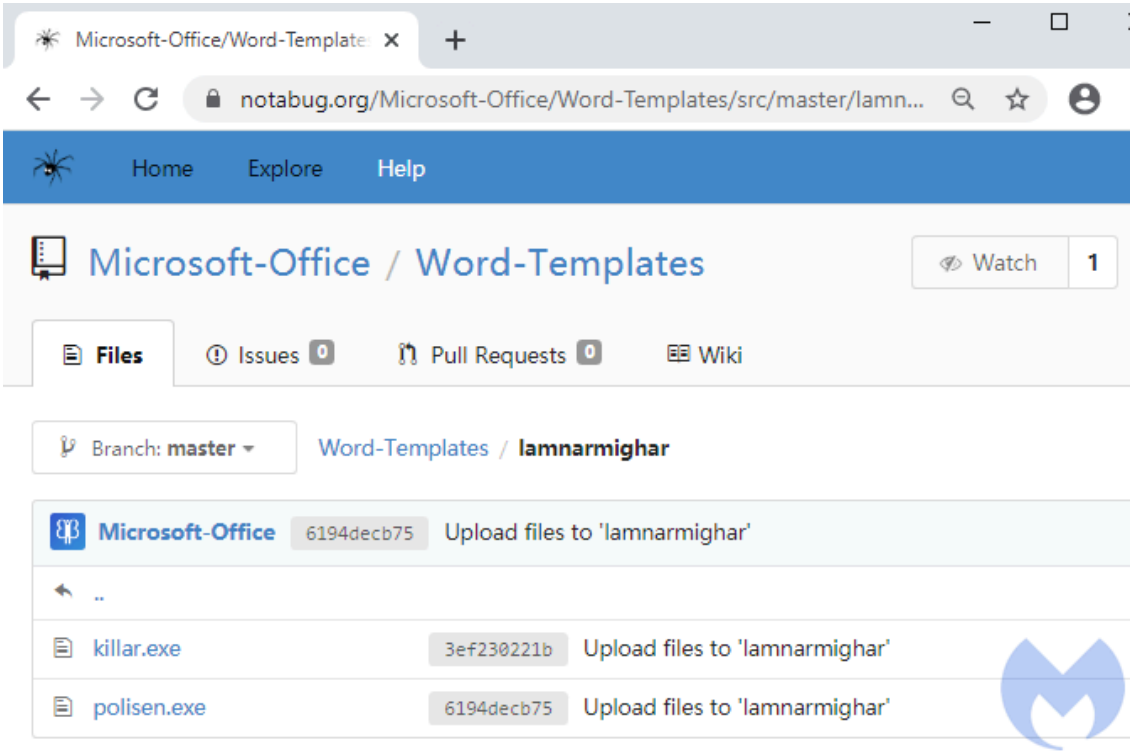
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



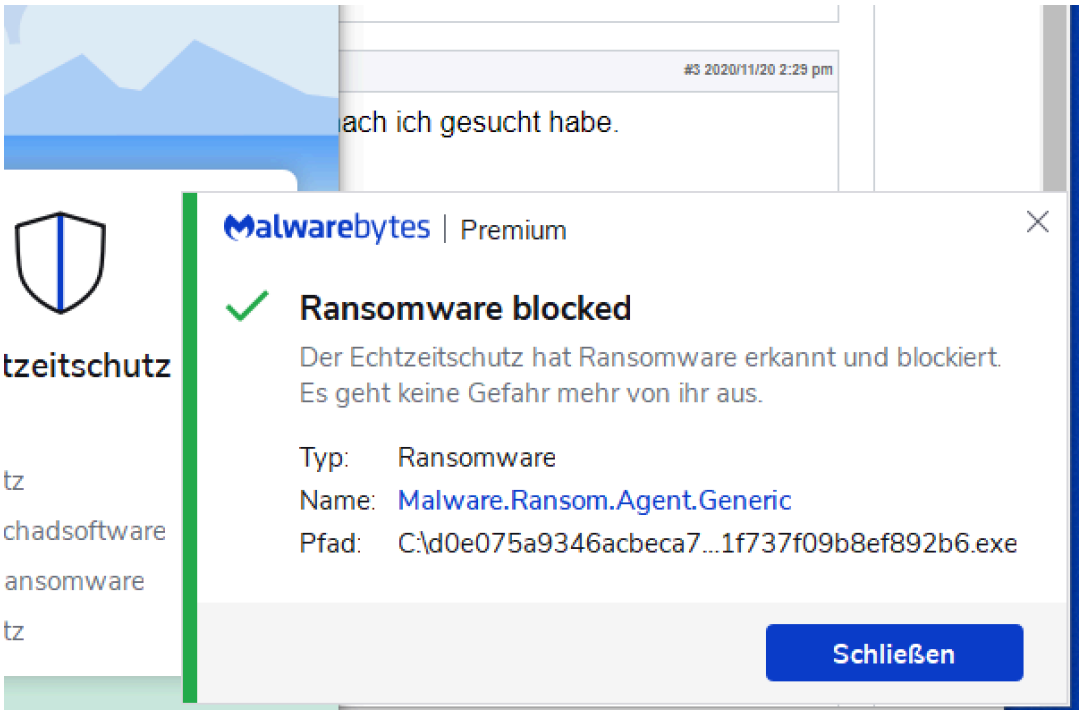
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

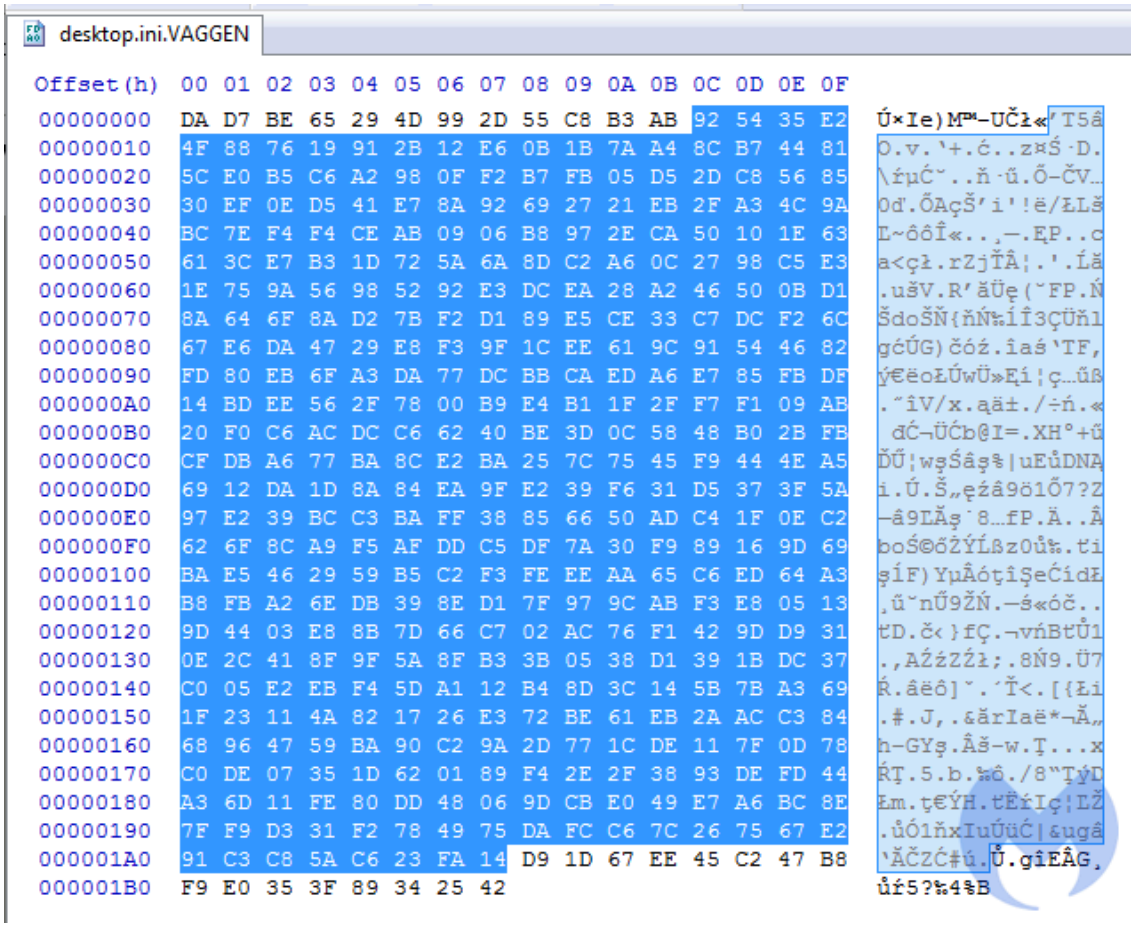
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

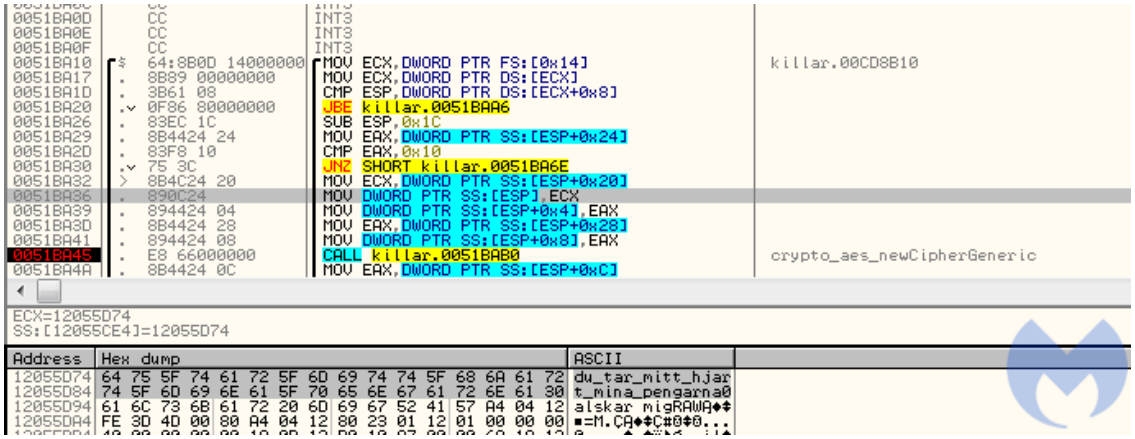


The content of the output file (with .VAGGEN extension) contains:

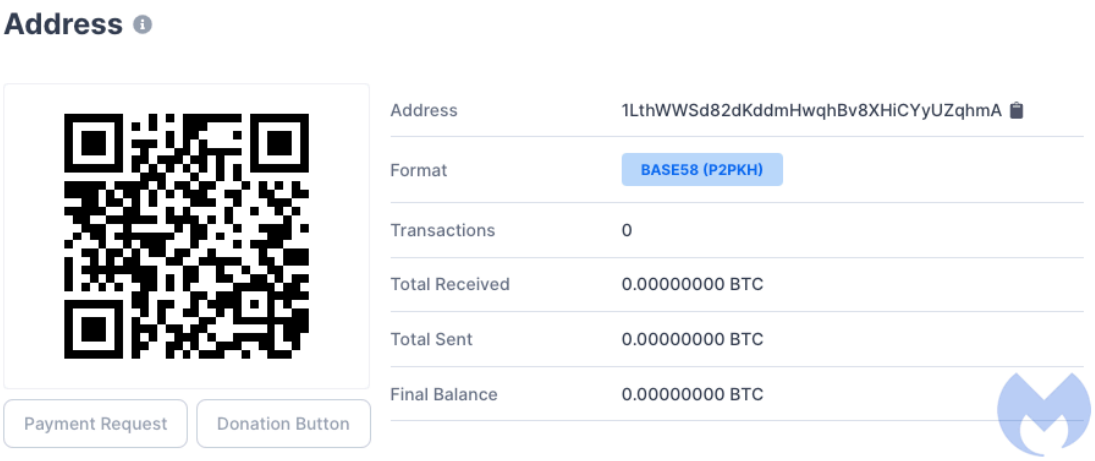
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

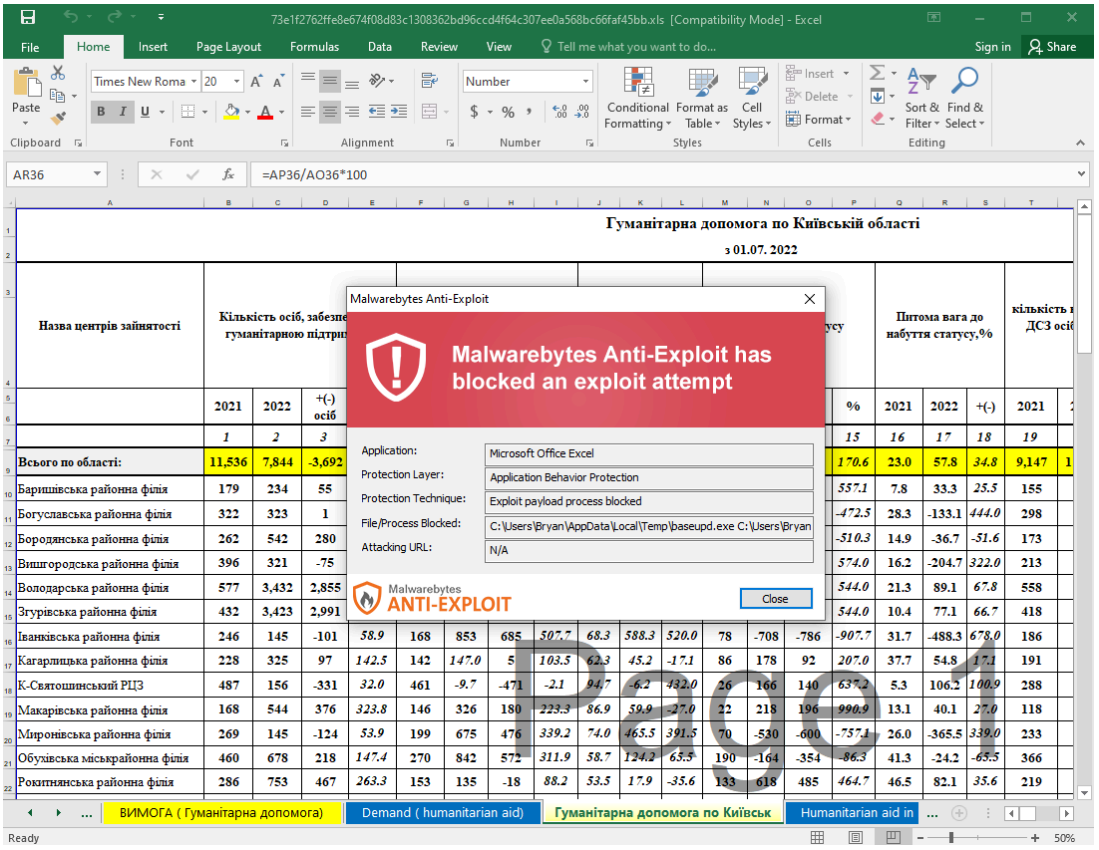
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

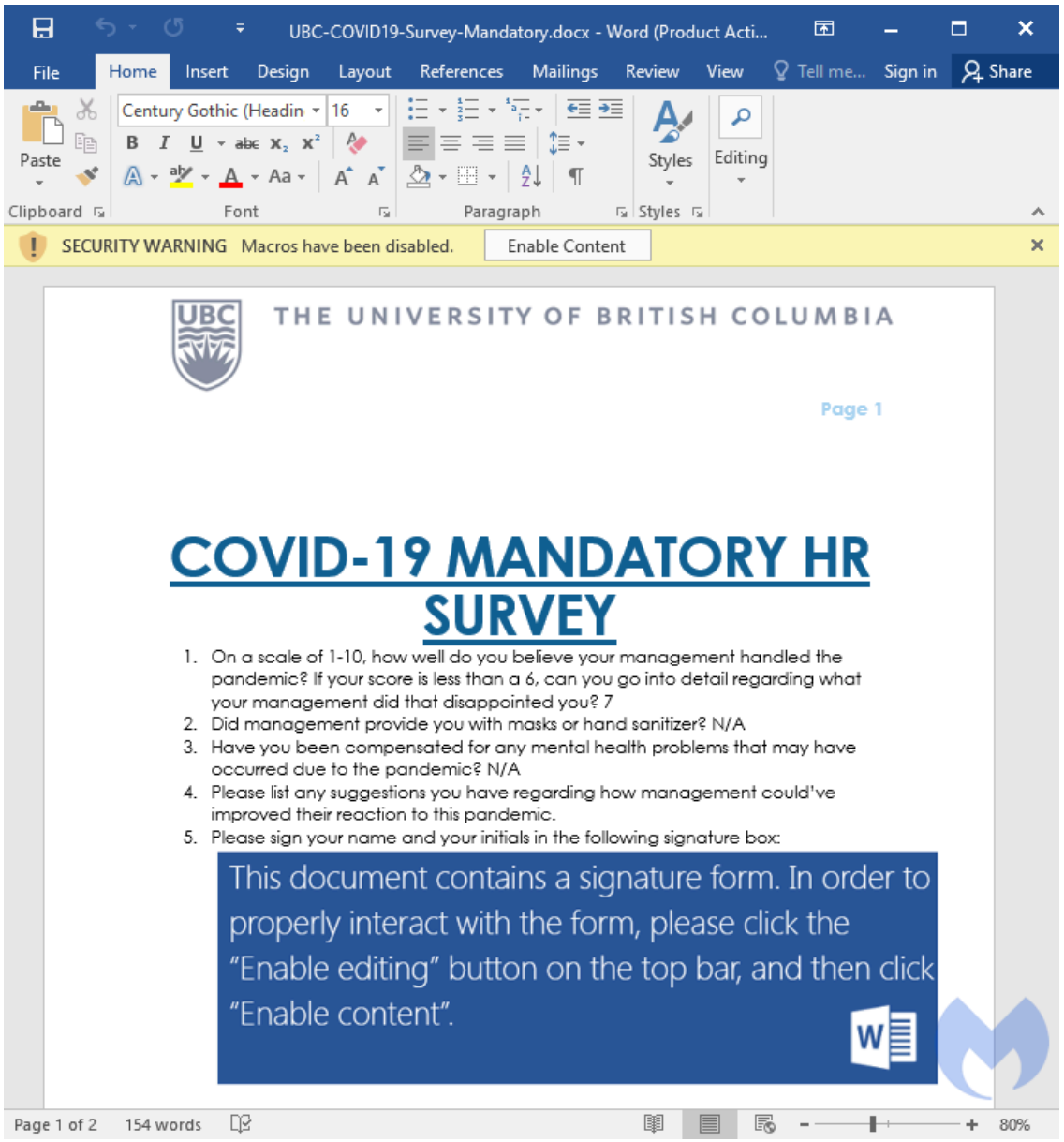
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

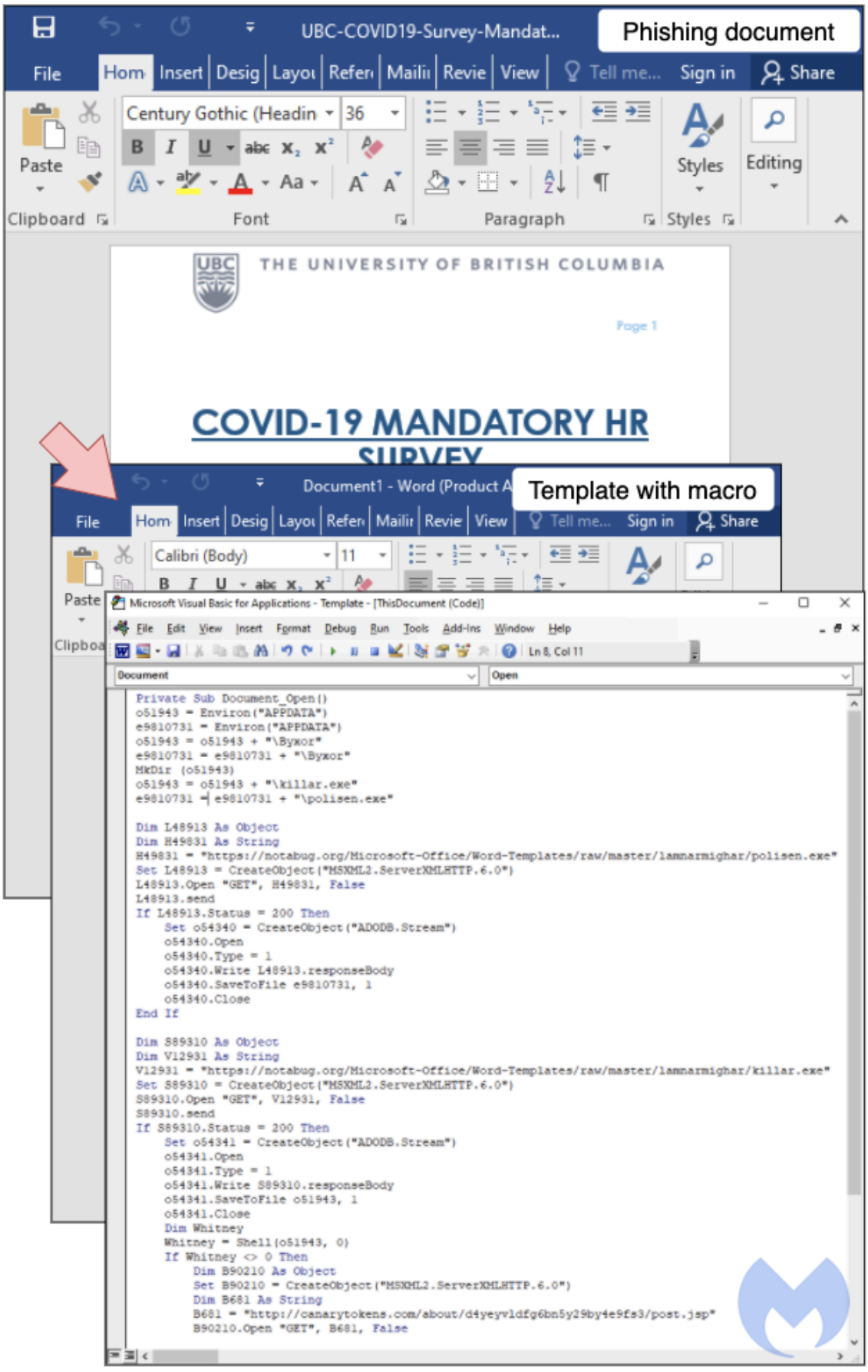
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

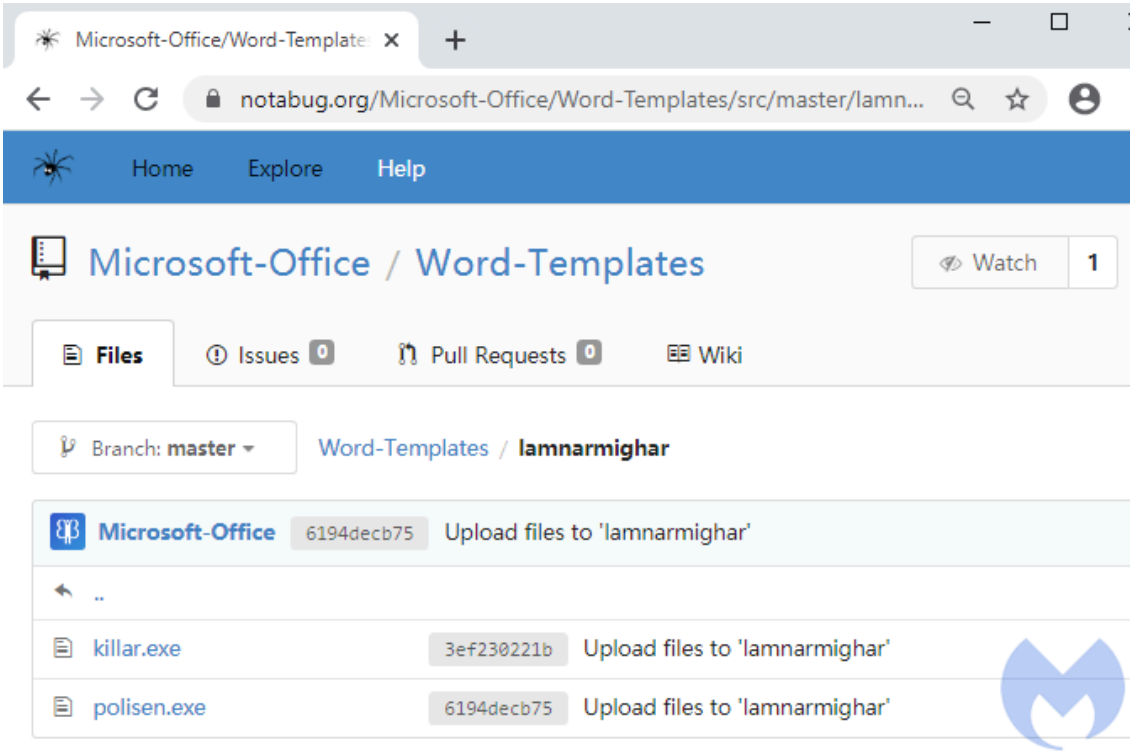
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



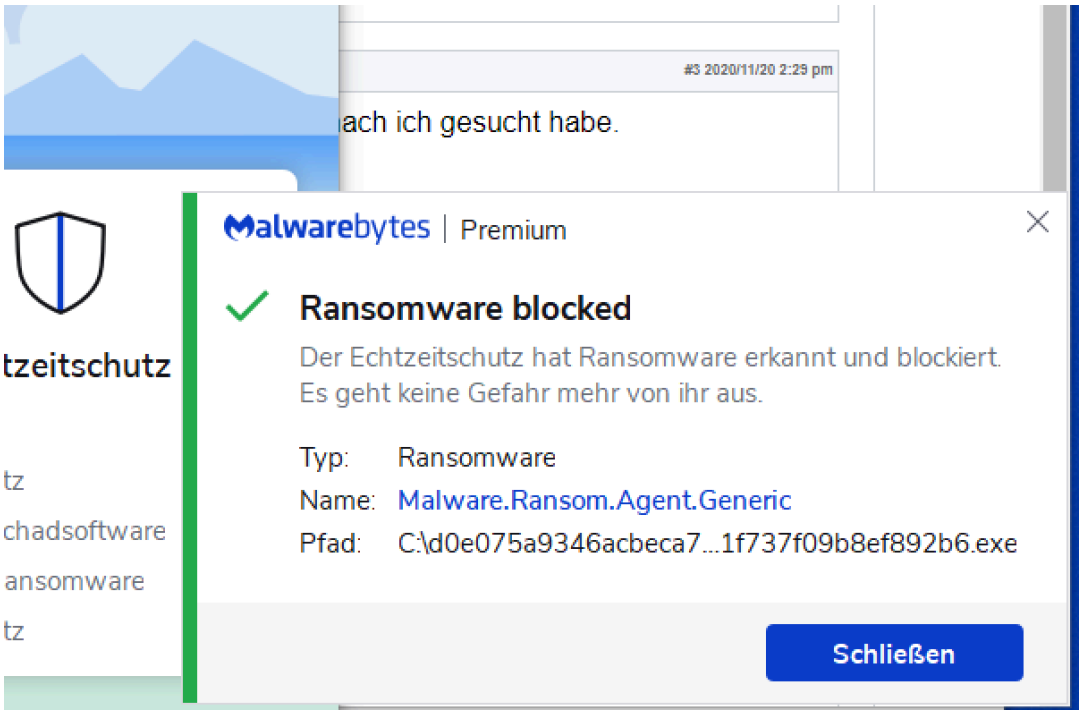
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

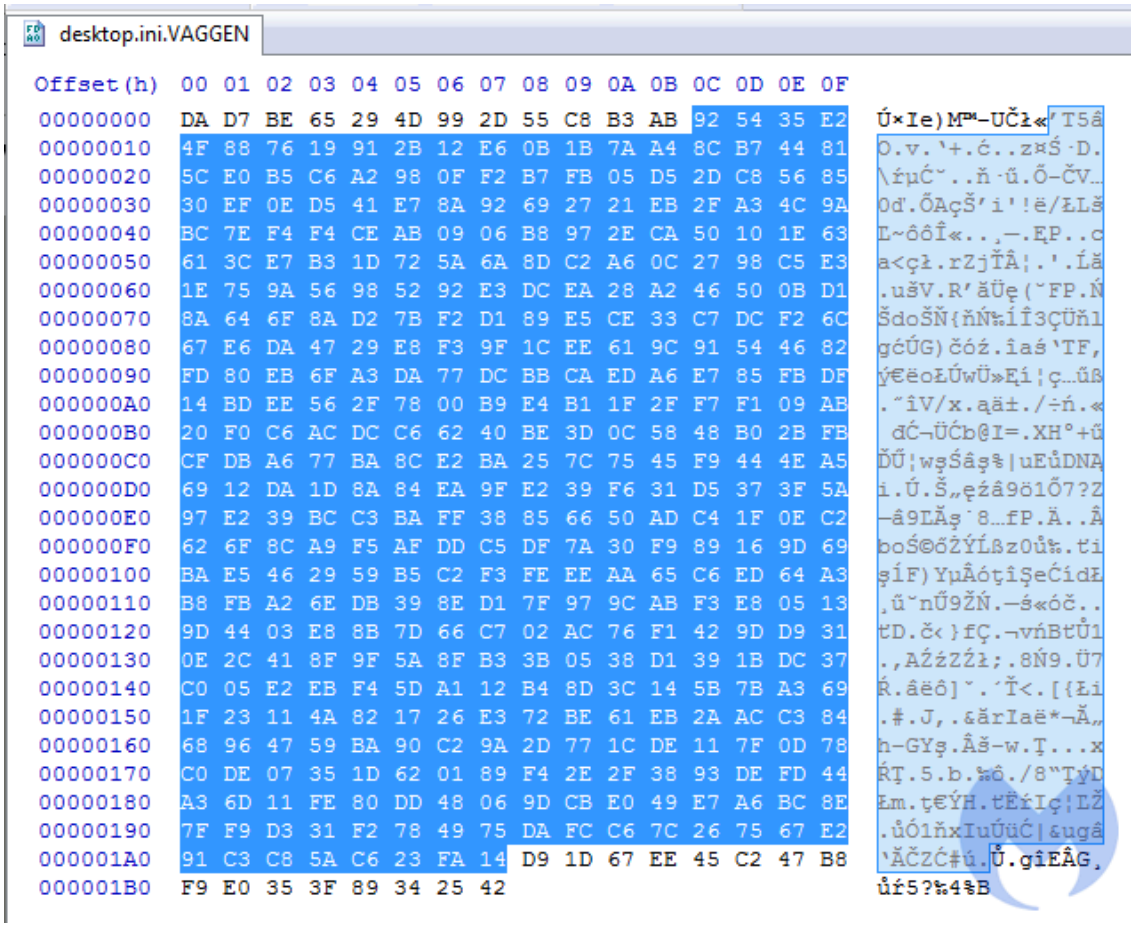
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



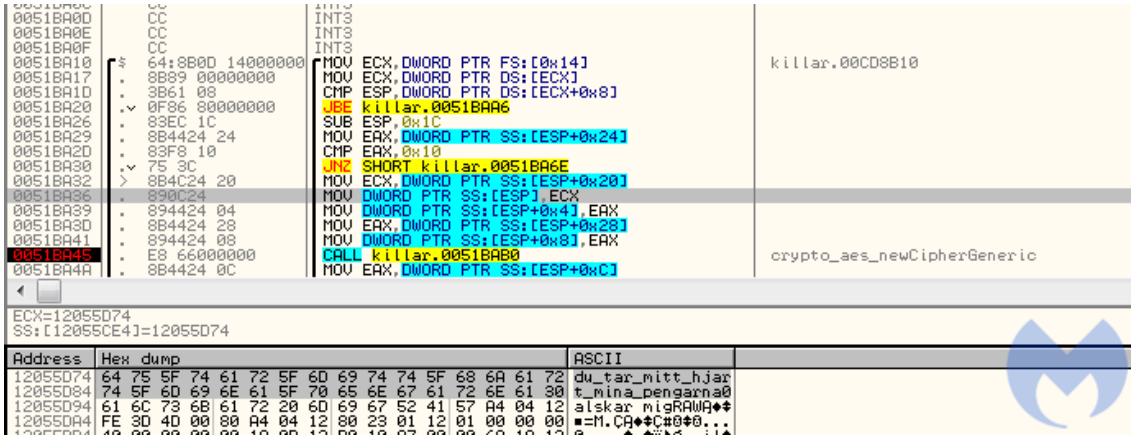
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

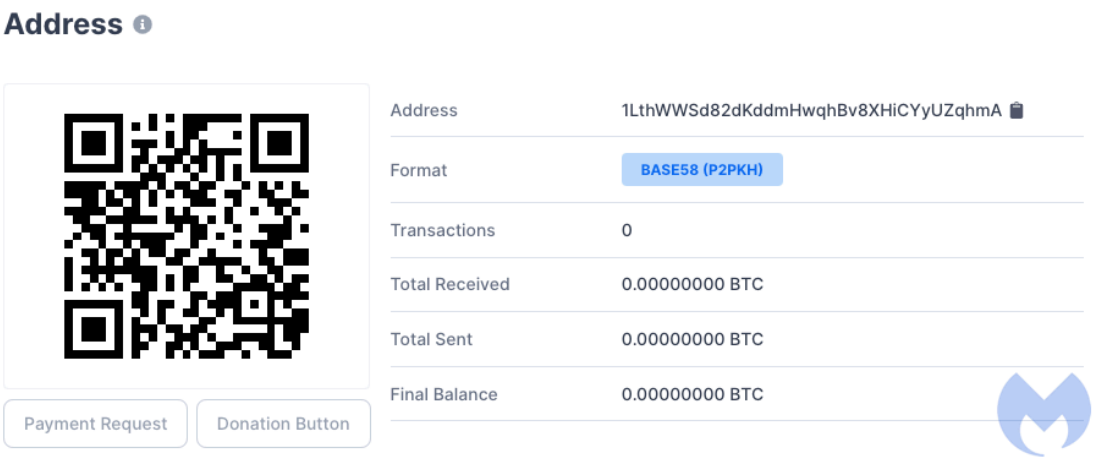
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

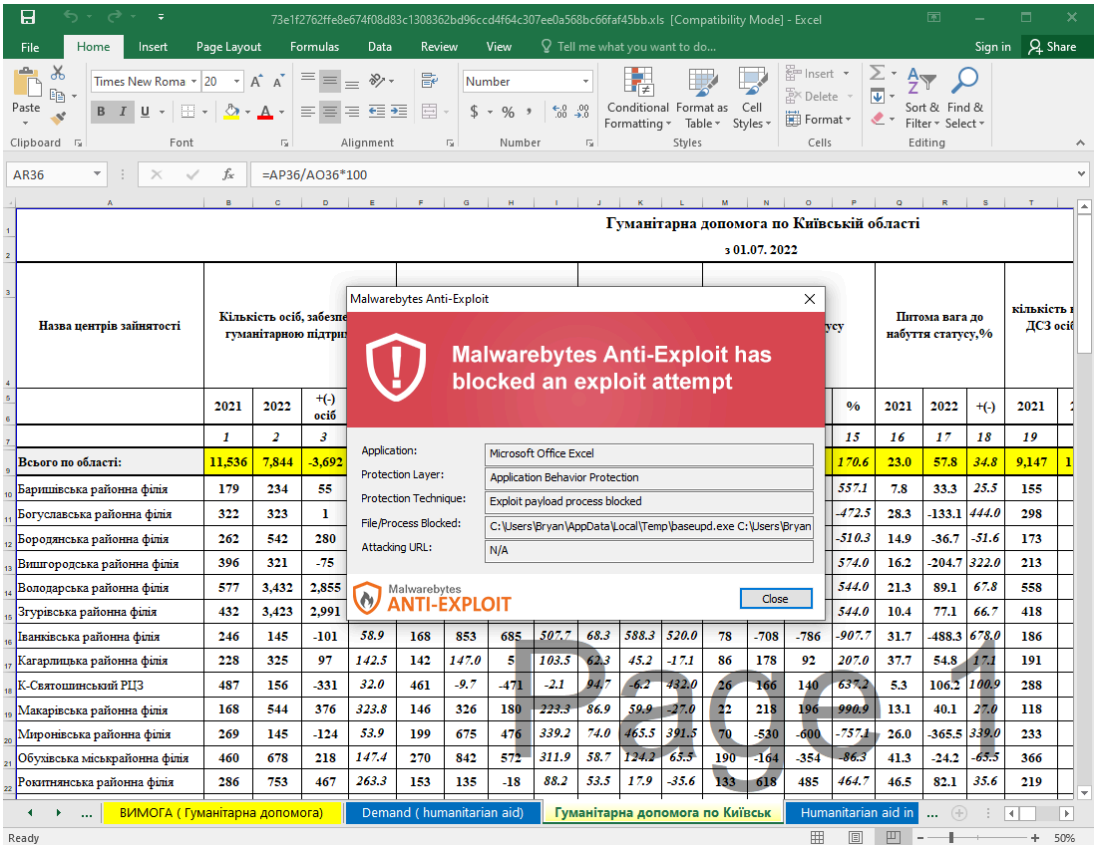
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

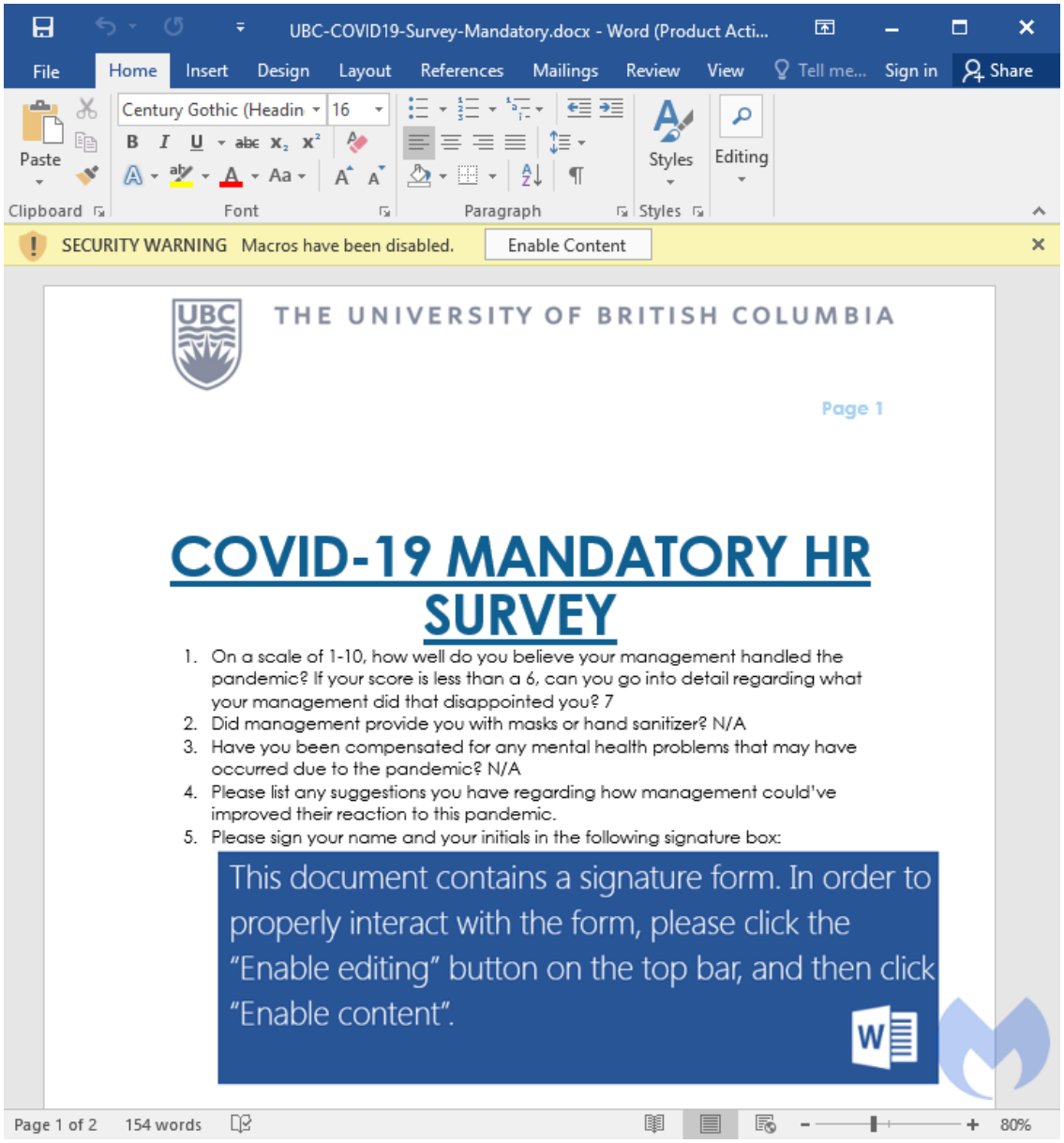
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

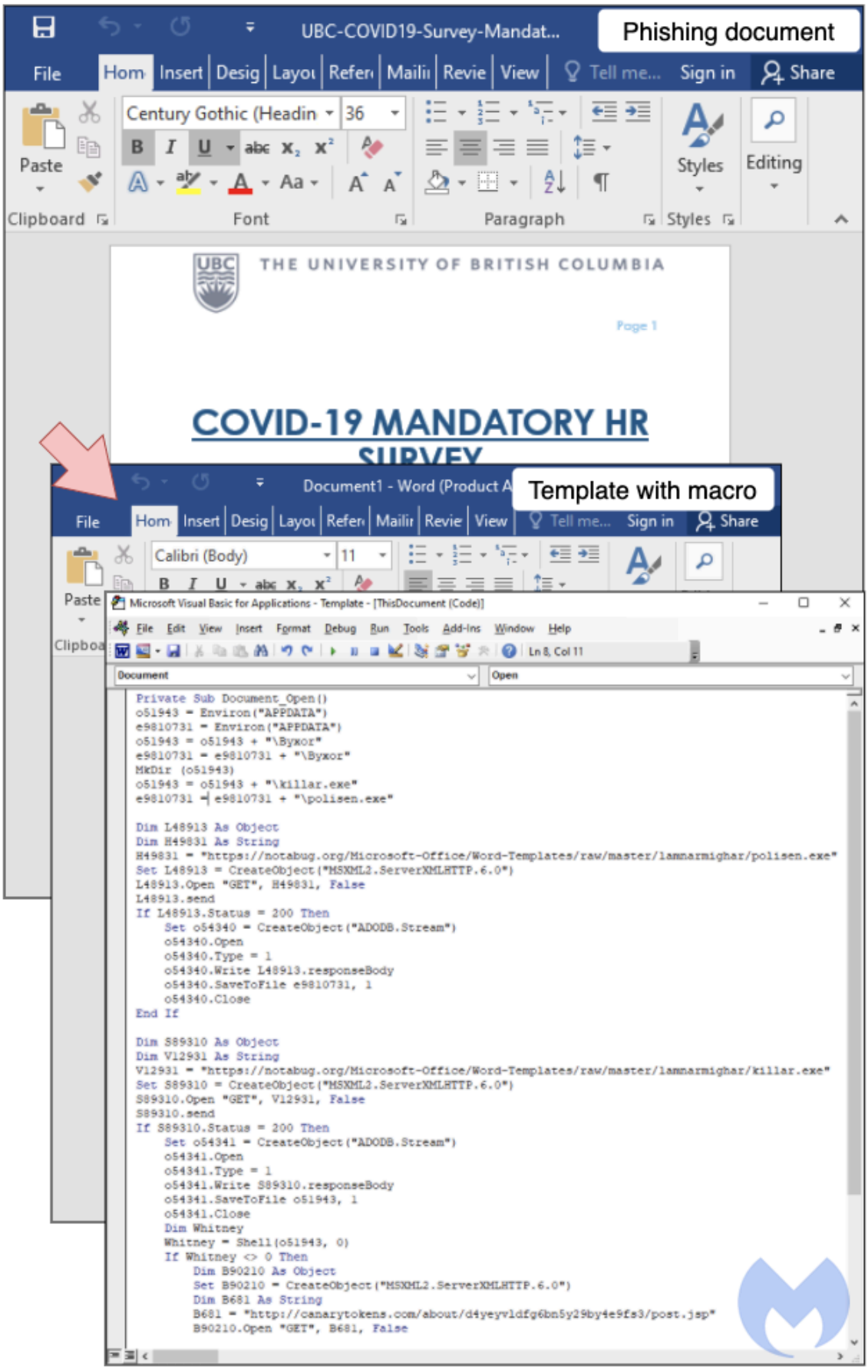
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

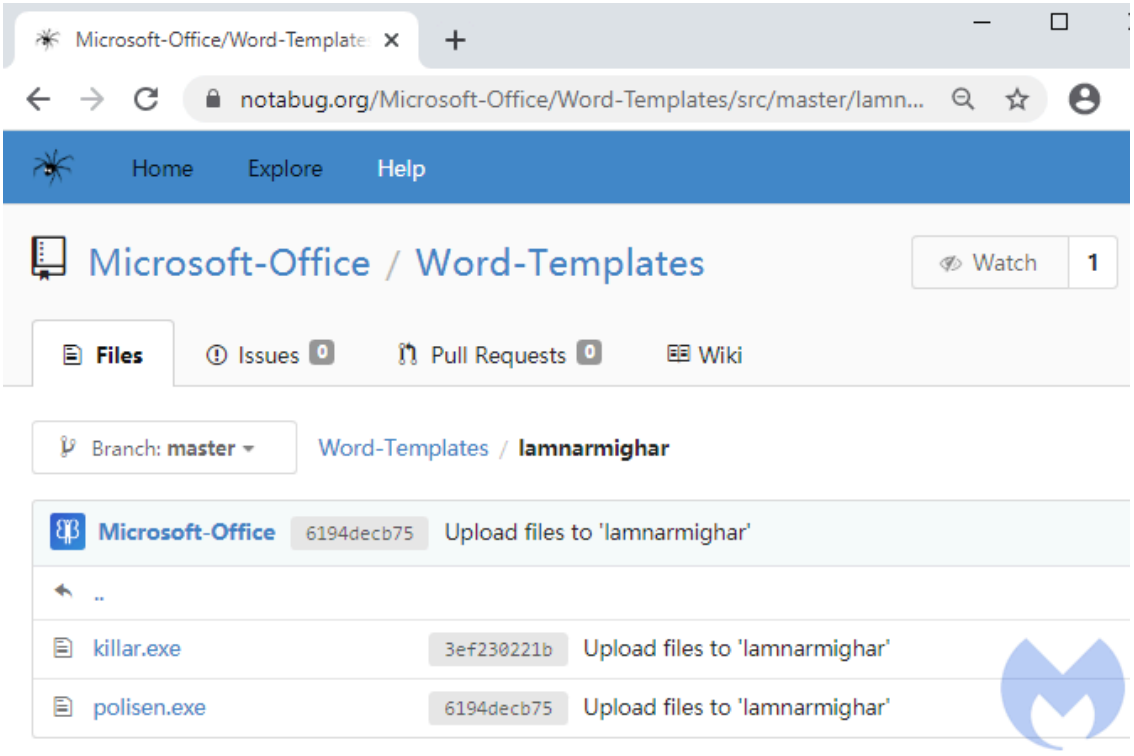
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



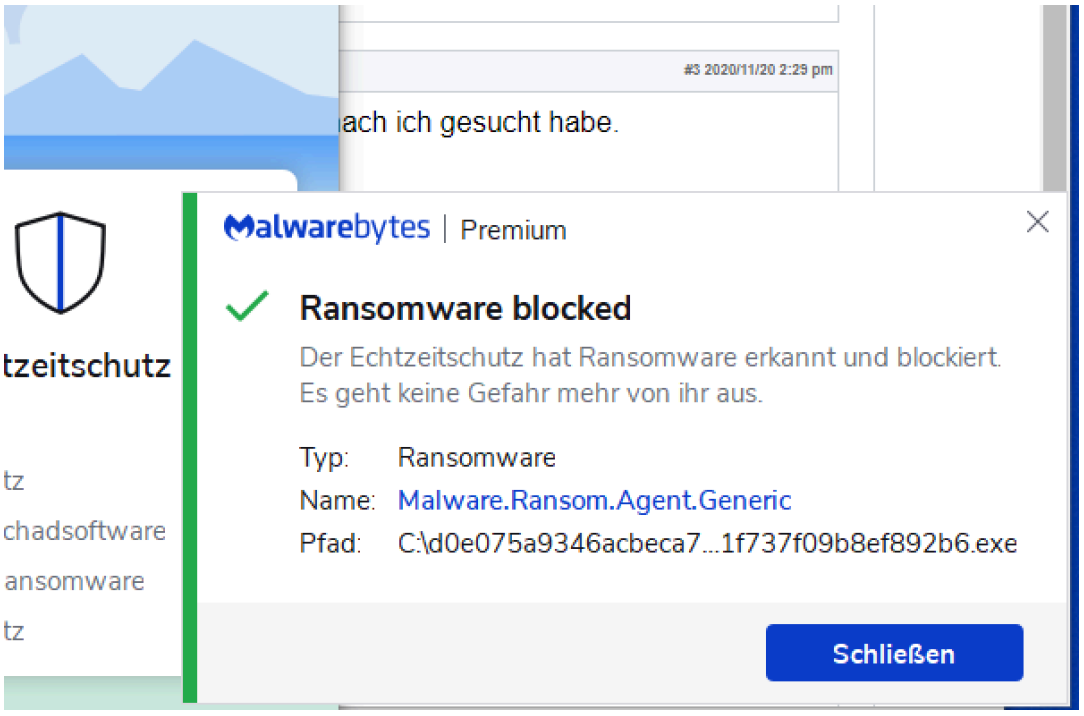
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

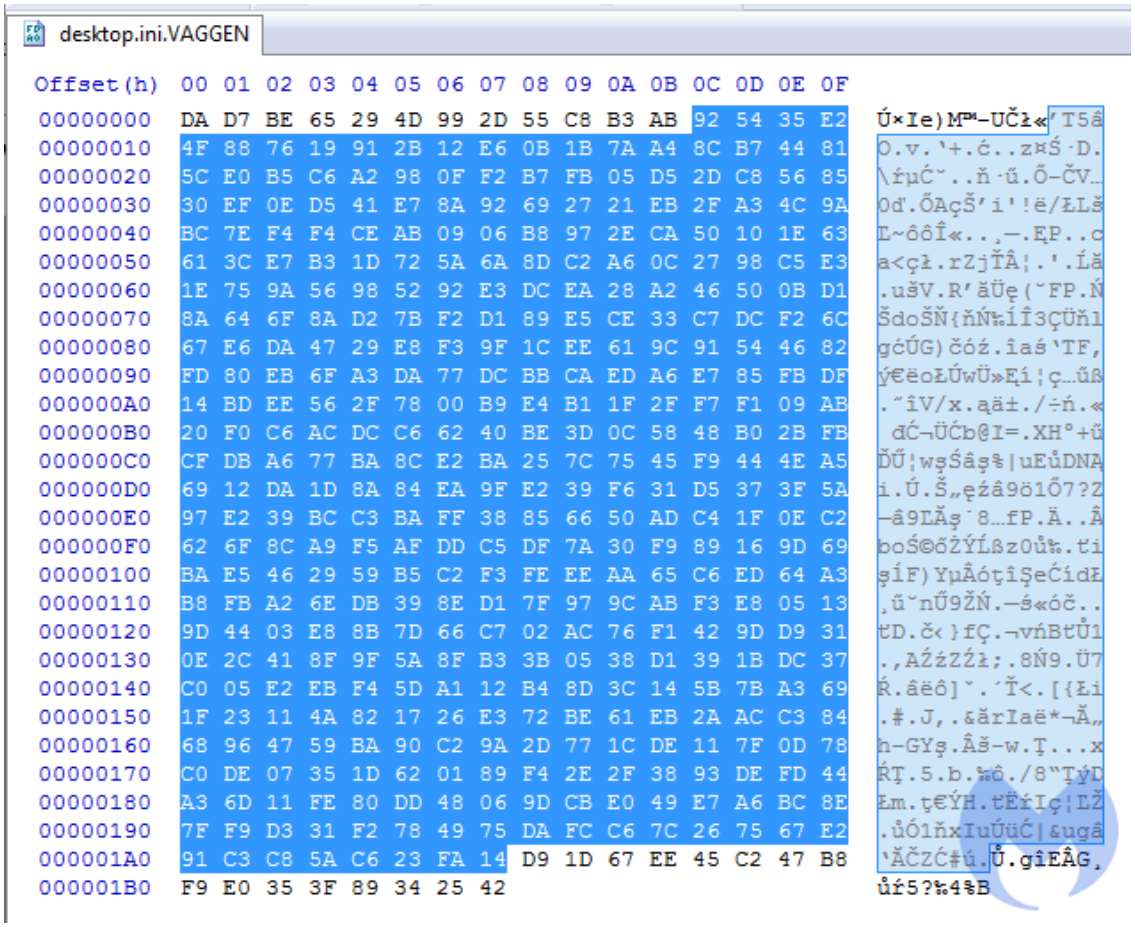
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

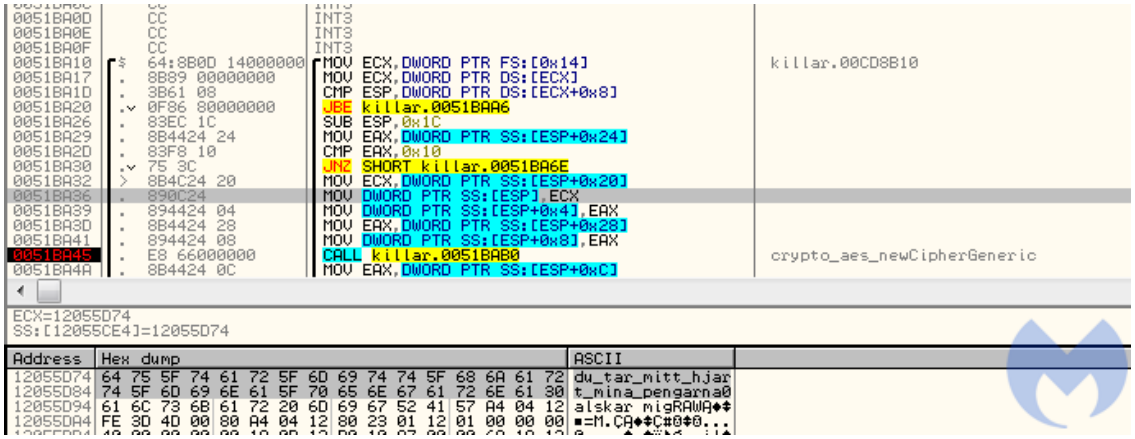


The content of the output file (with .VAGGEN extension) contains:

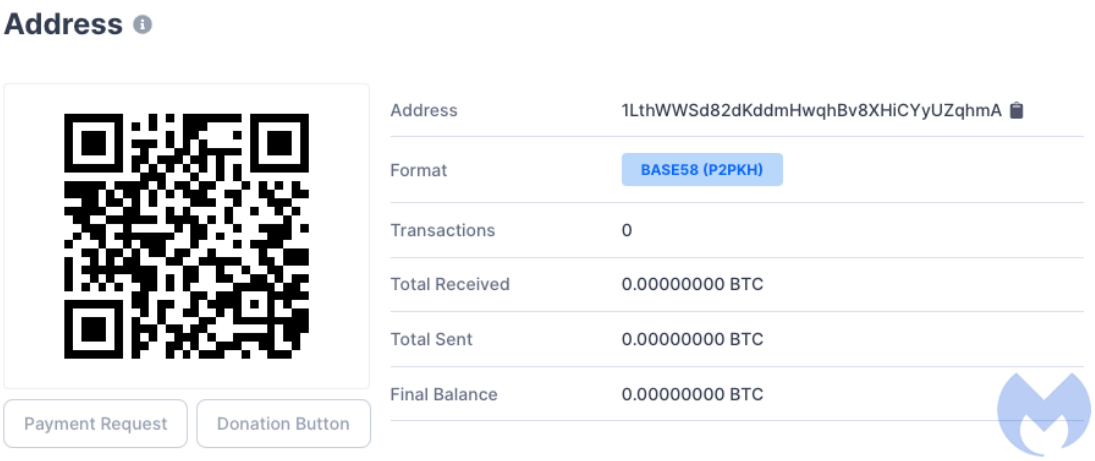
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

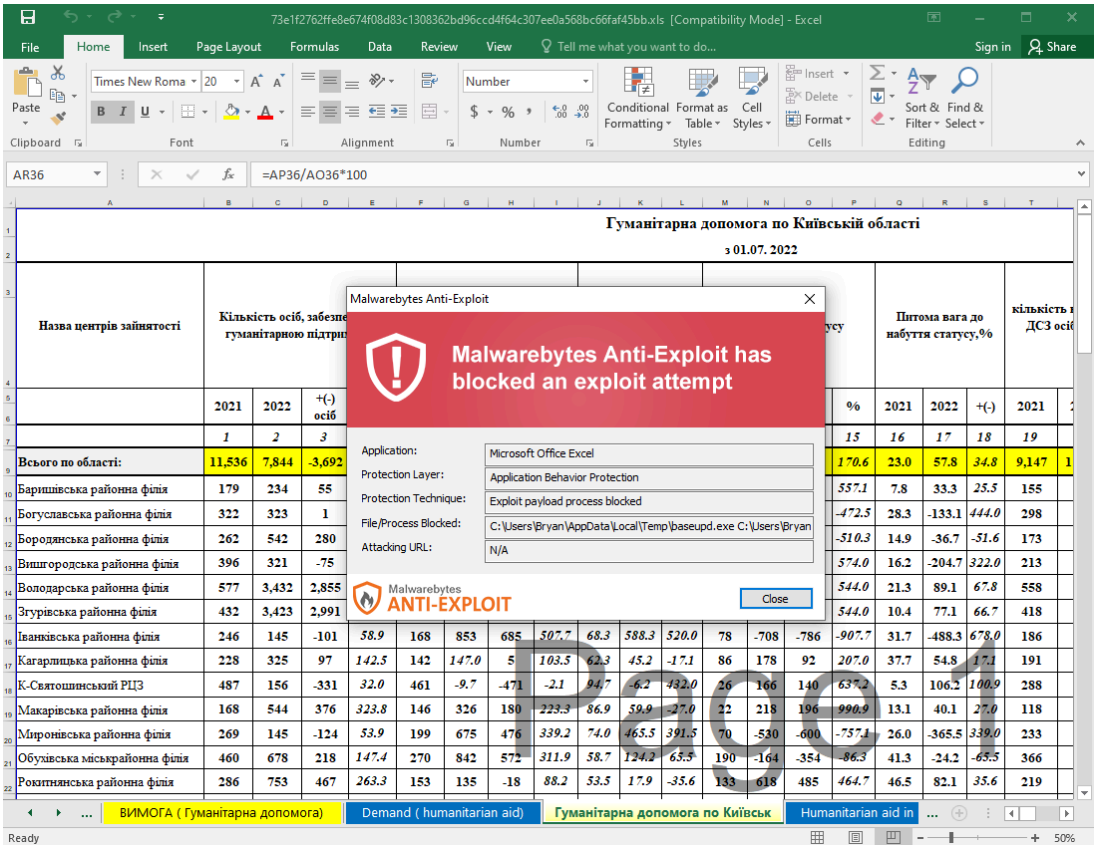
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

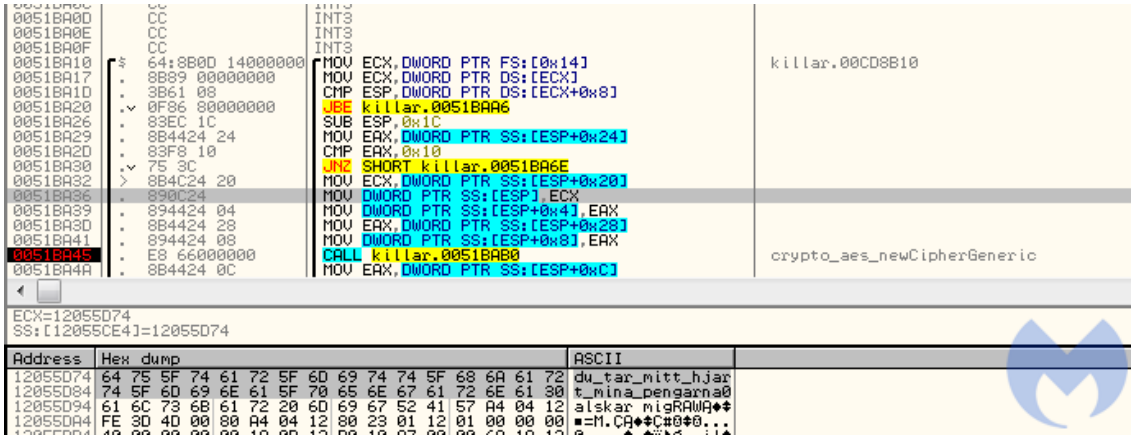
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

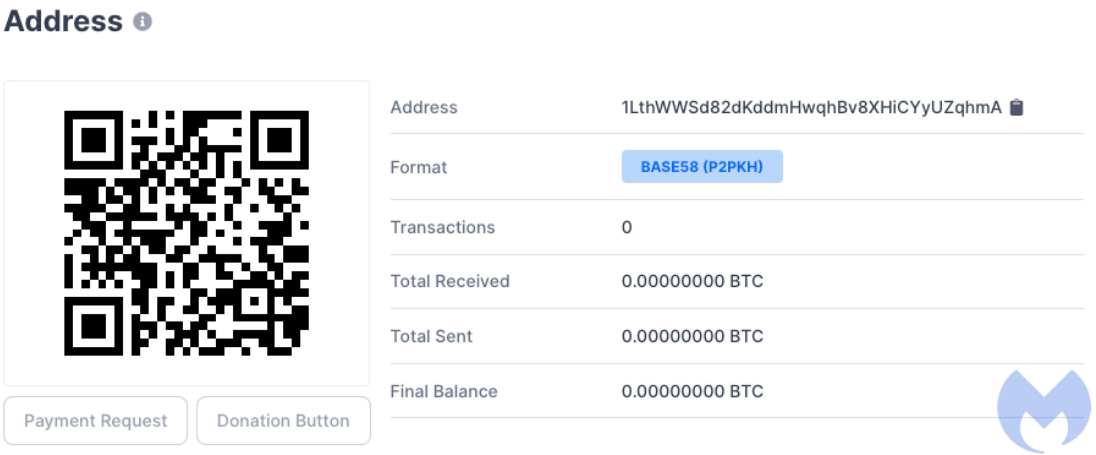
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

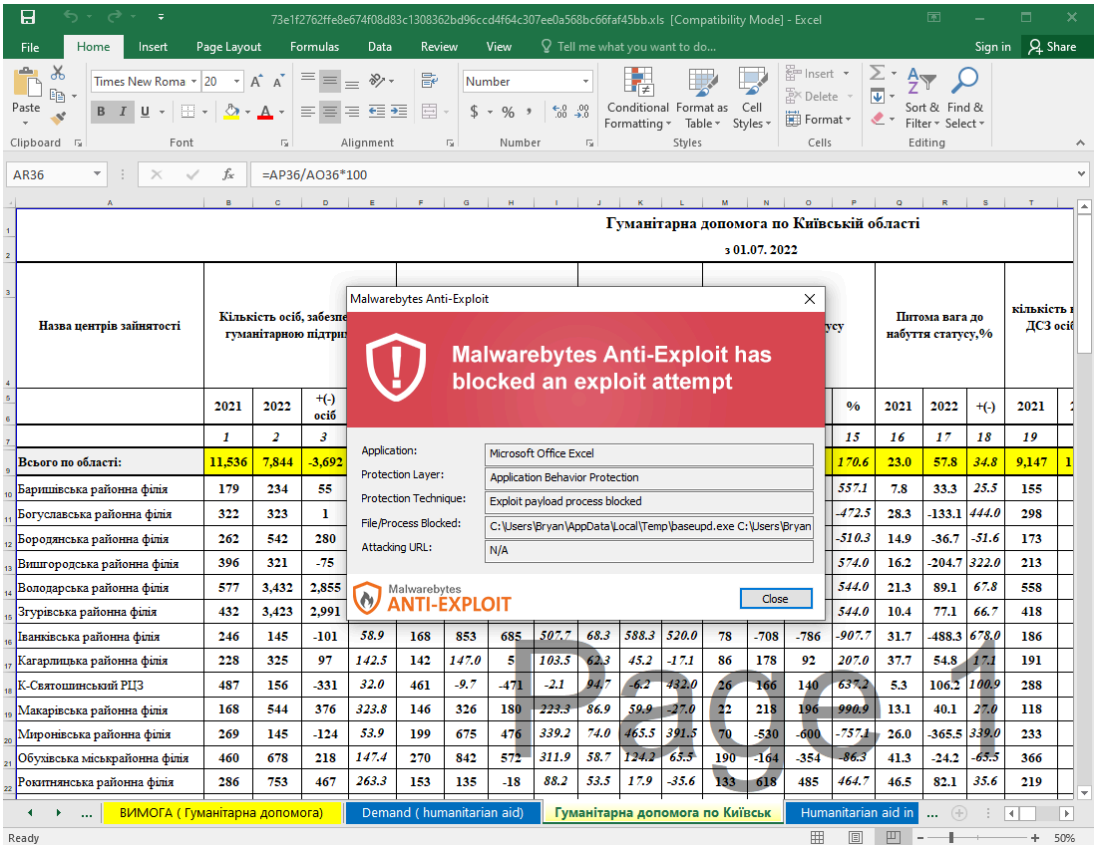
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

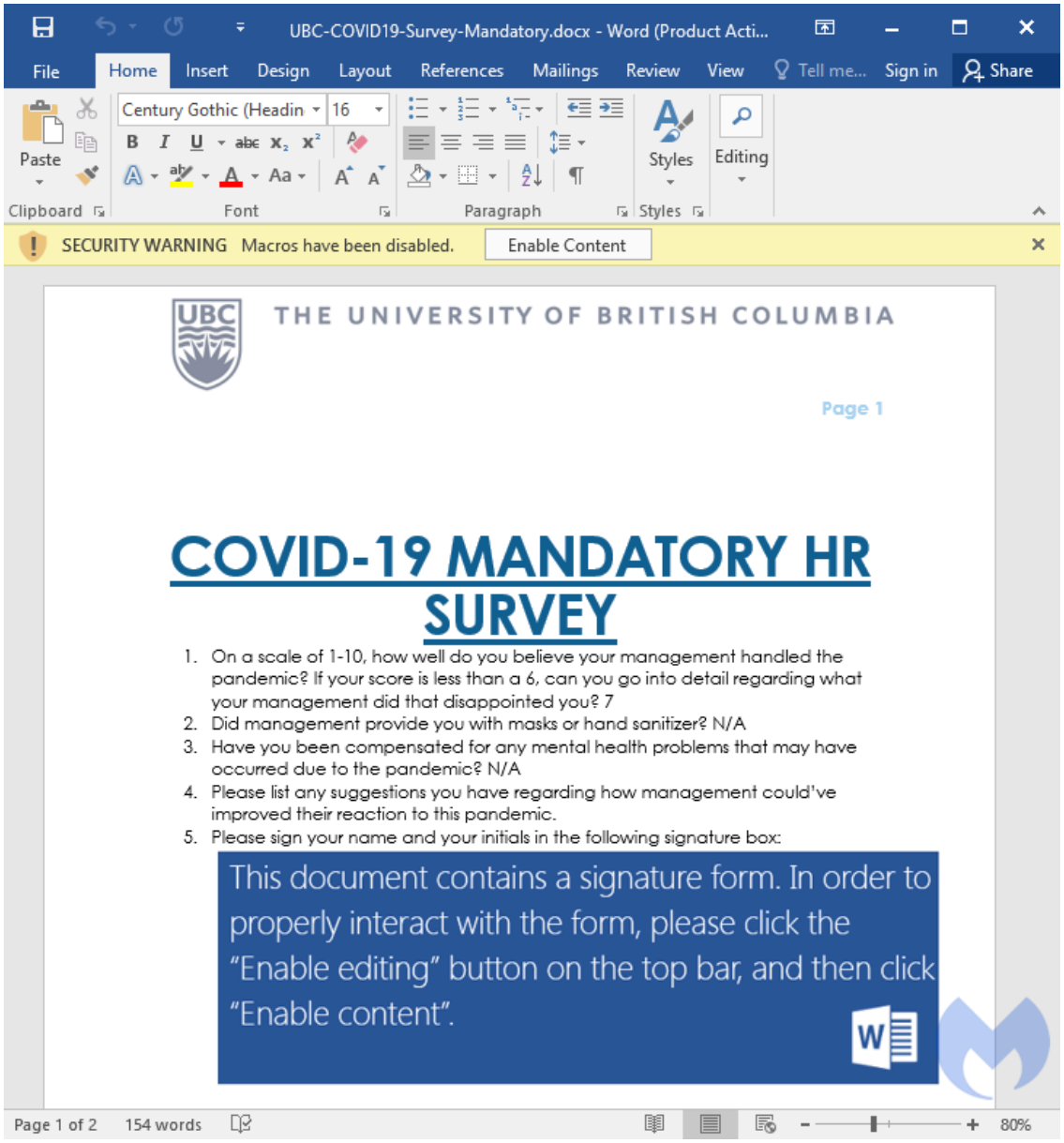
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

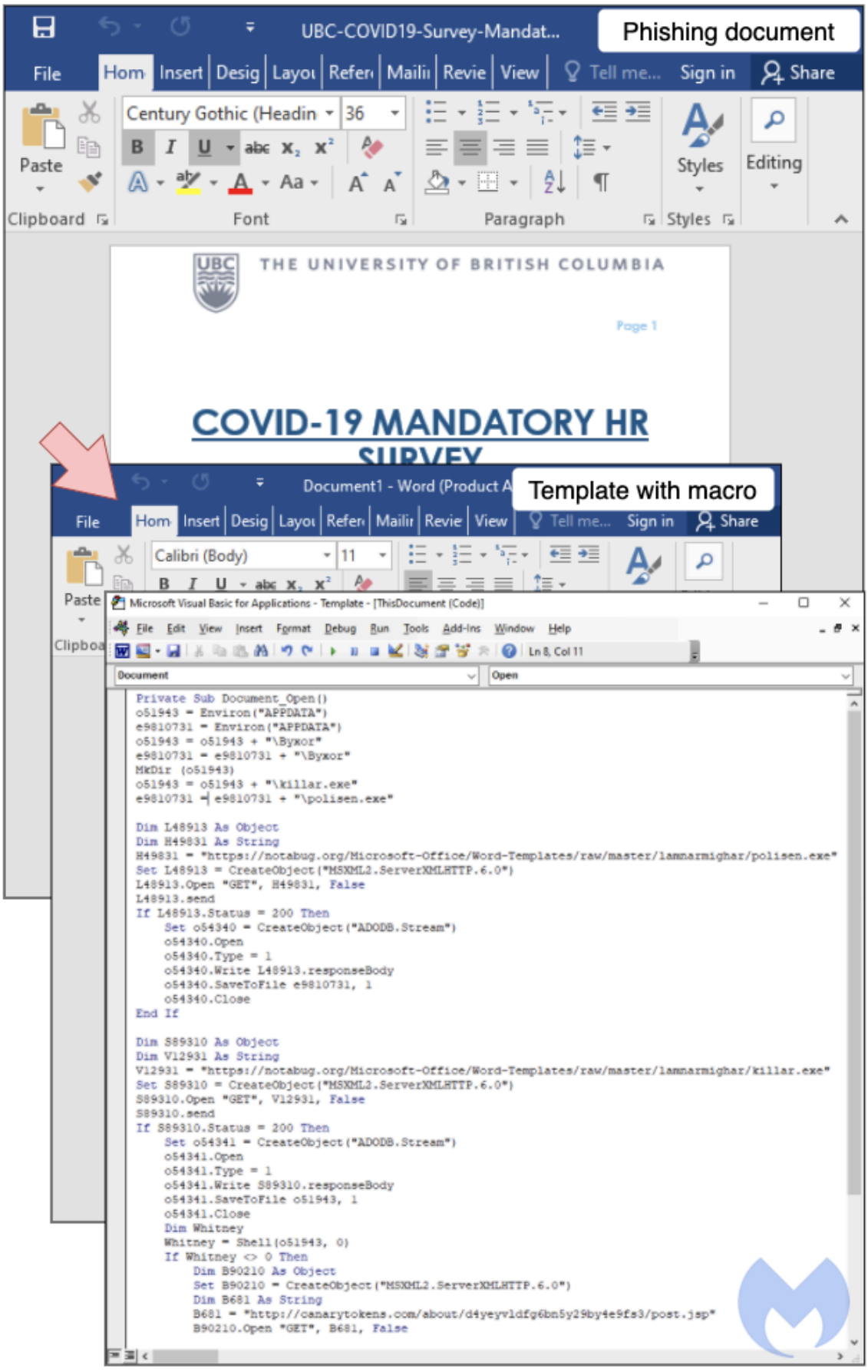
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

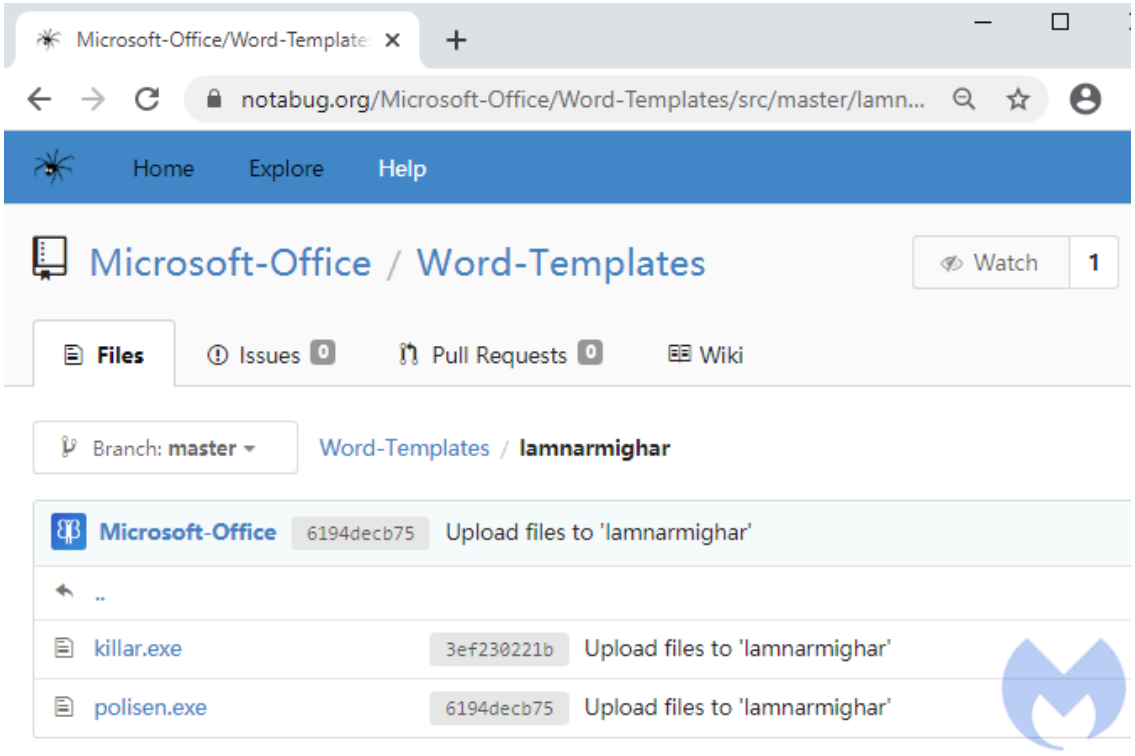
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



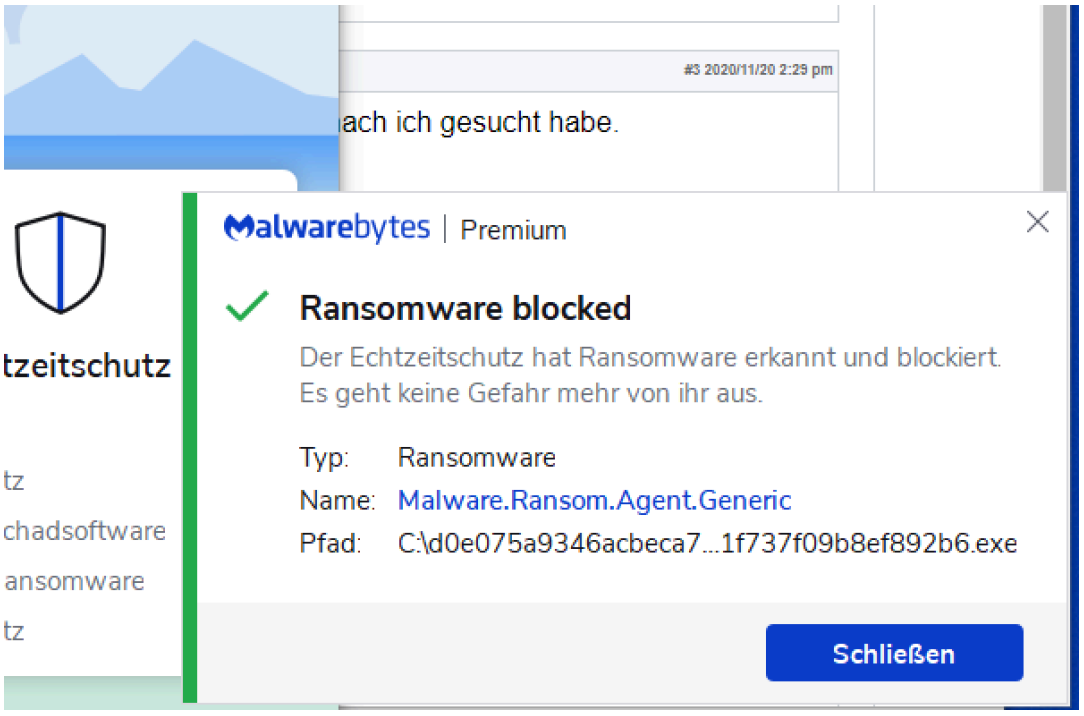
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

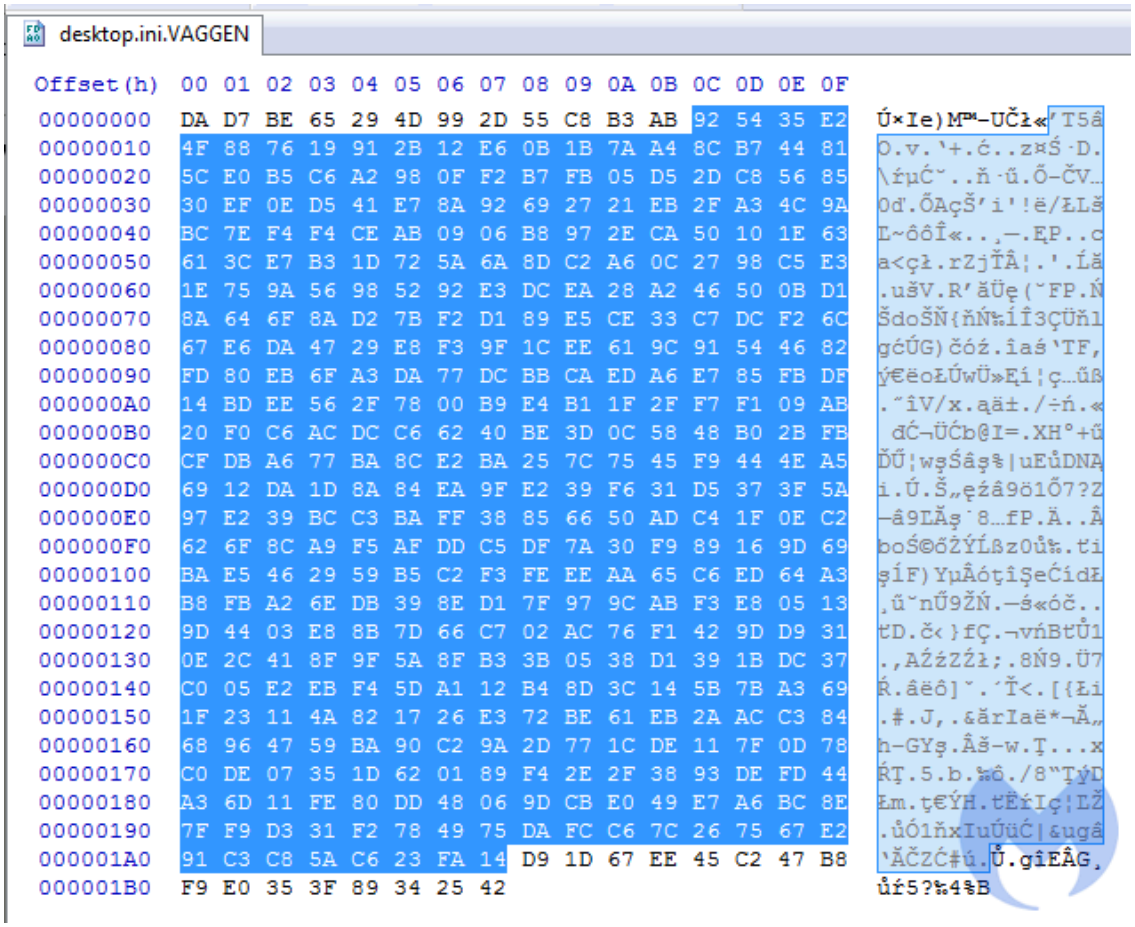
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

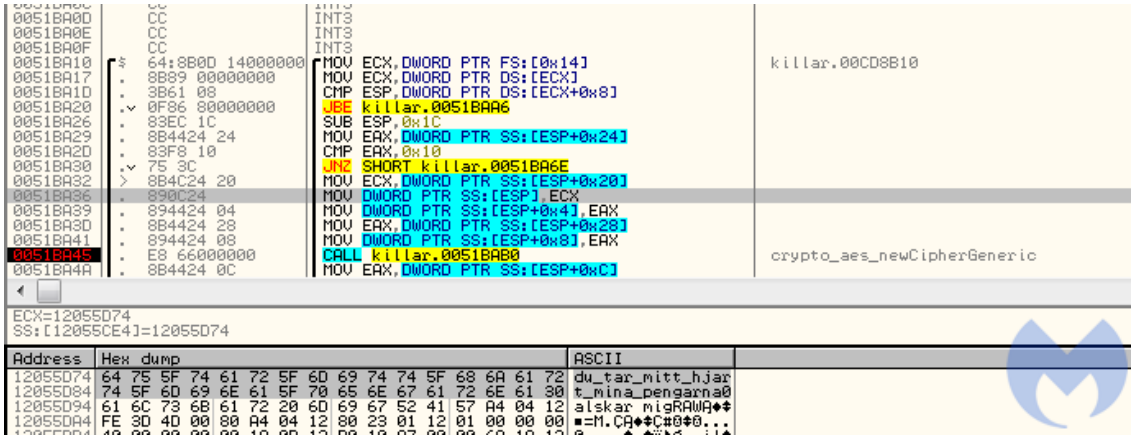


The content of the output file (with .VAGGEN extension) contains:

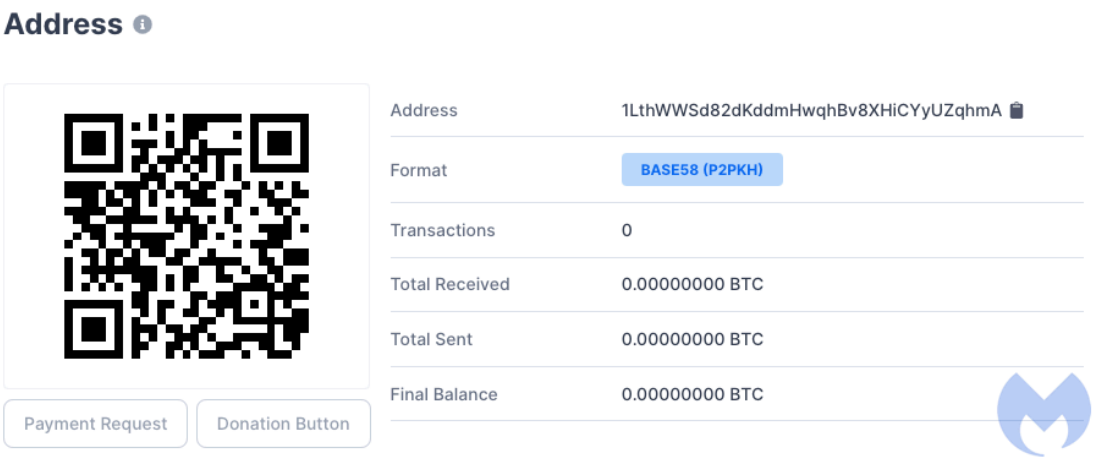
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

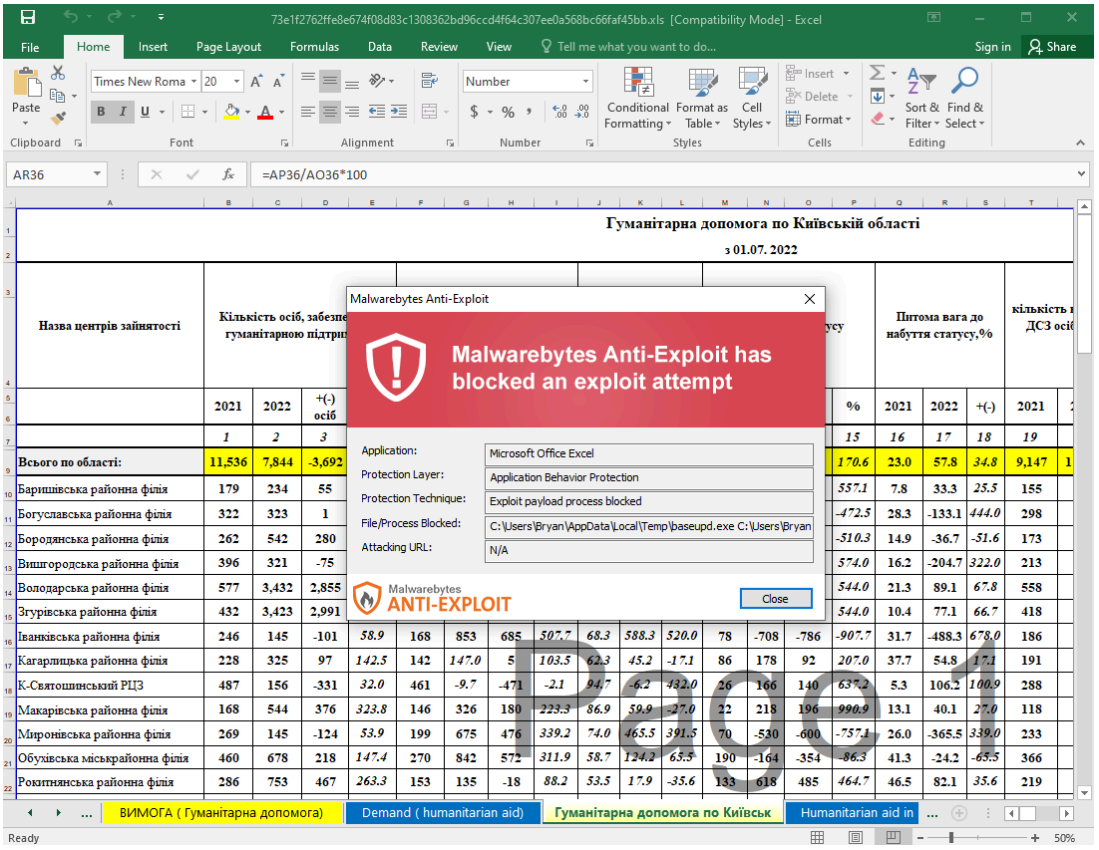
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

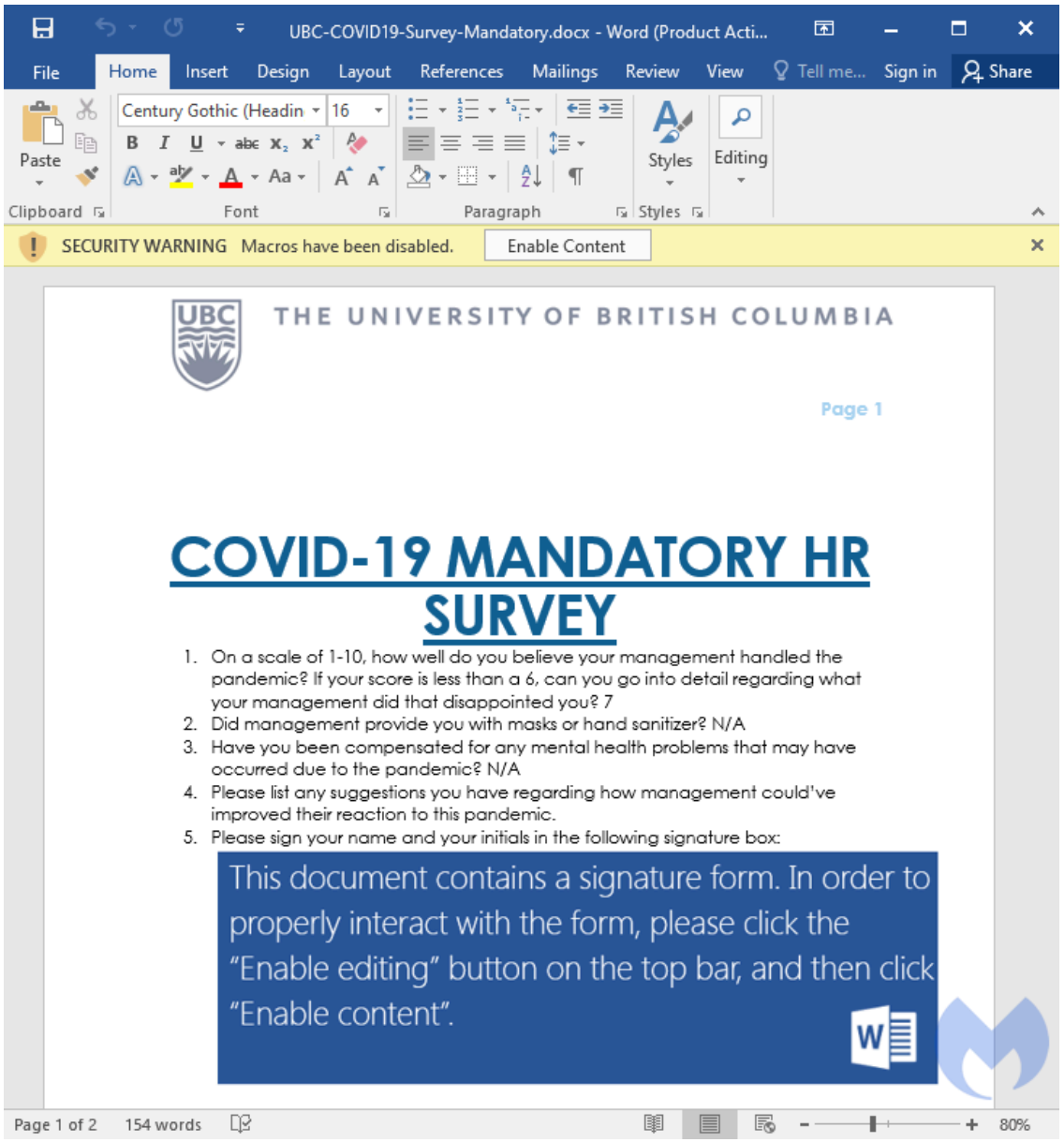
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

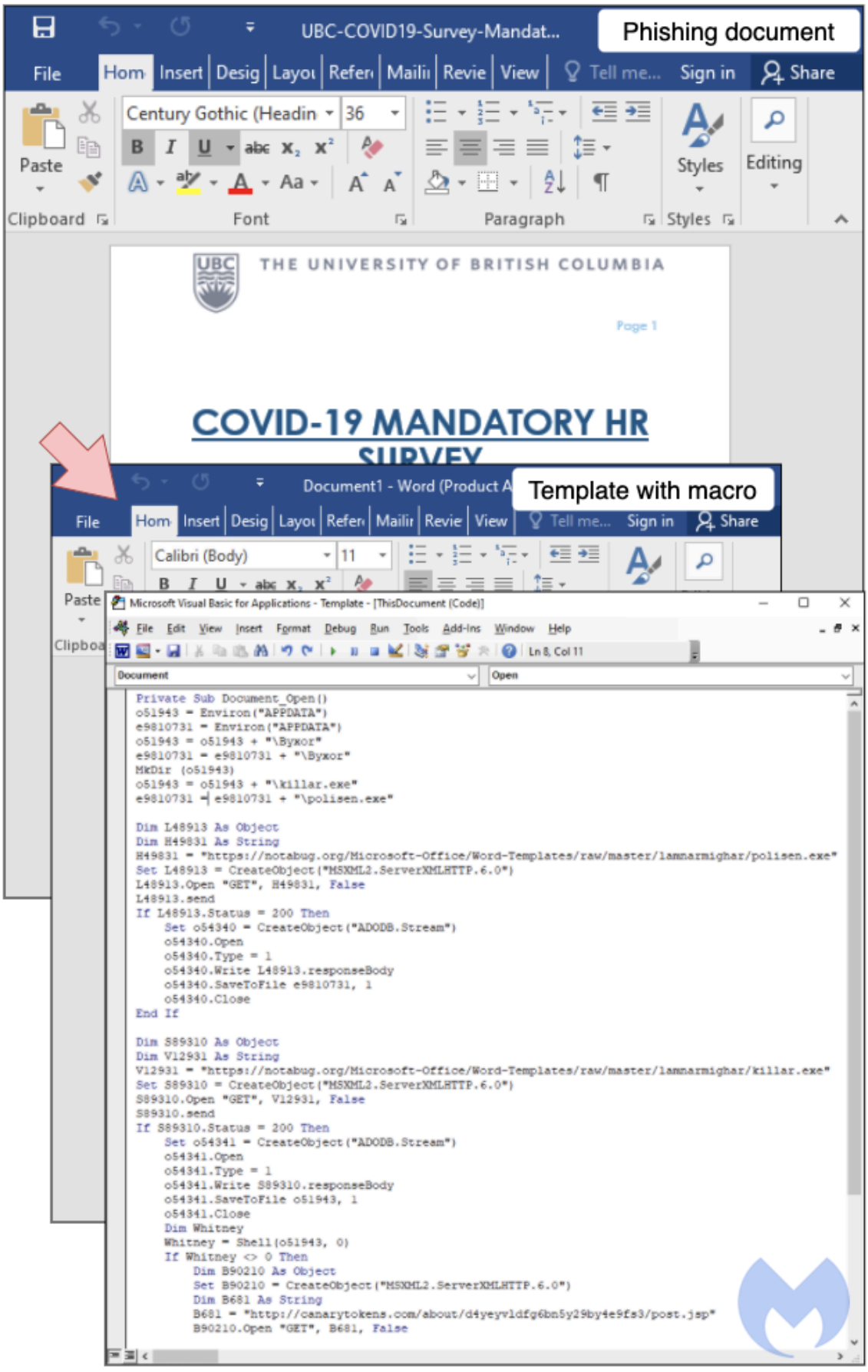
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

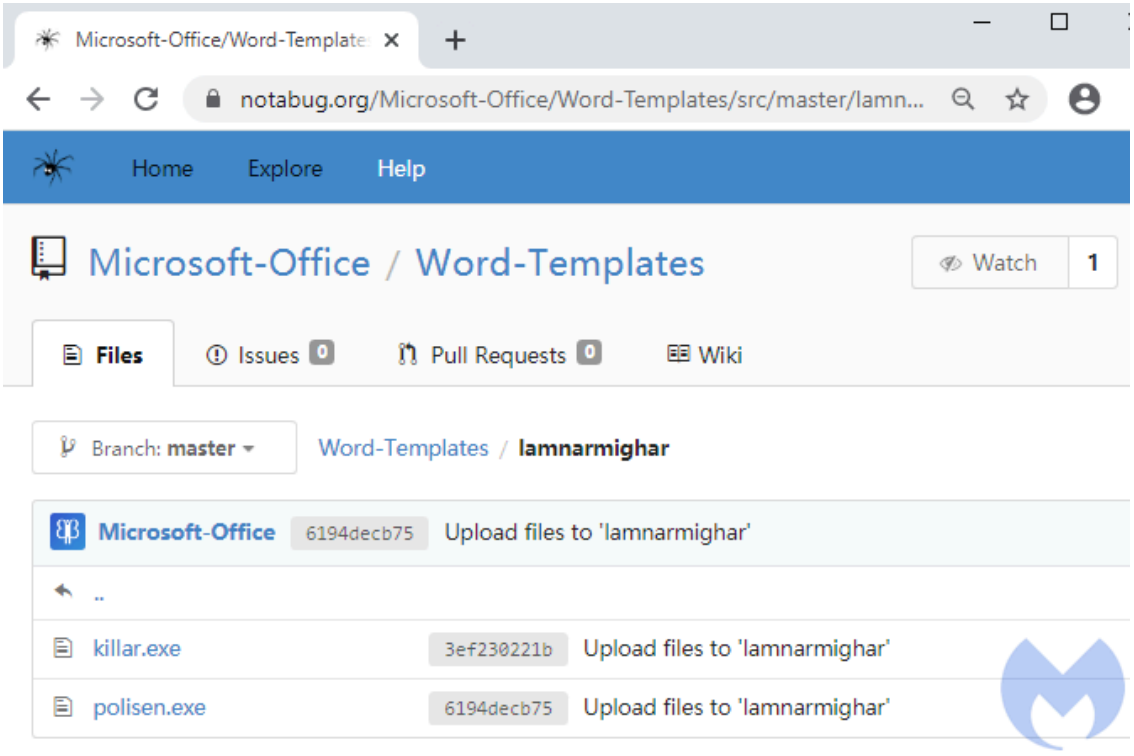
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



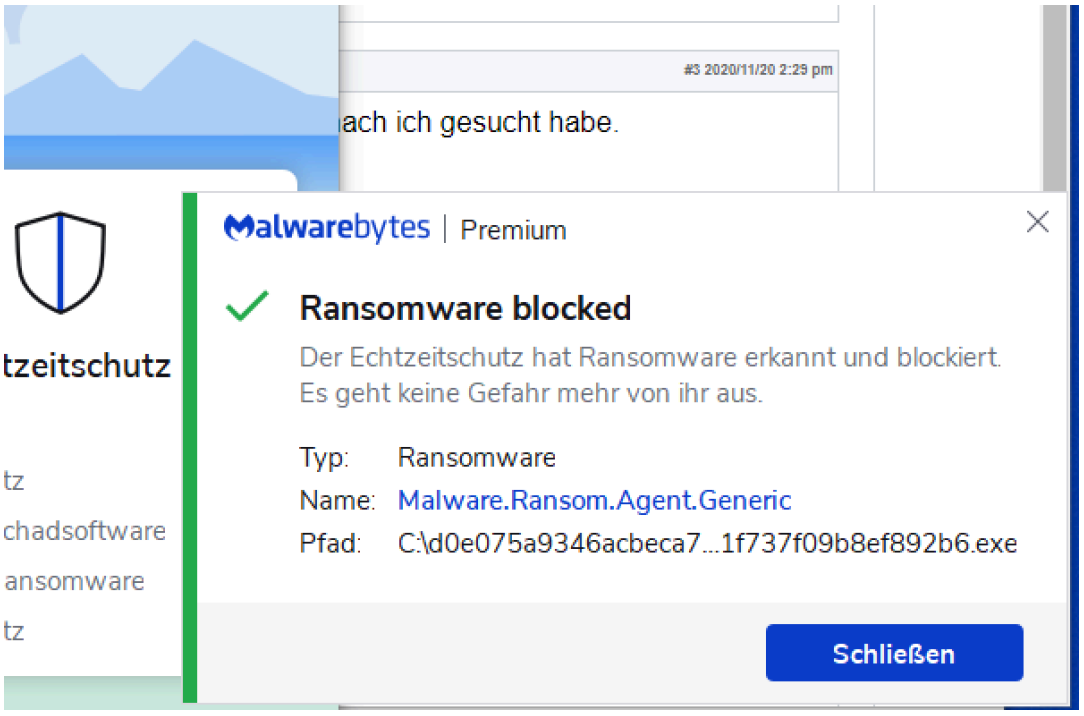
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

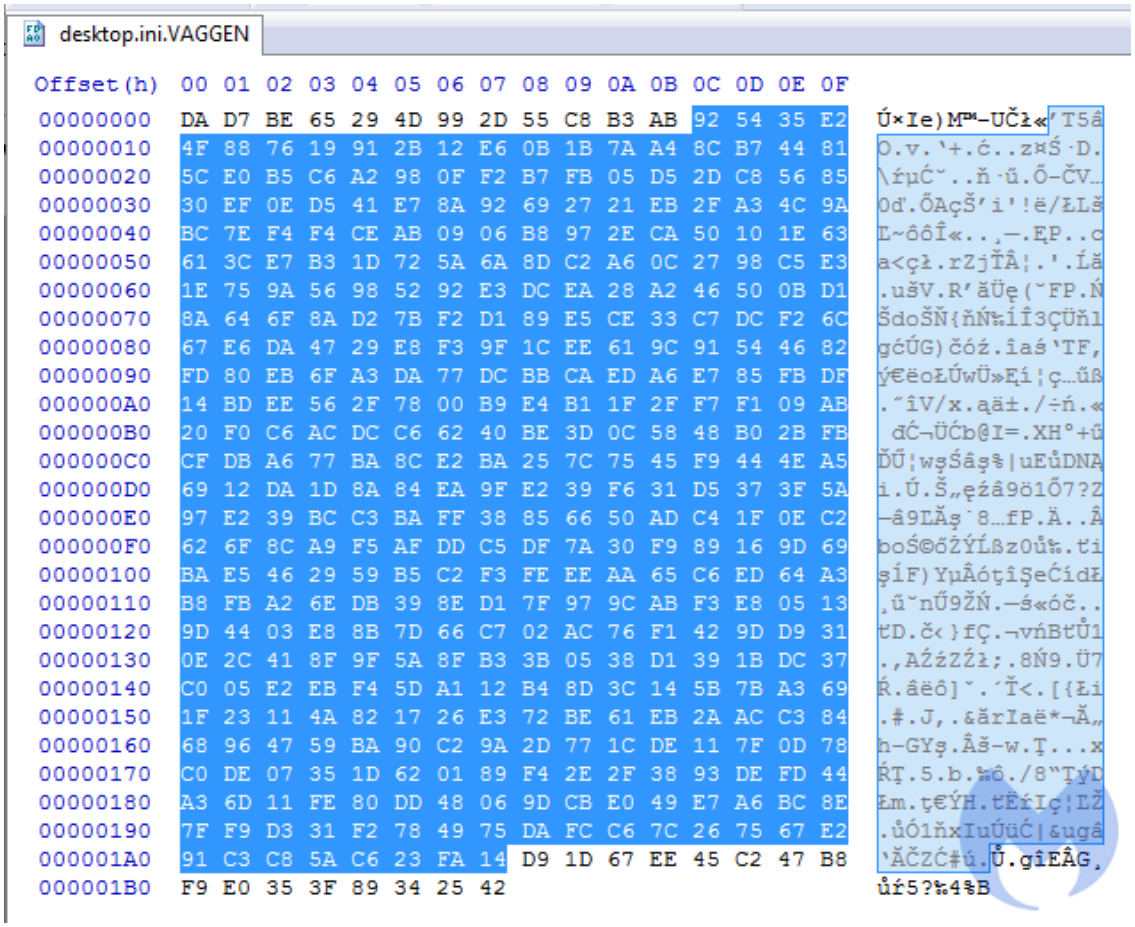
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

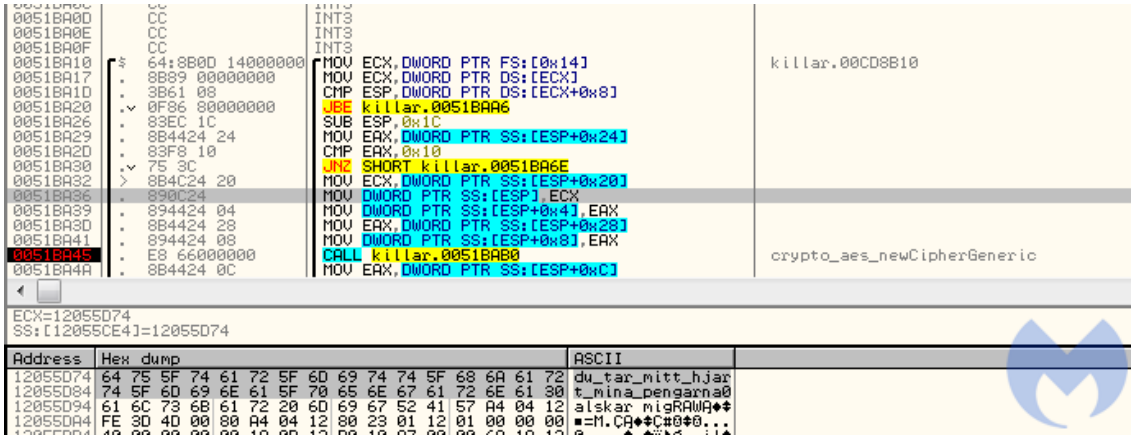


The content of the output file (with .VAGGEN extension) contains:

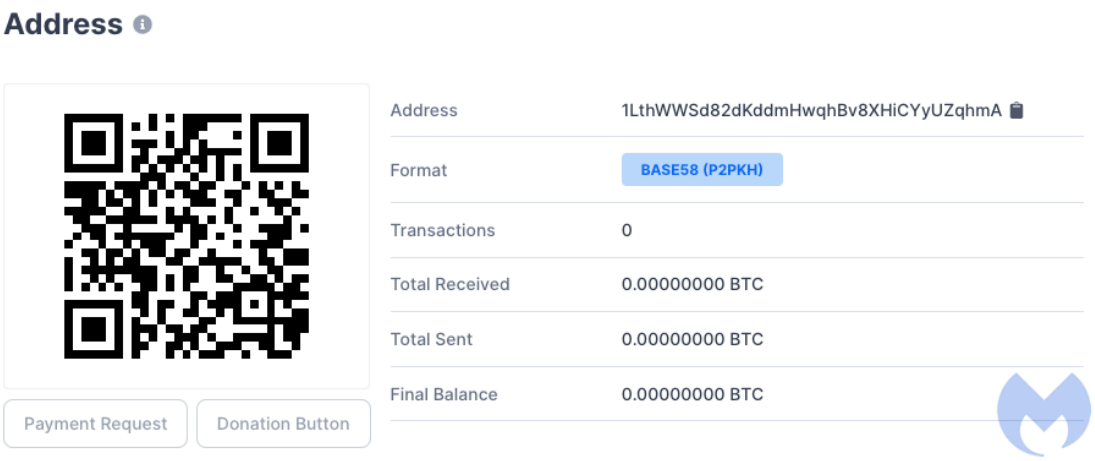
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

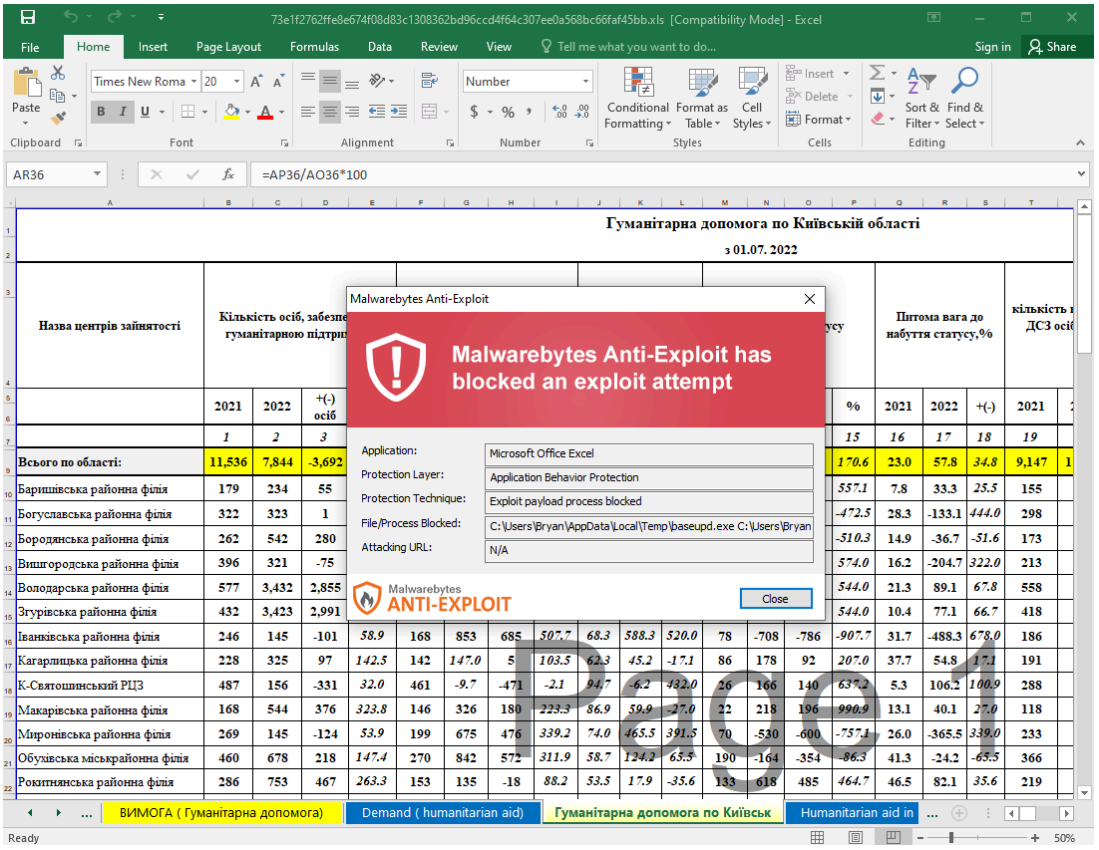
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

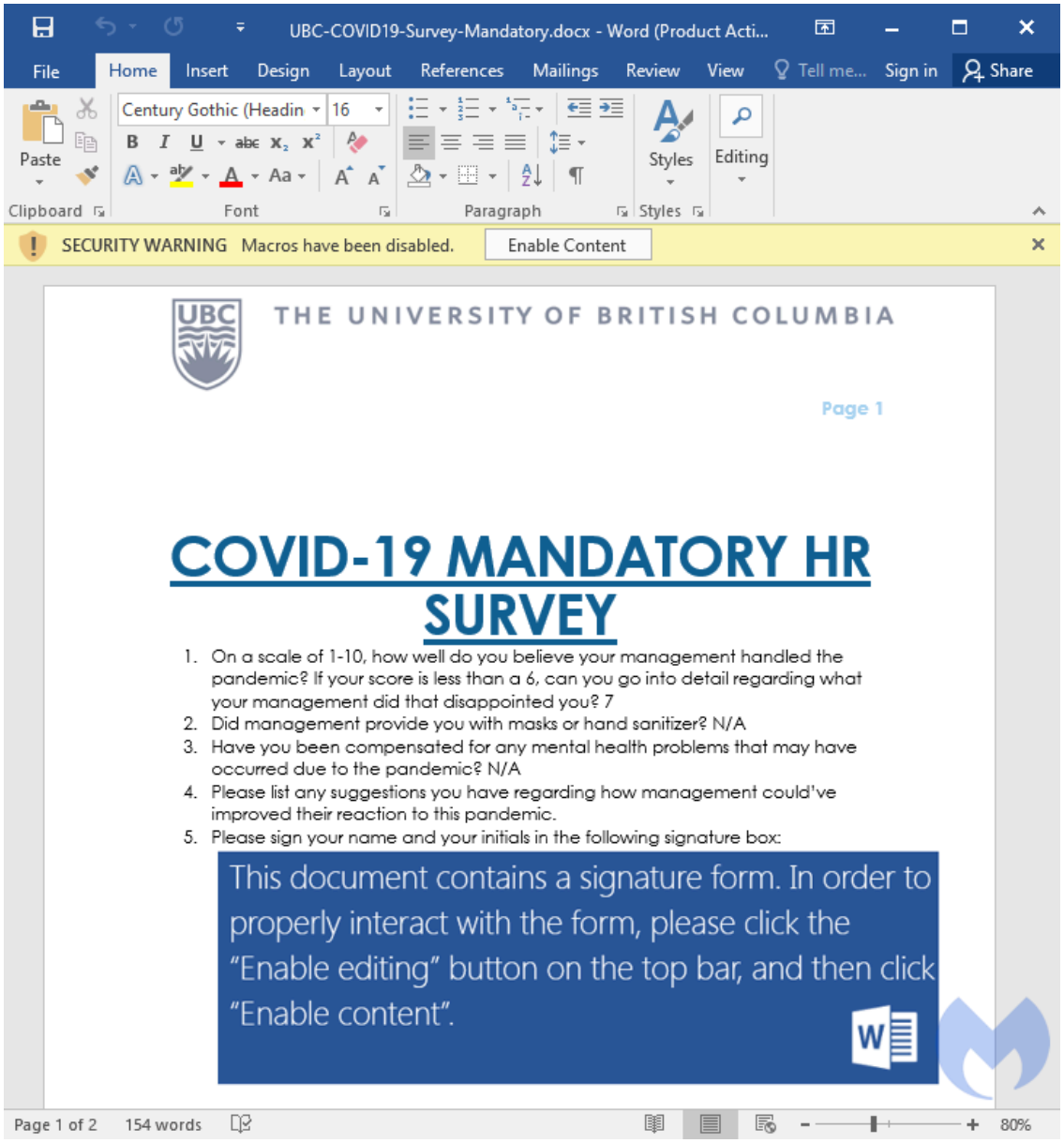
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

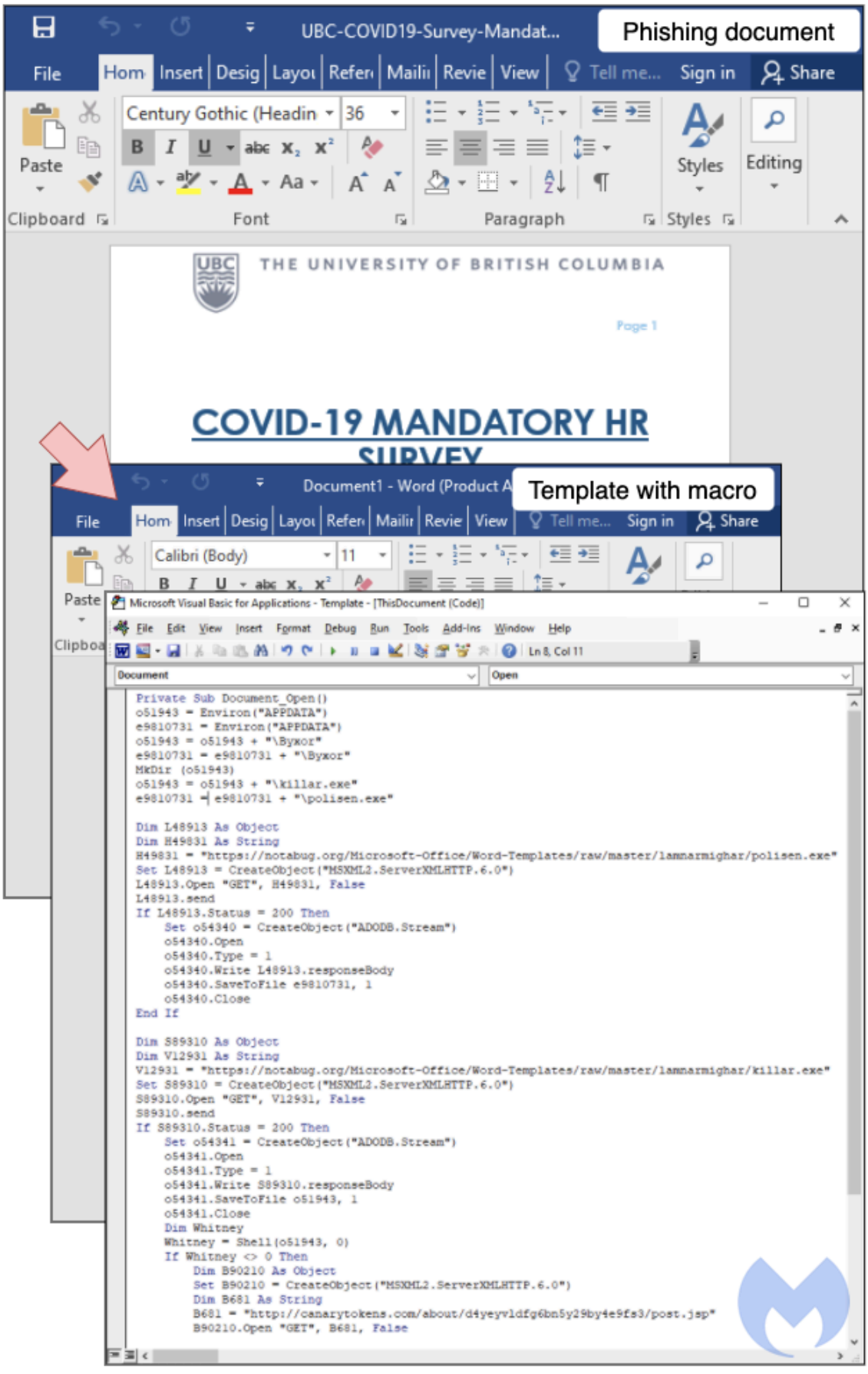
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

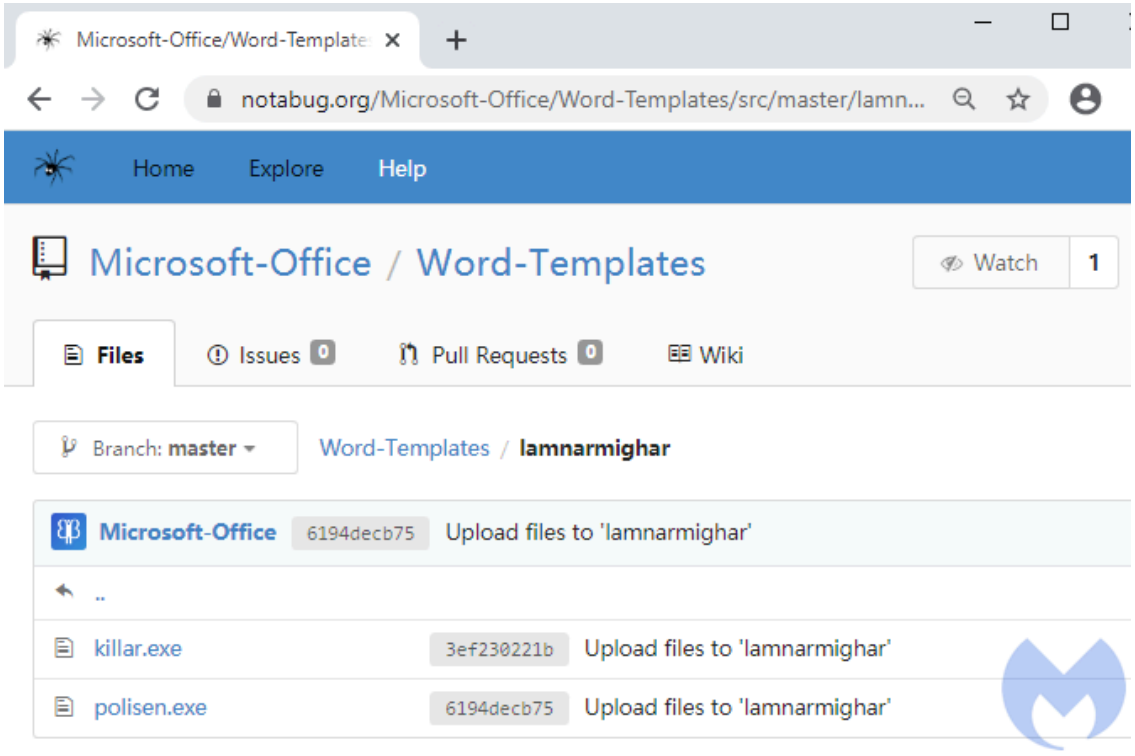
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



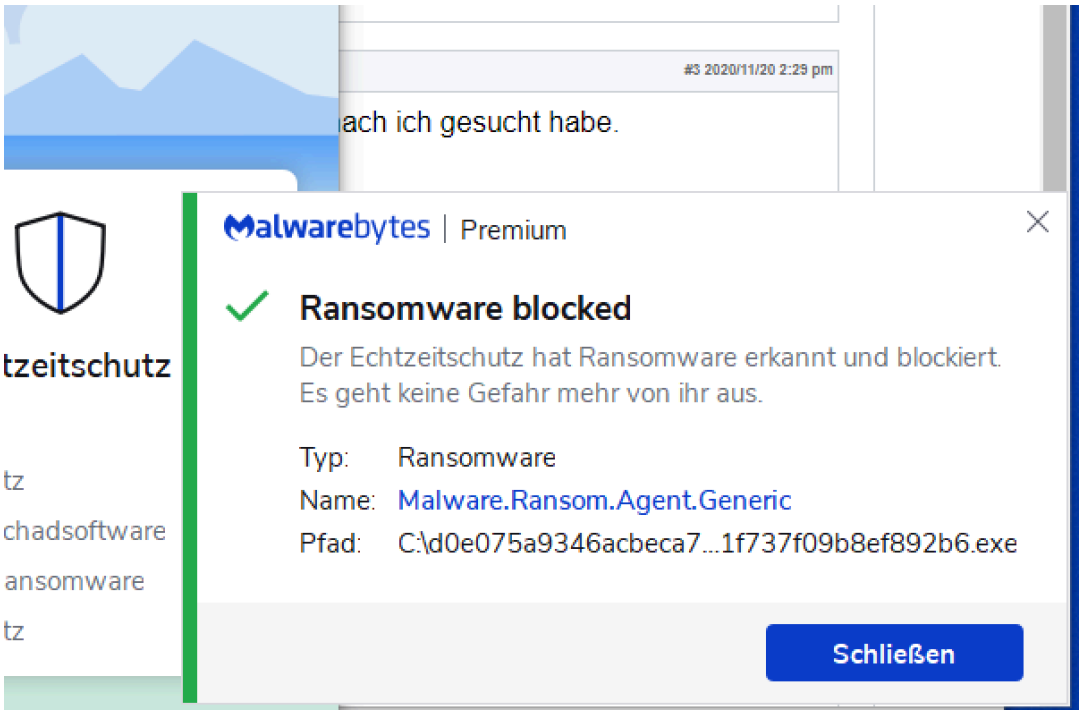
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

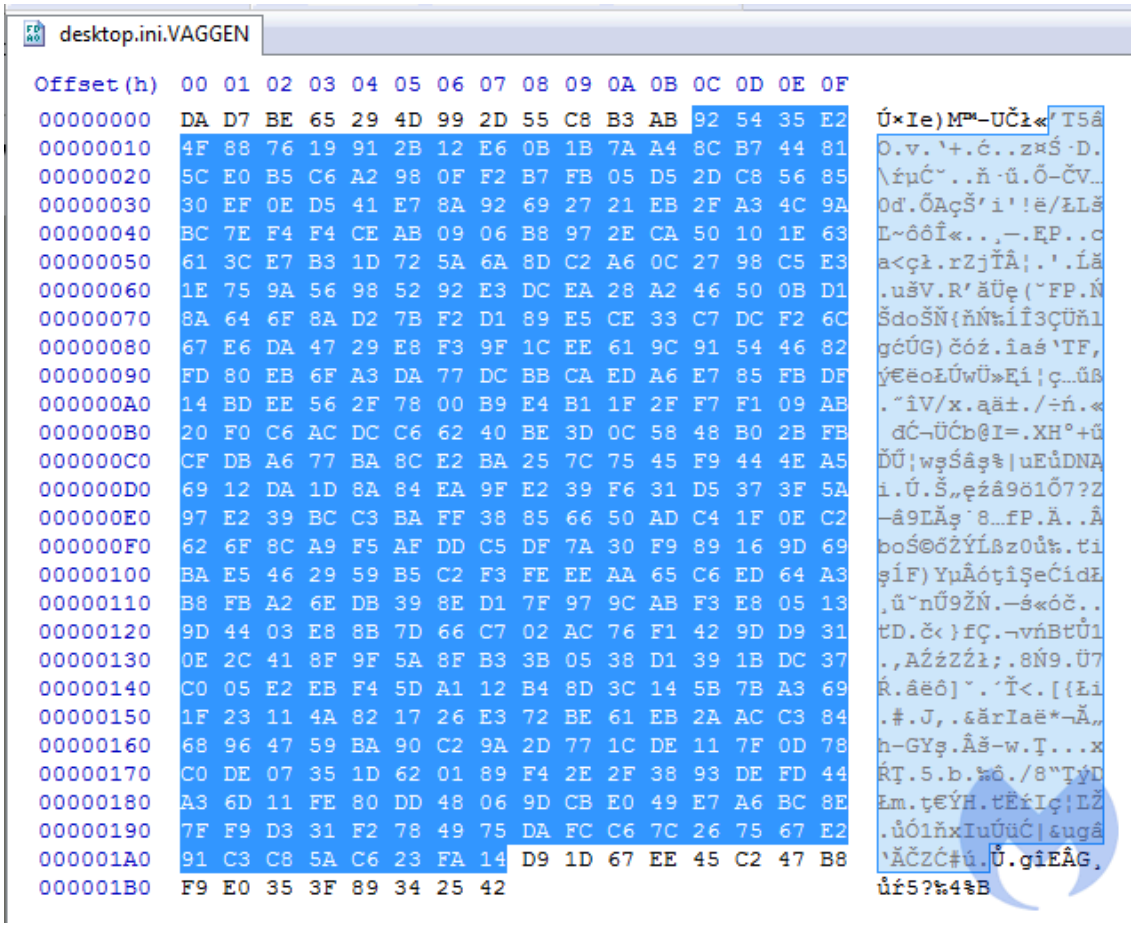
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

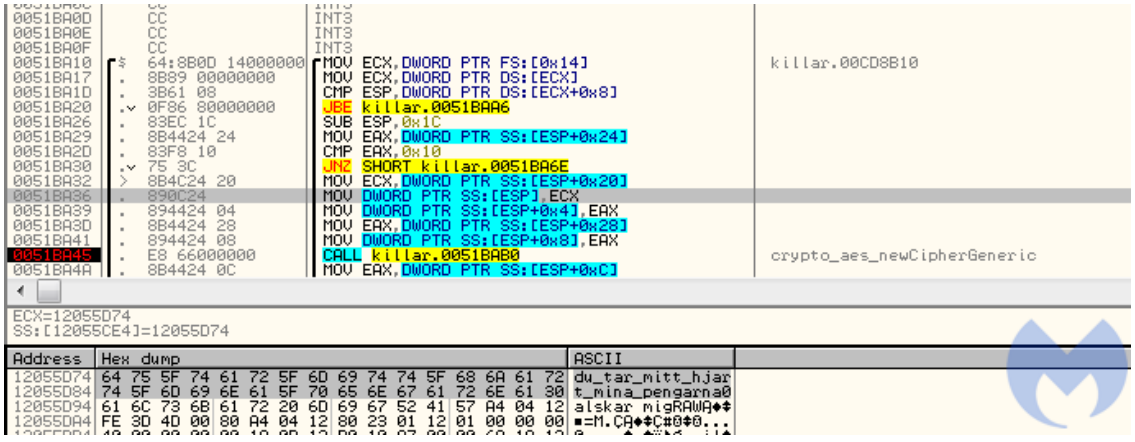


The content of the output file (with .VAGGEN extension) contains:

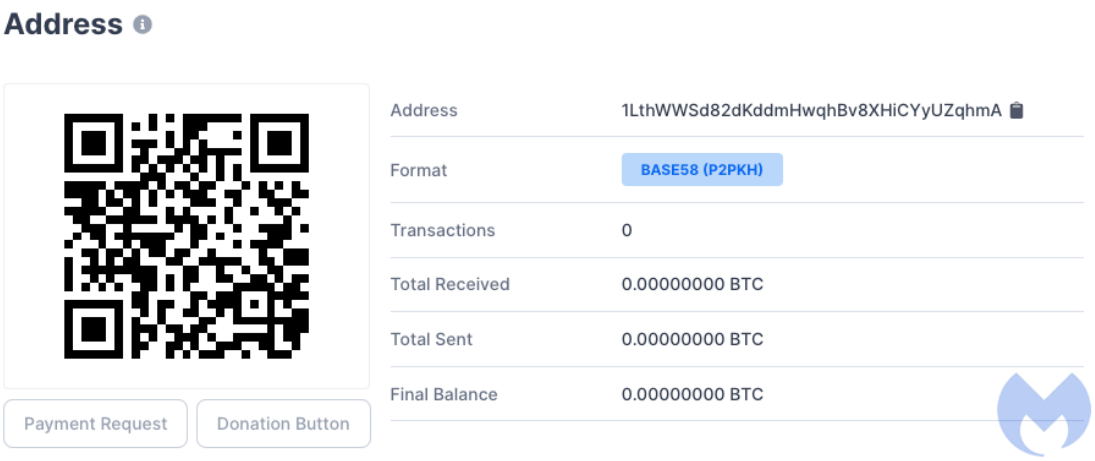
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

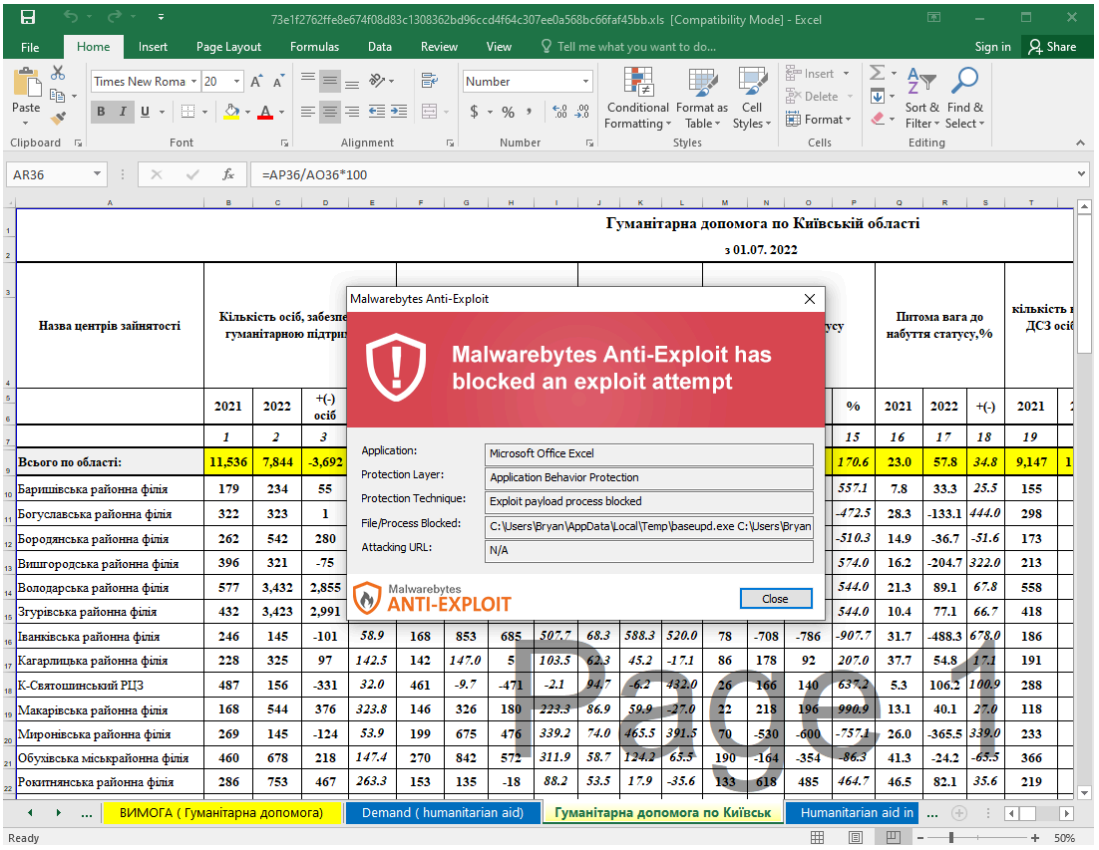
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

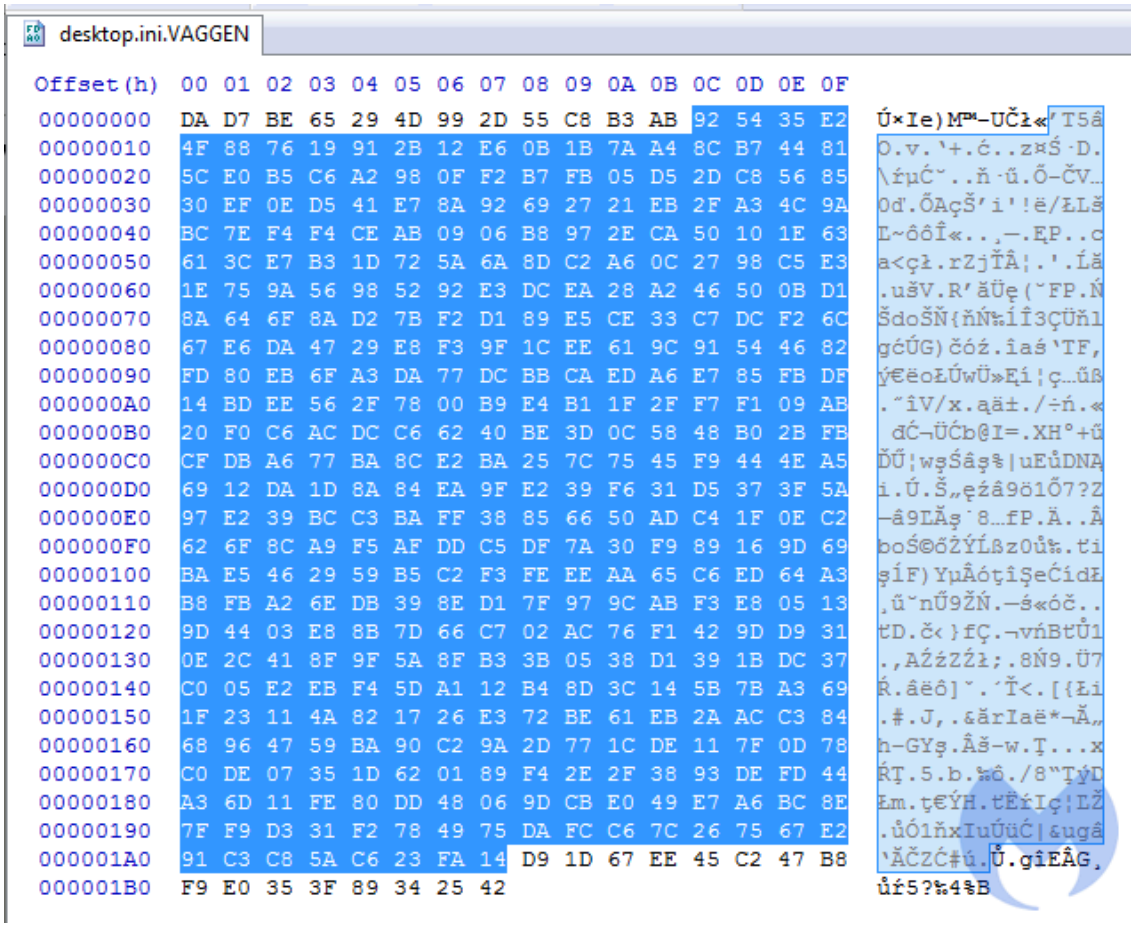
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

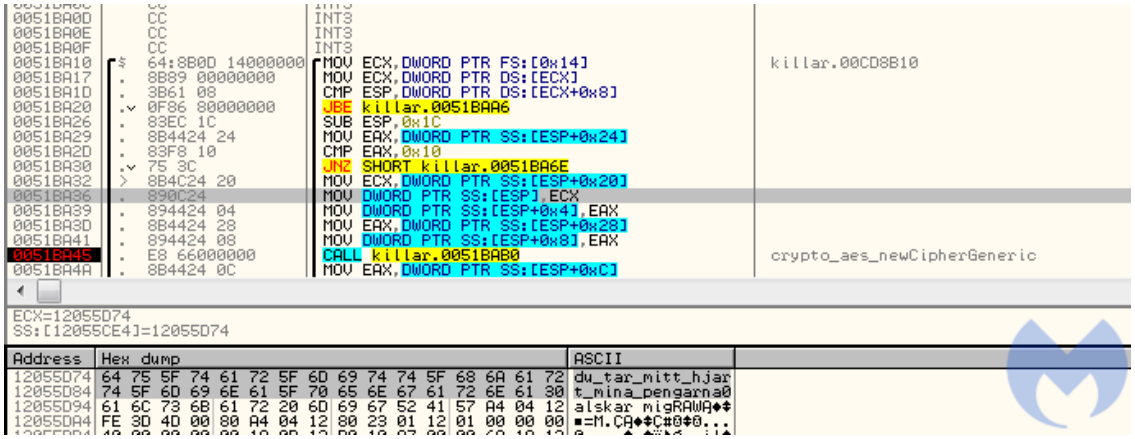
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

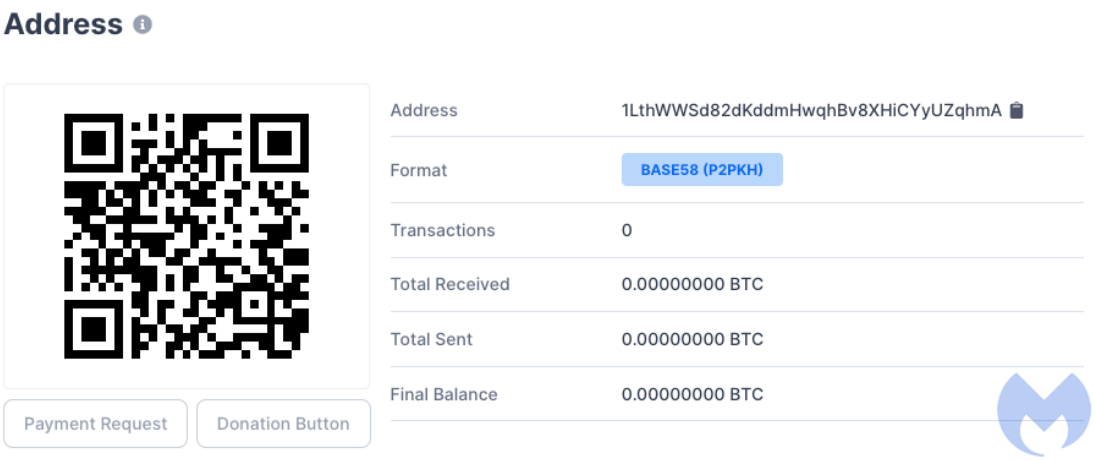
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

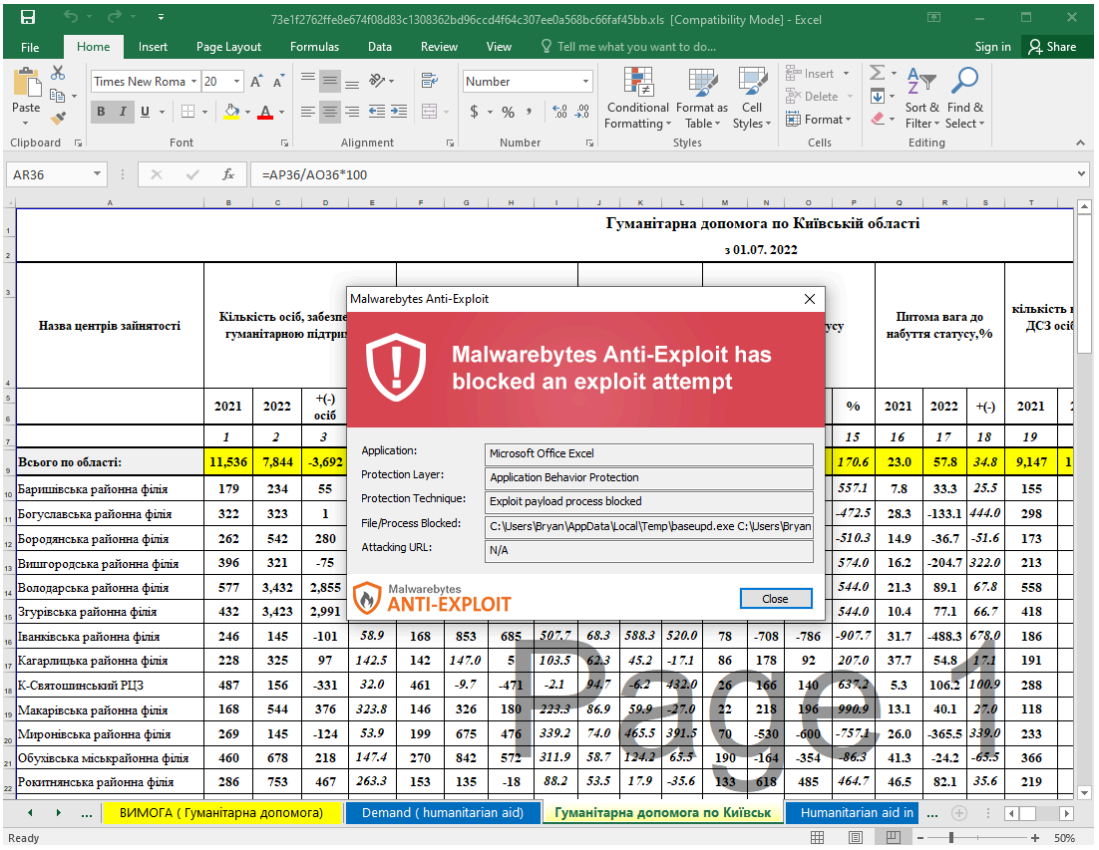
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

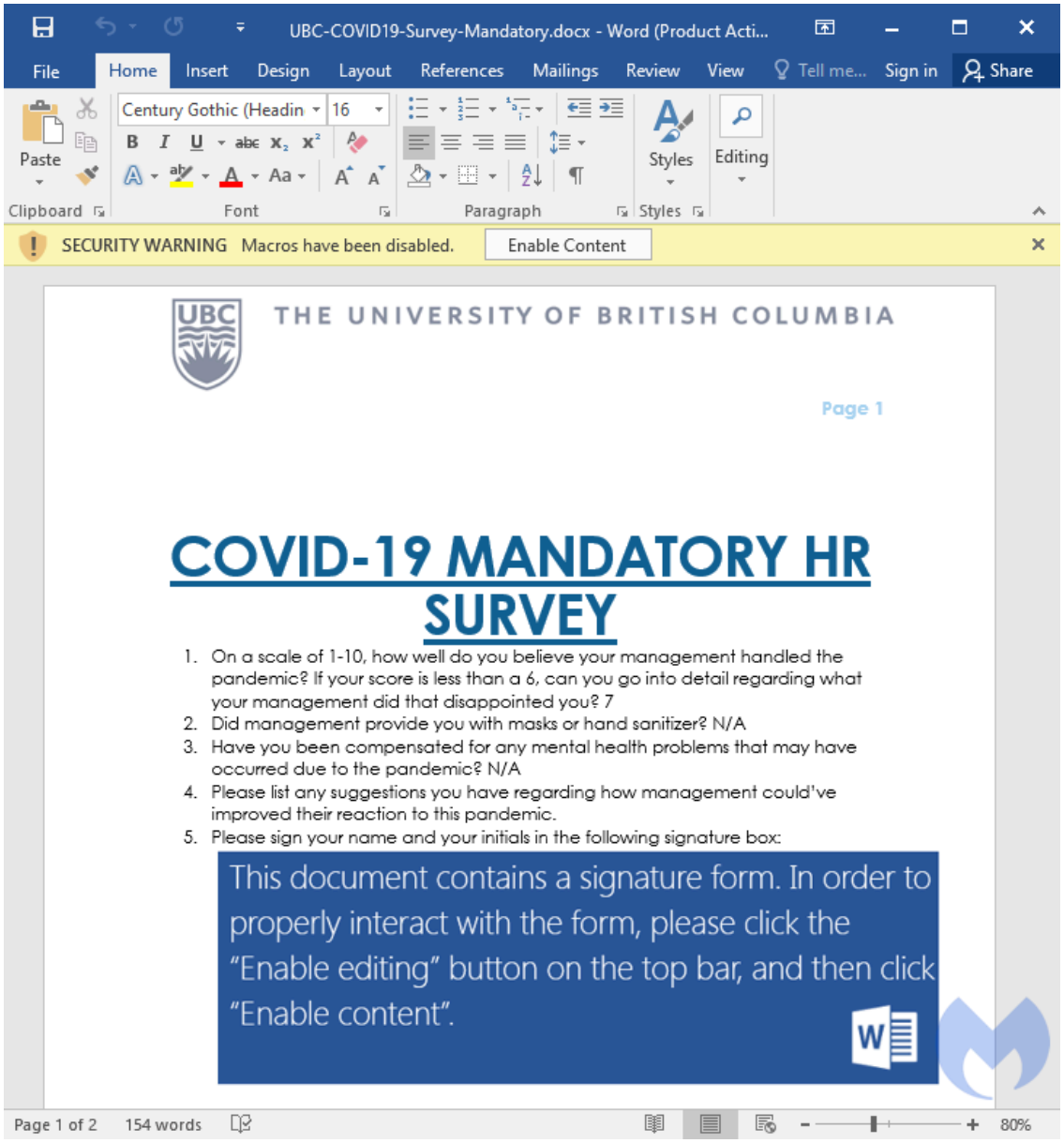
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

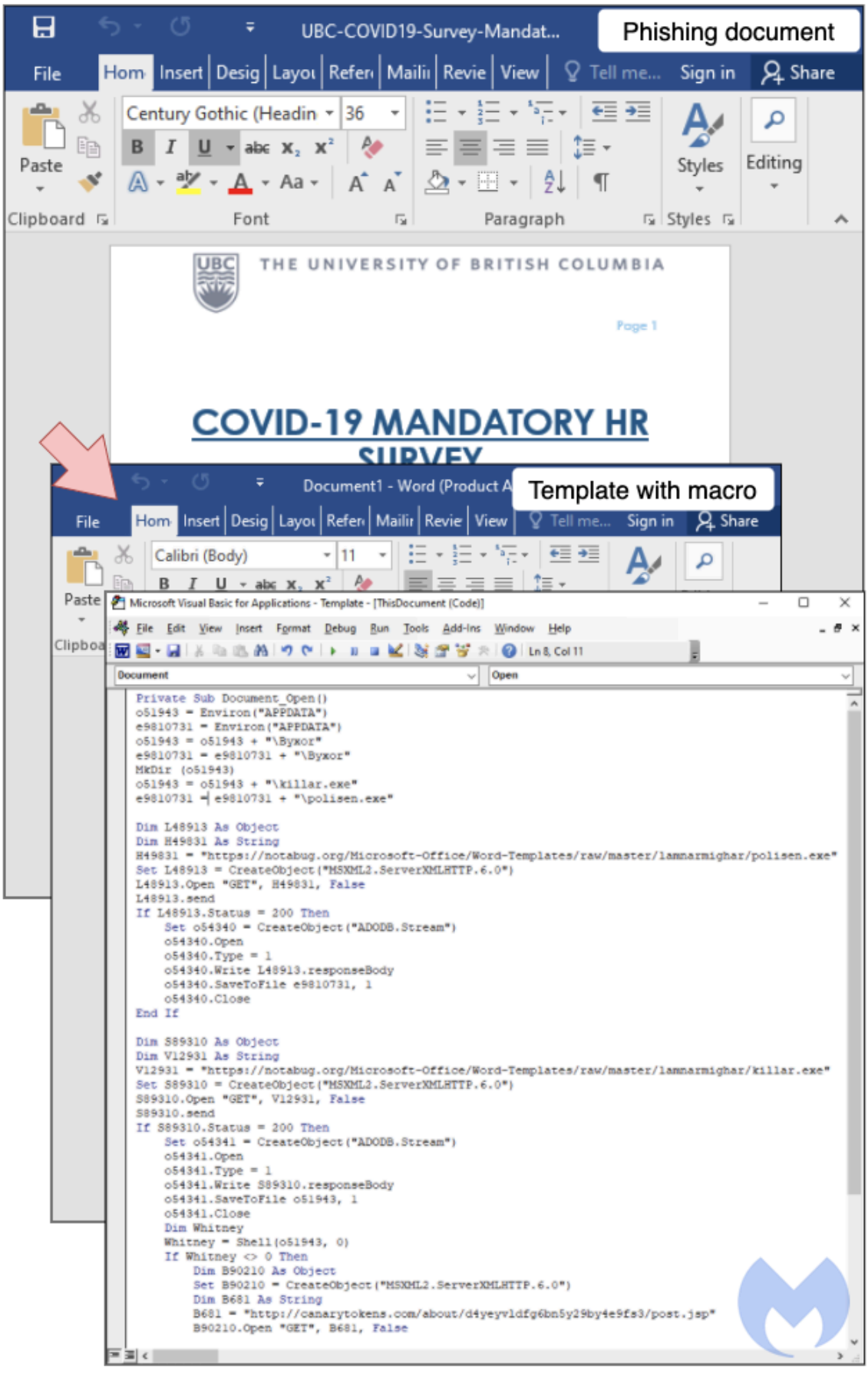
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

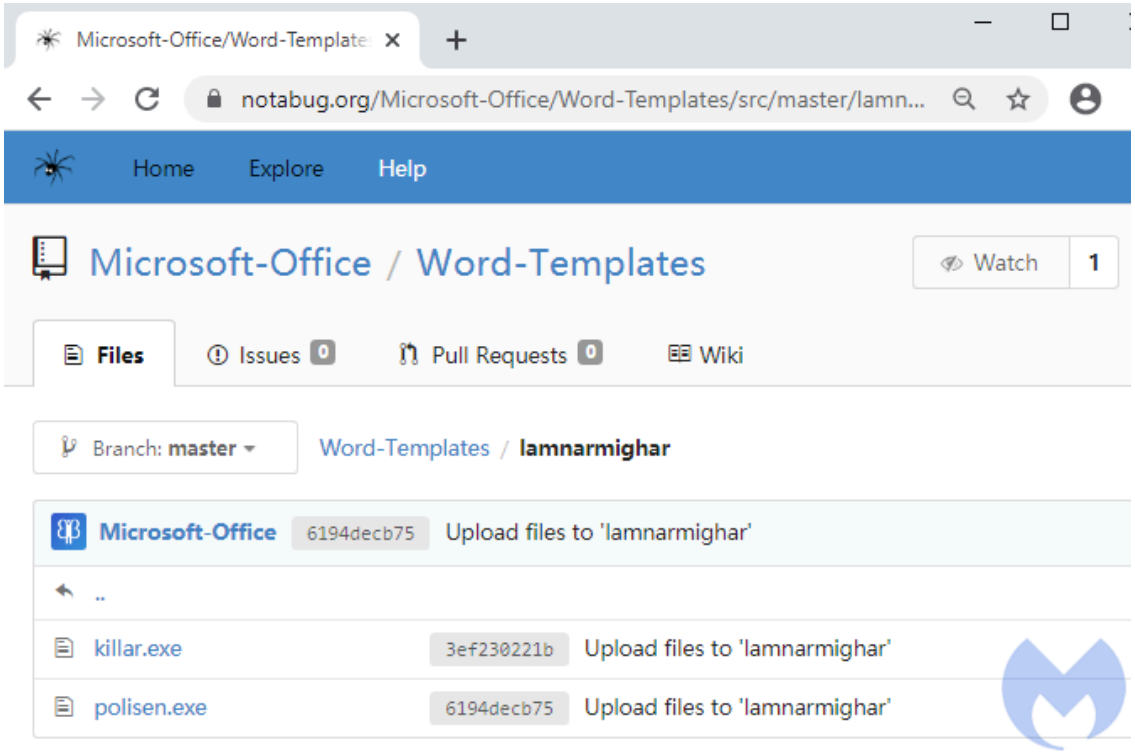
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



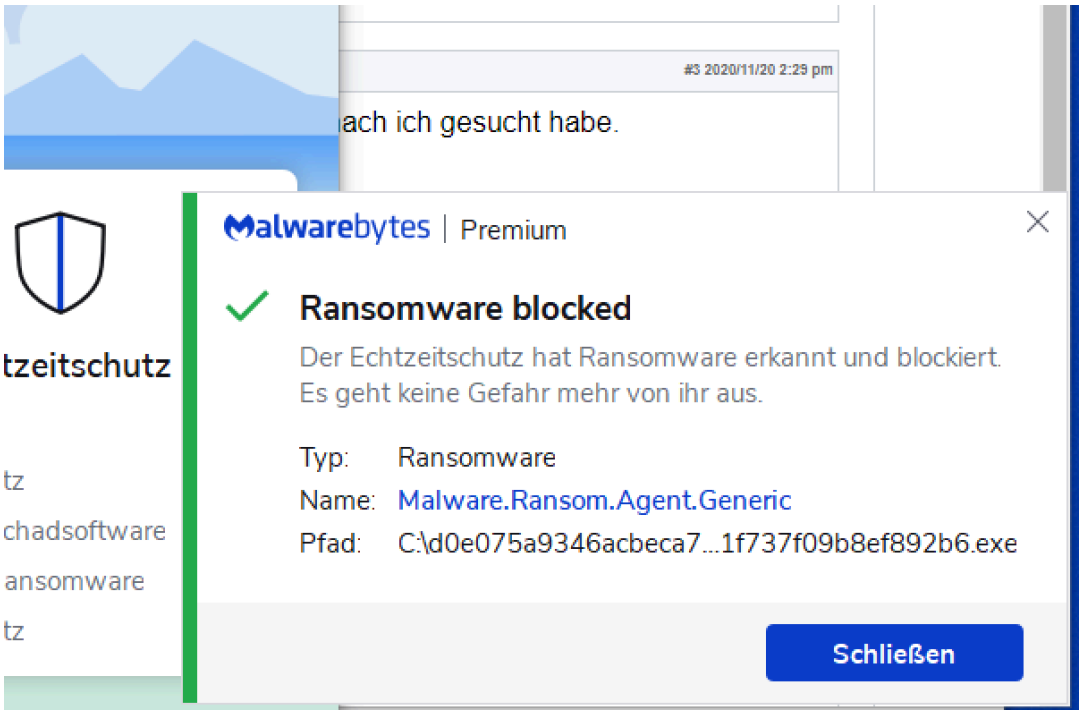
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

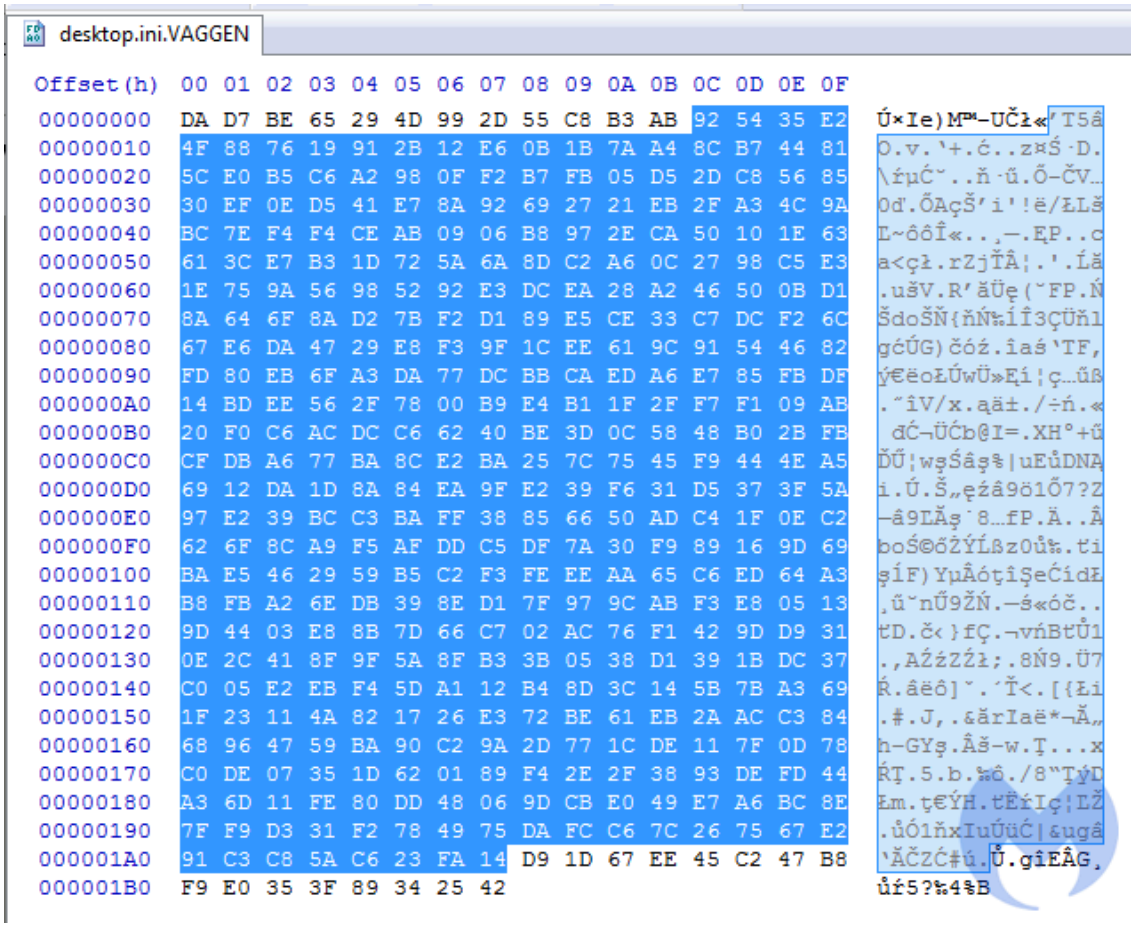
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



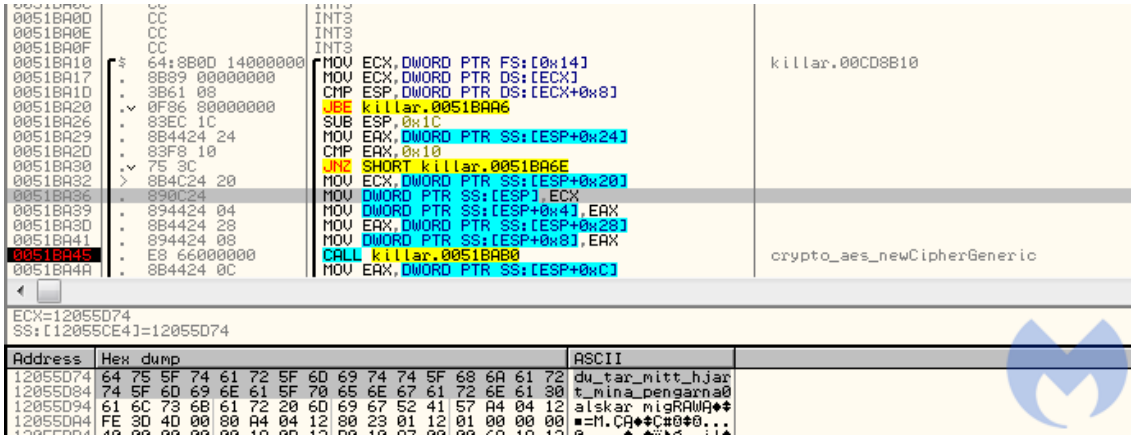
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

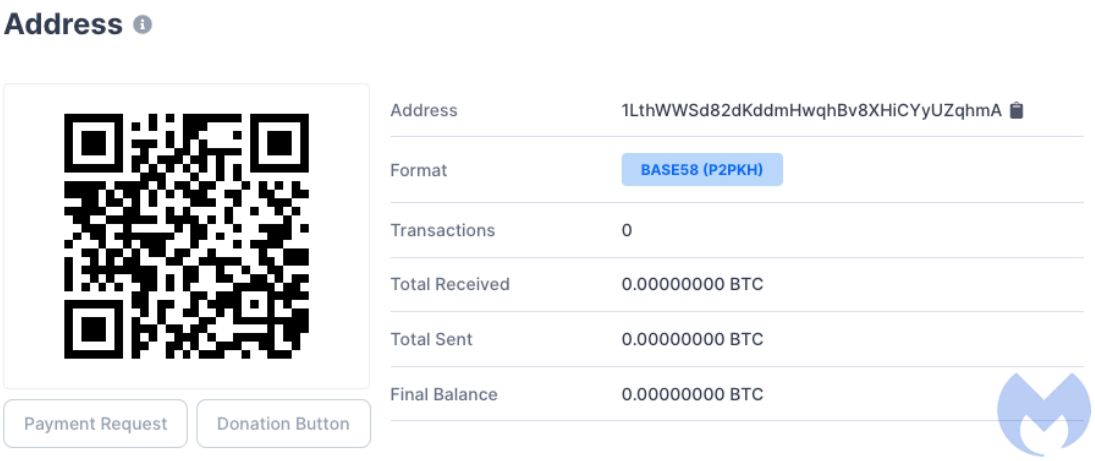
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

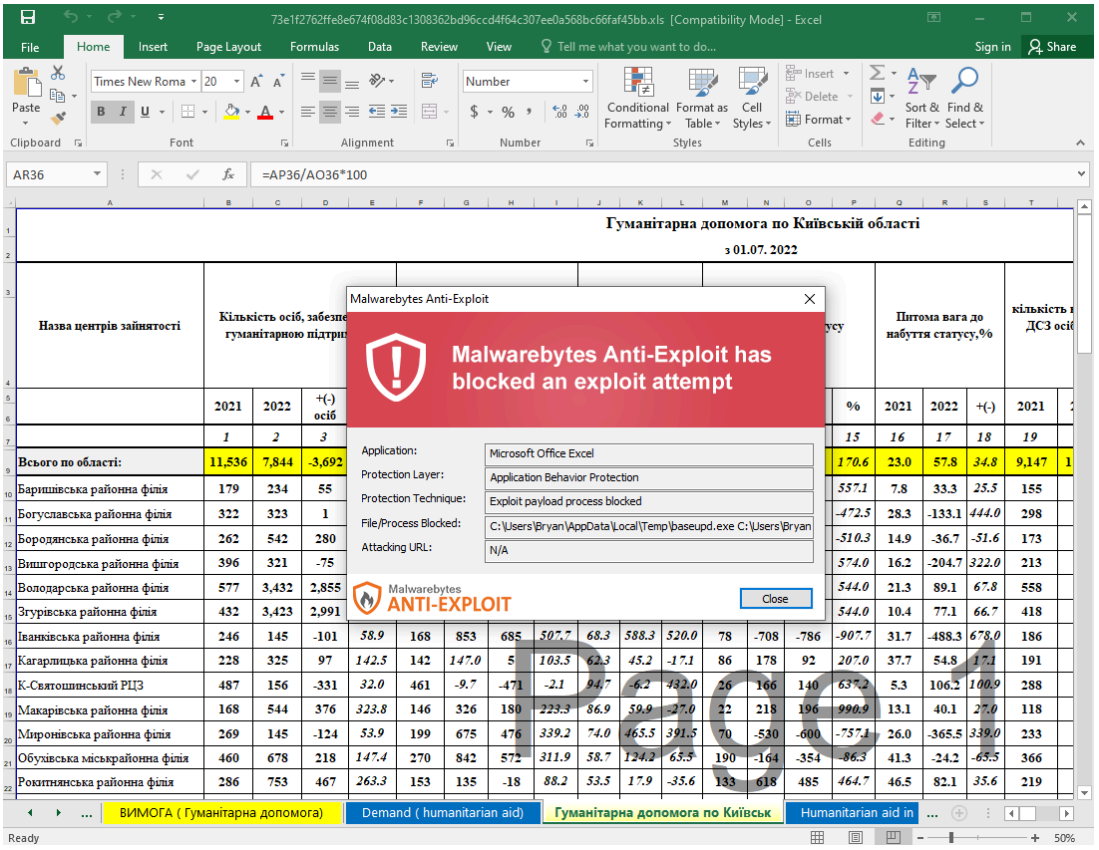
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

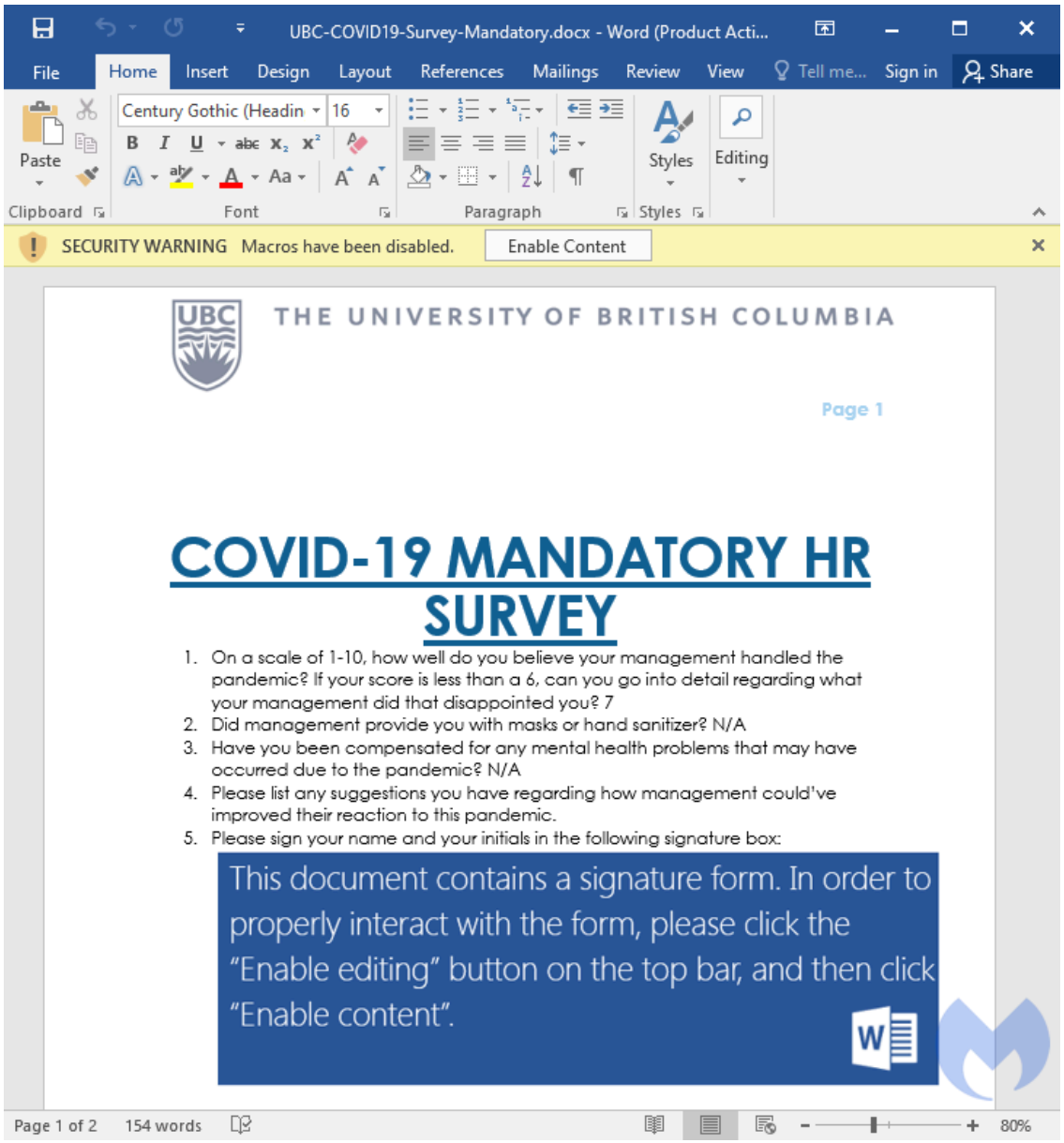
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

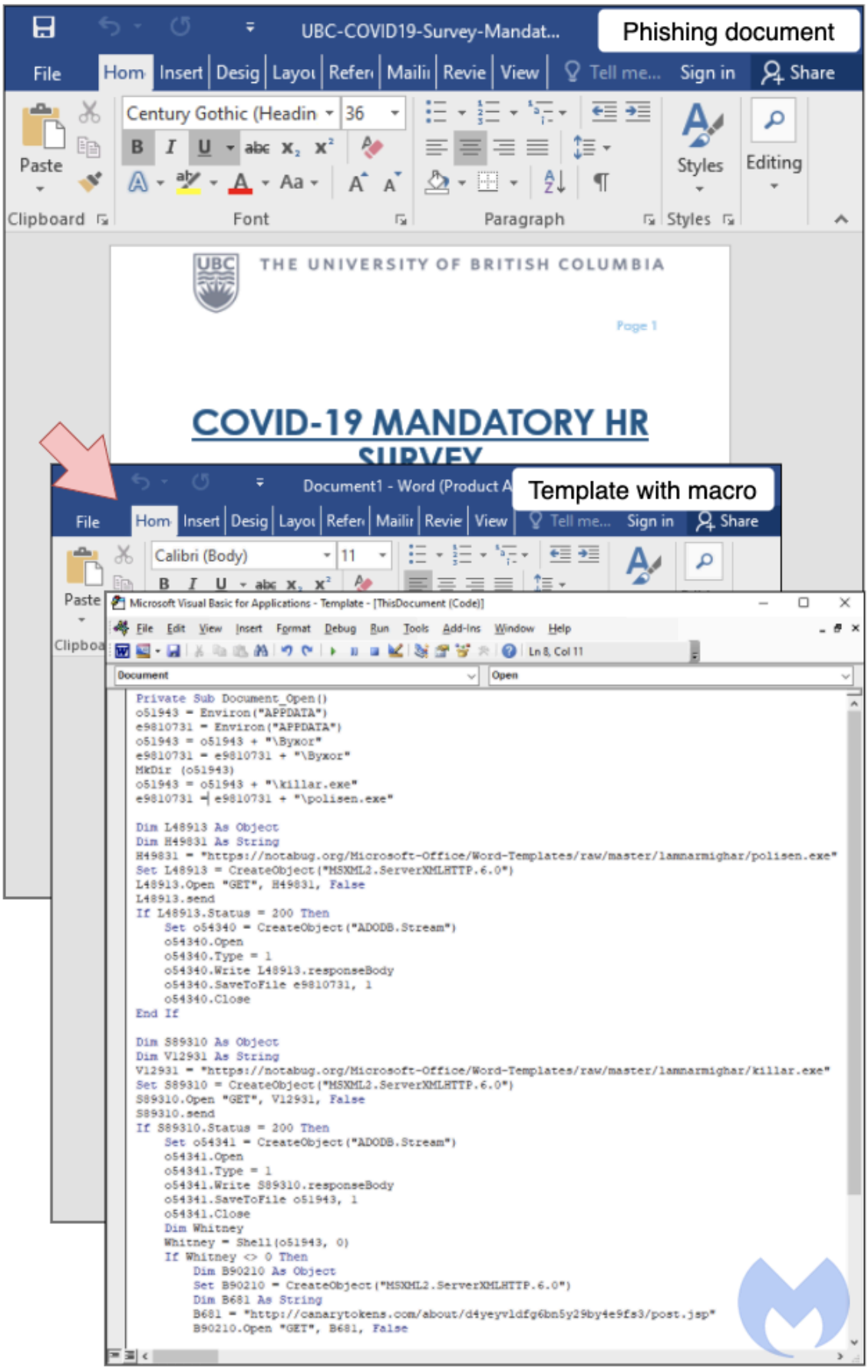
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

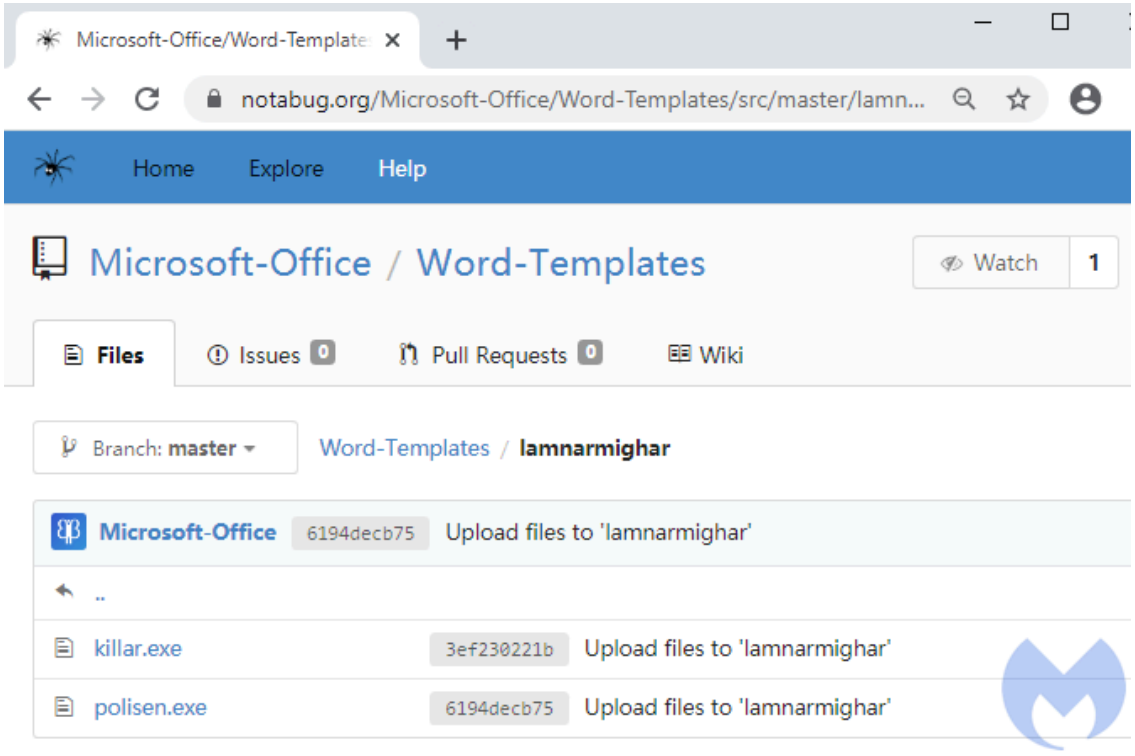
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



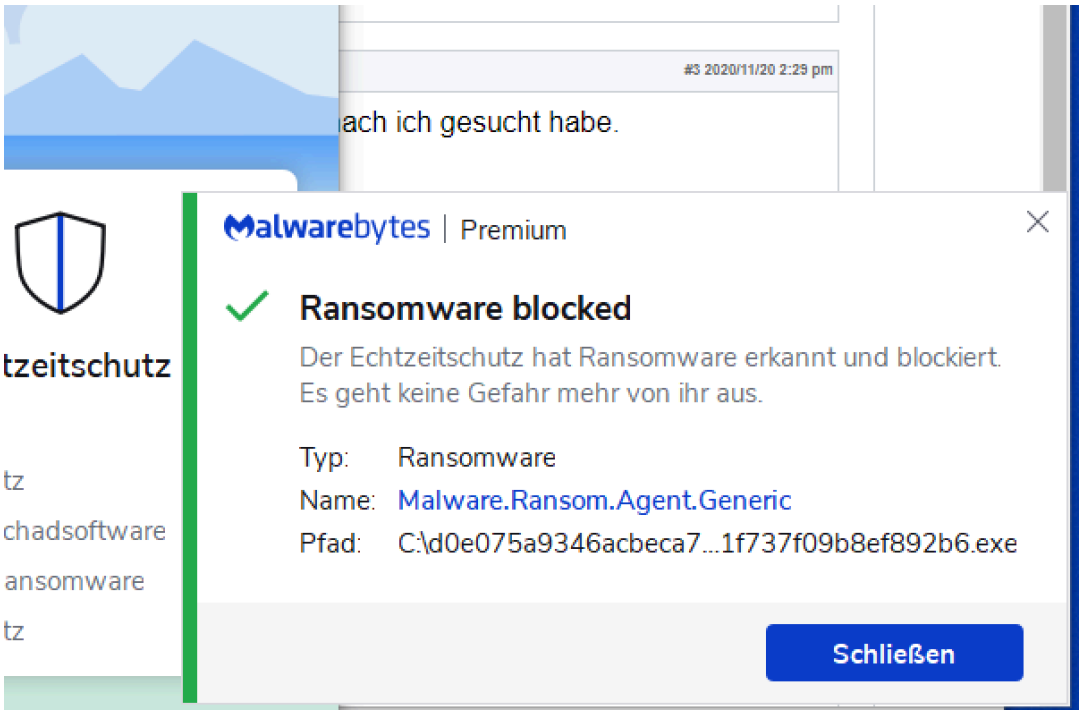
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

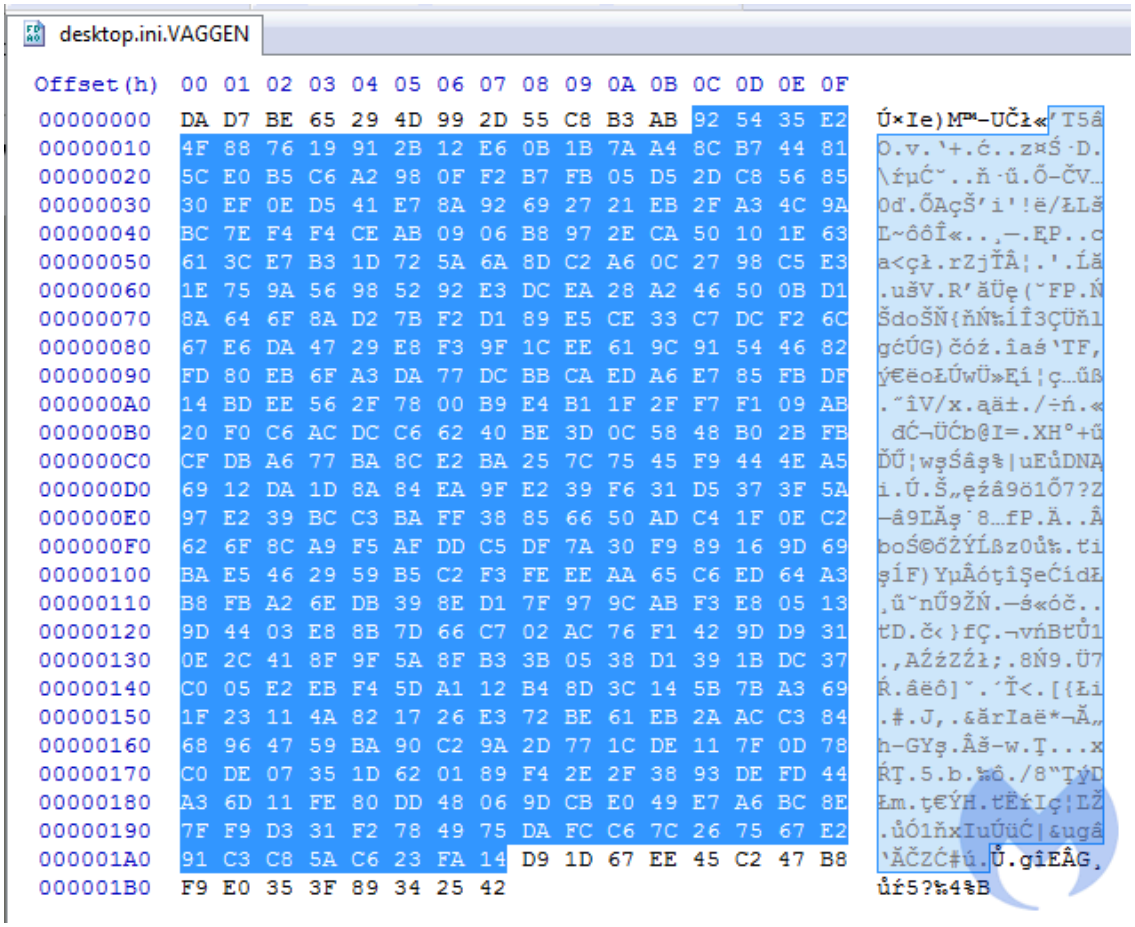
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

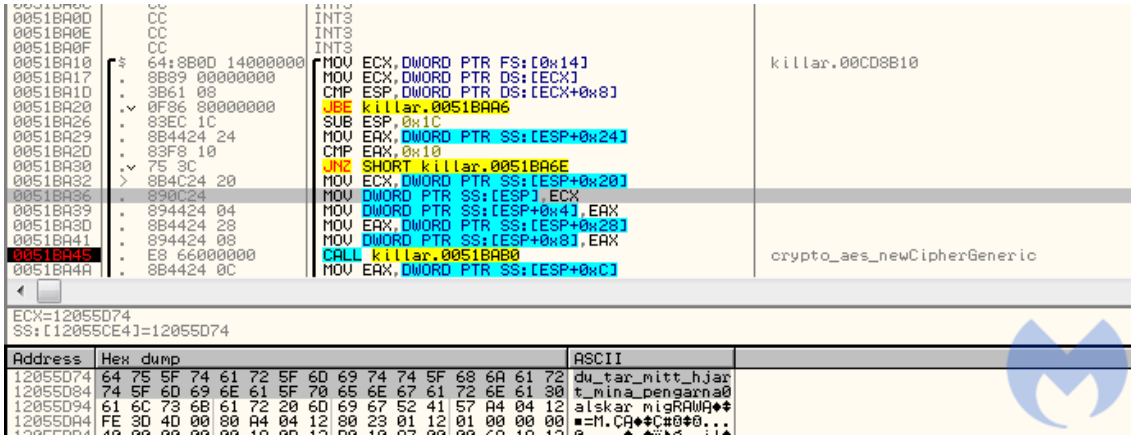


The content of the output file (with .VAGGEN extension) contains:

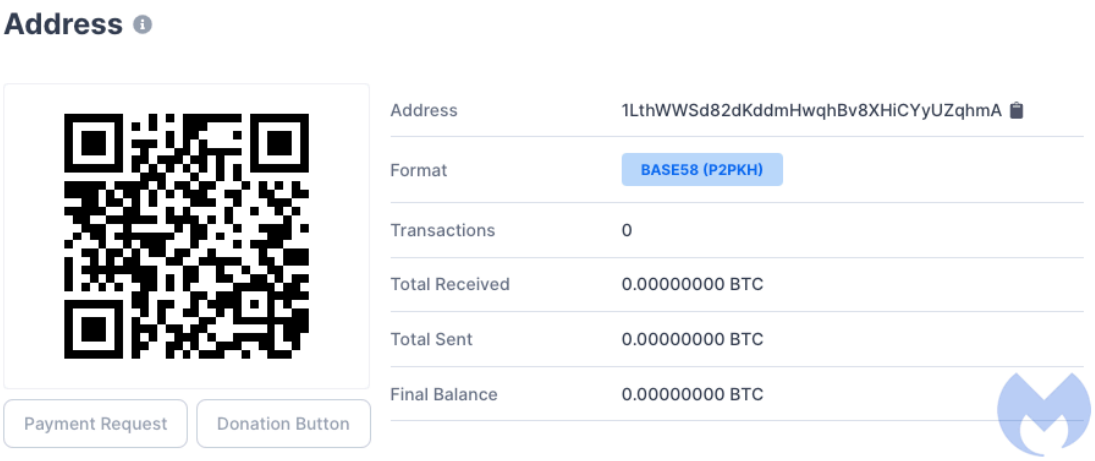
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

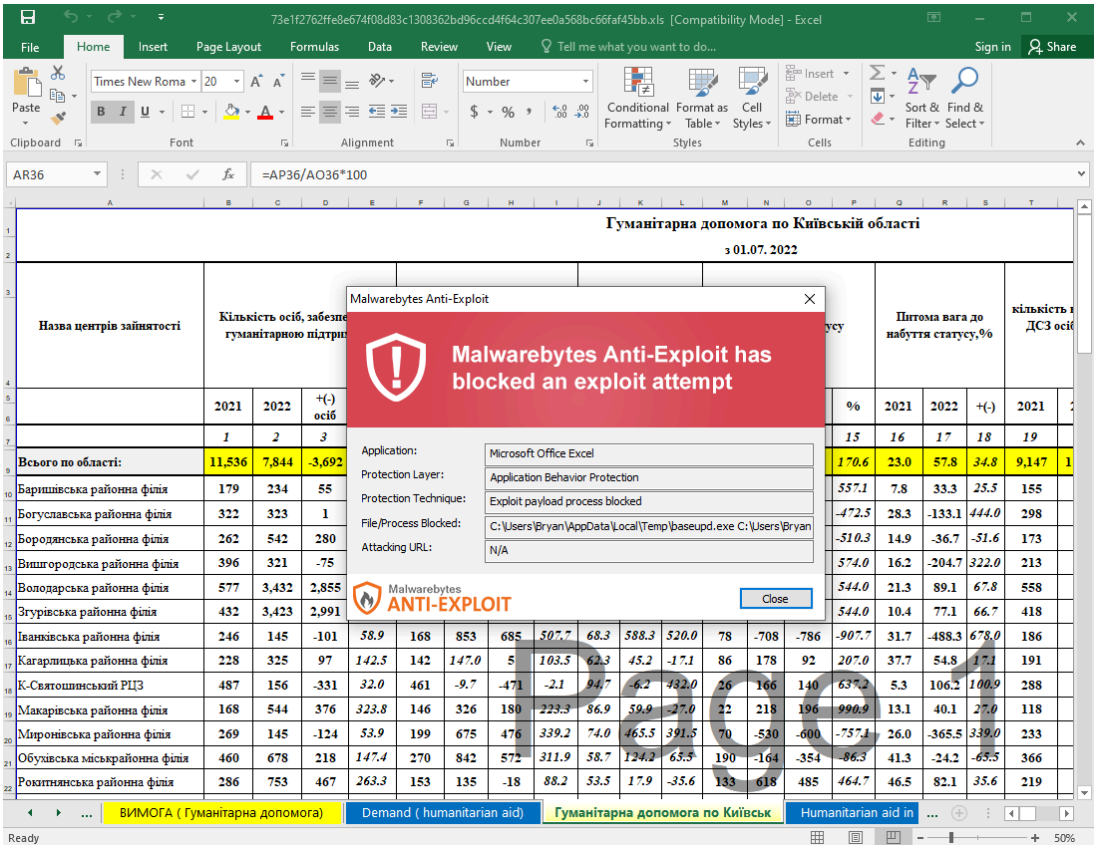
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

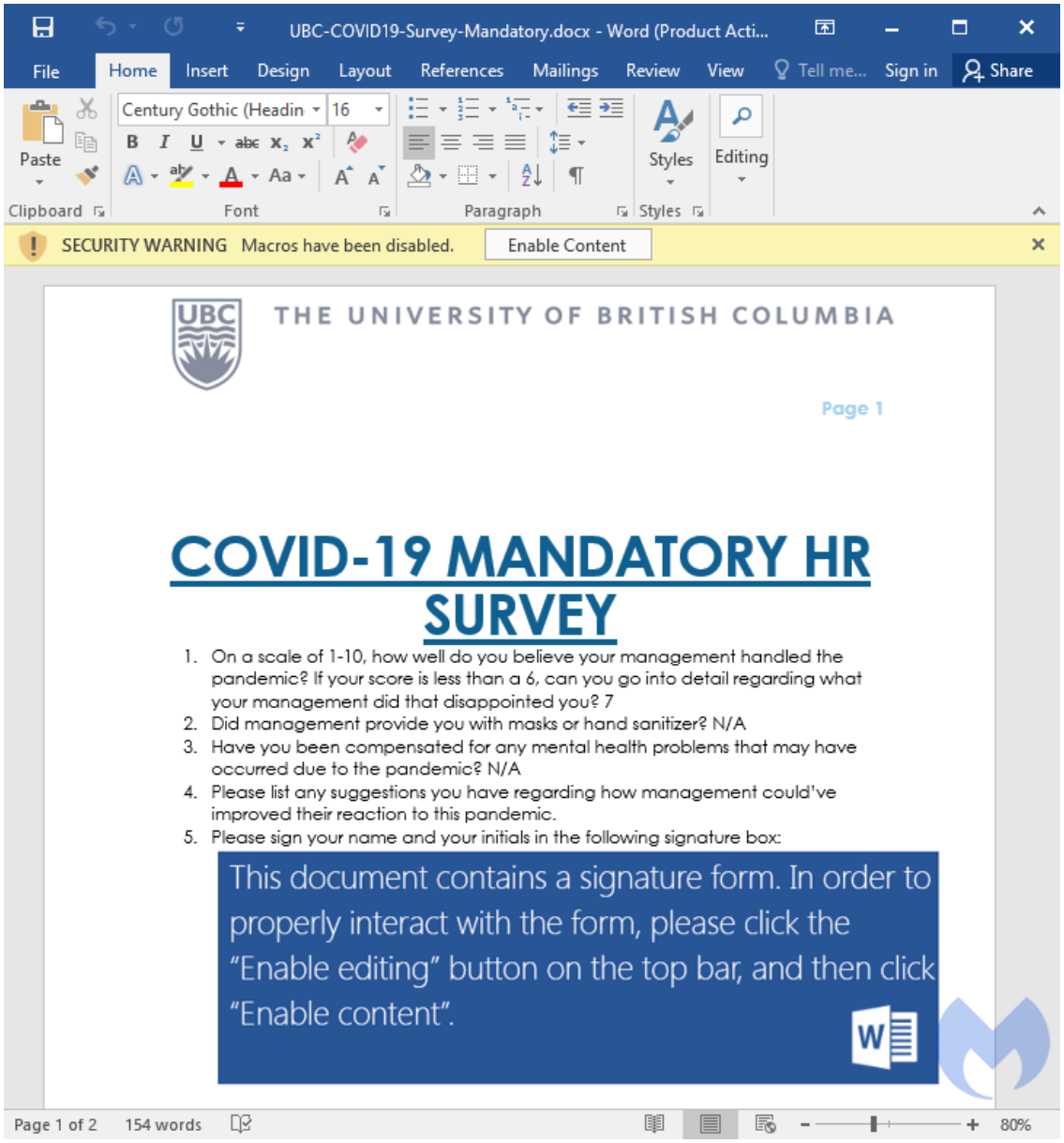
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

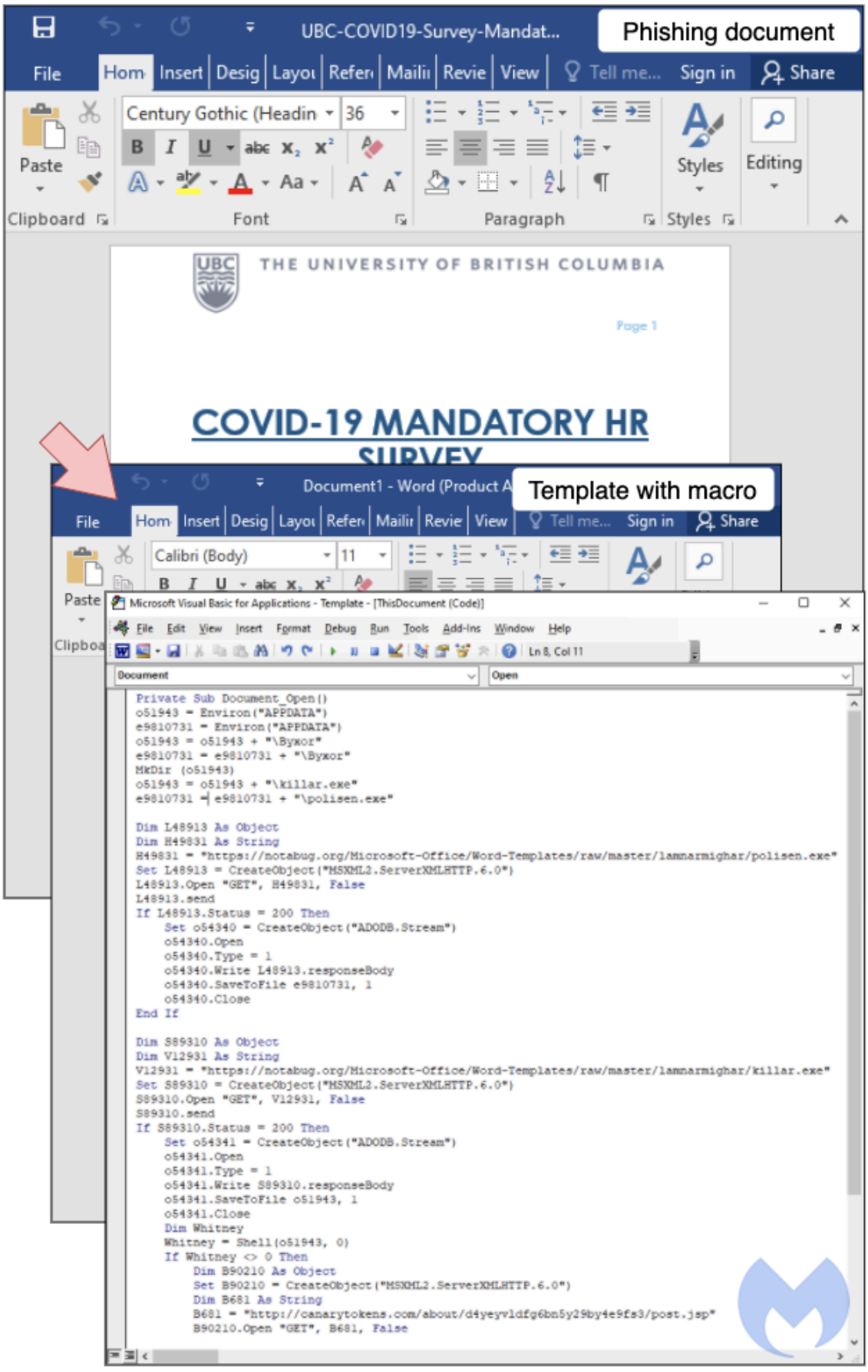
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

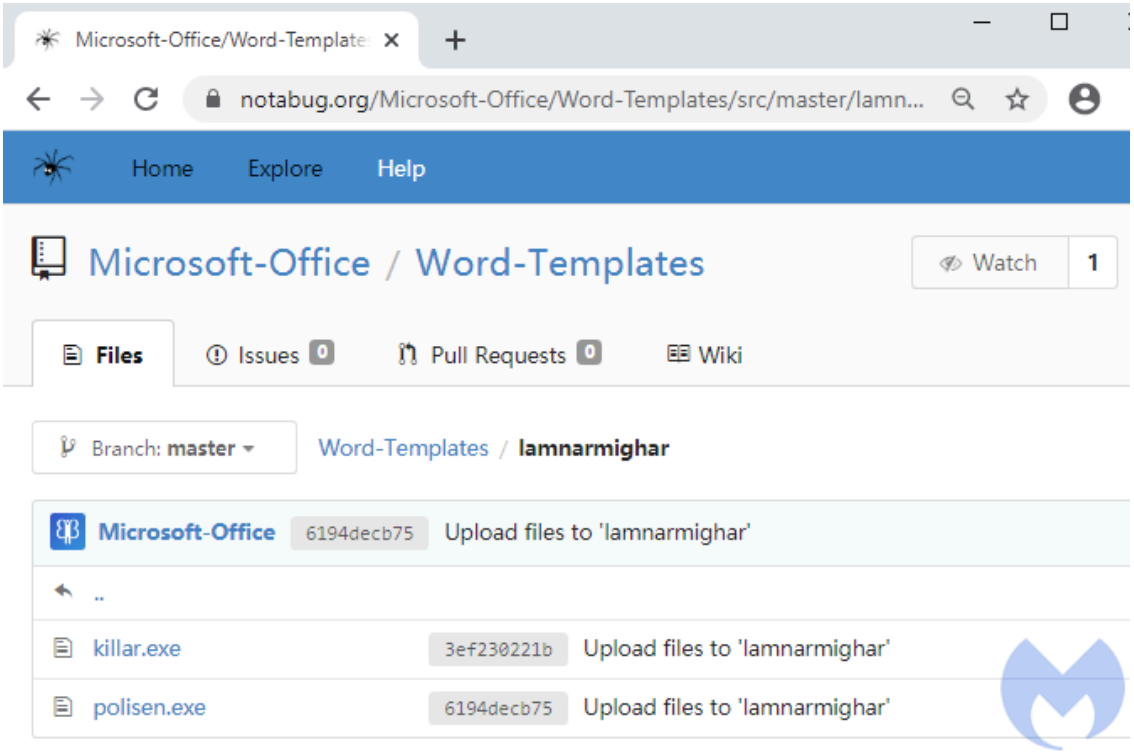
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



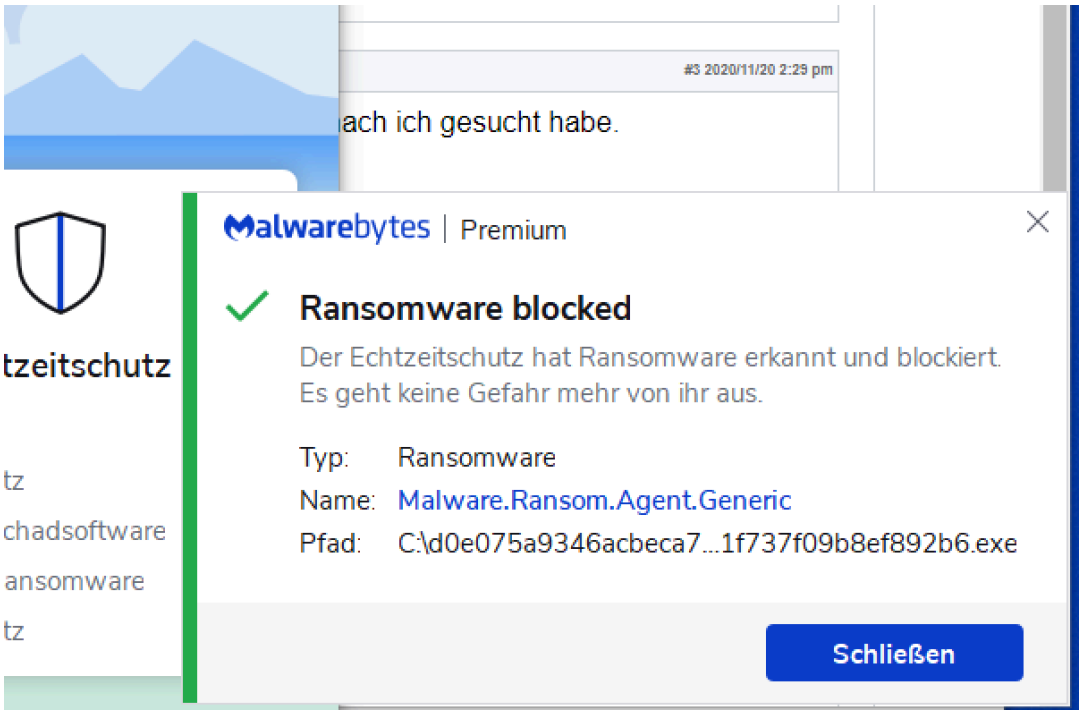
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

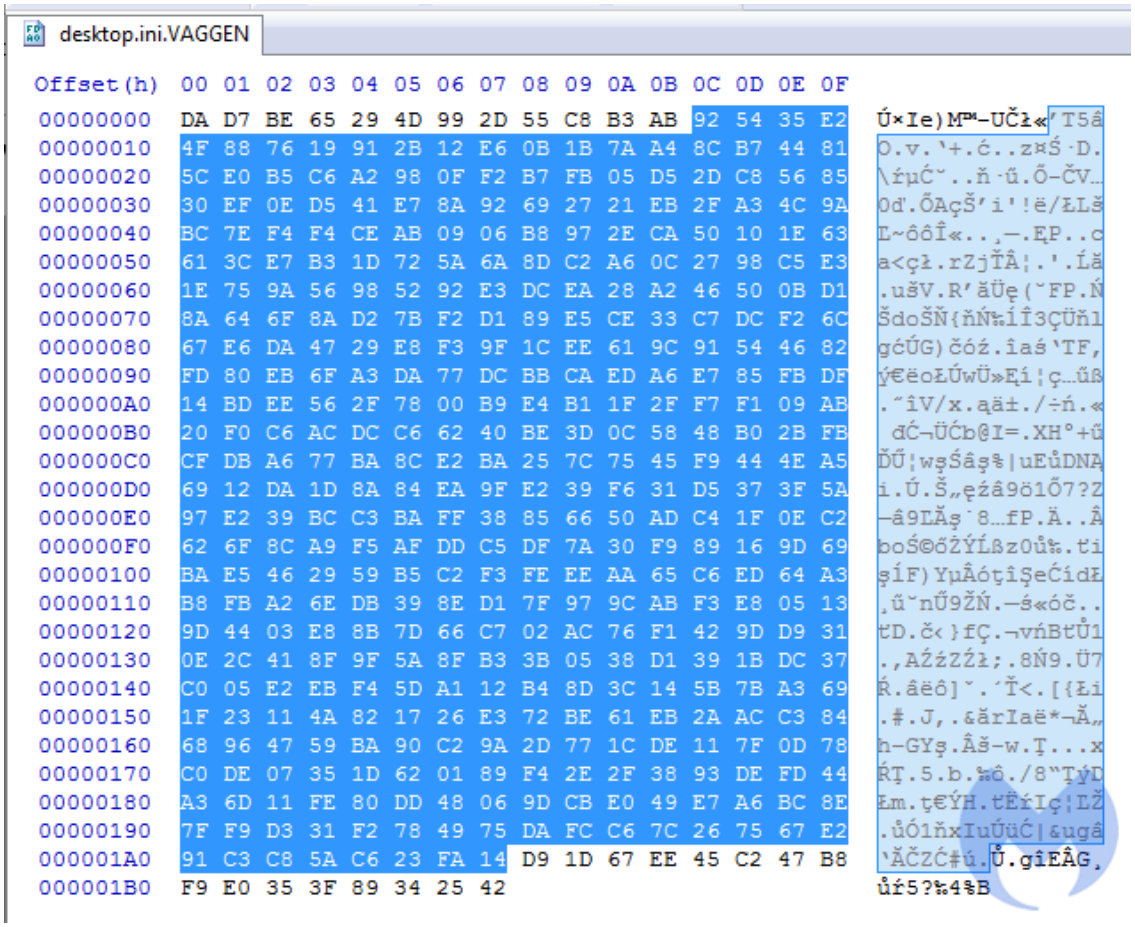
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

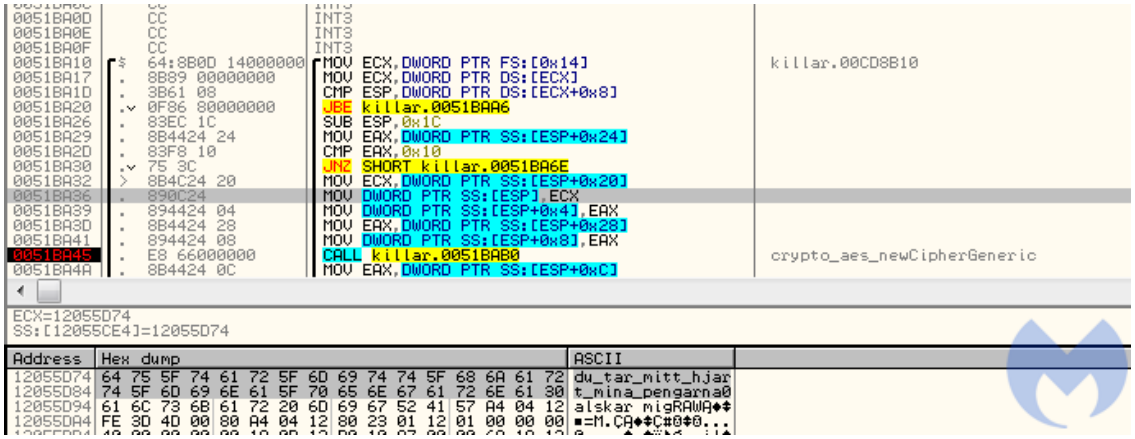


The content of the output file (with .VAGGEN extension) contains:

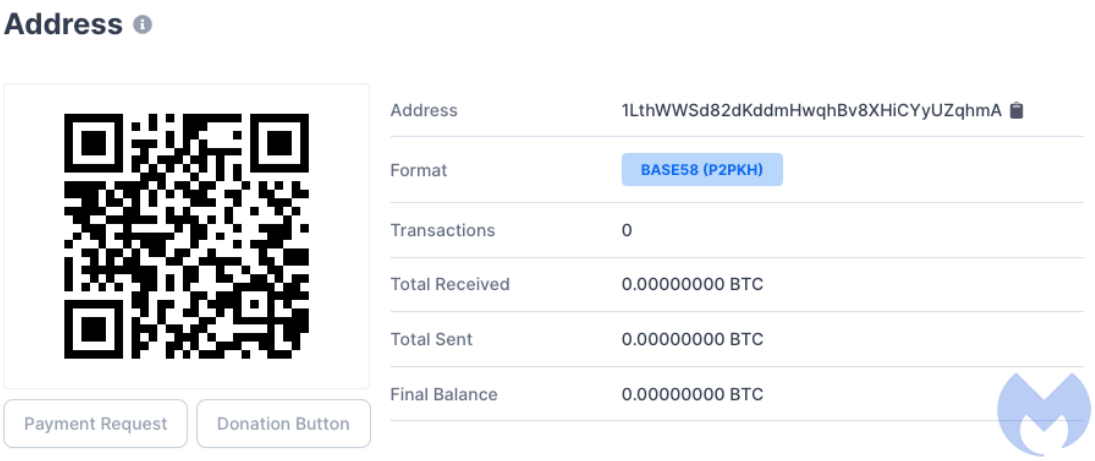
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

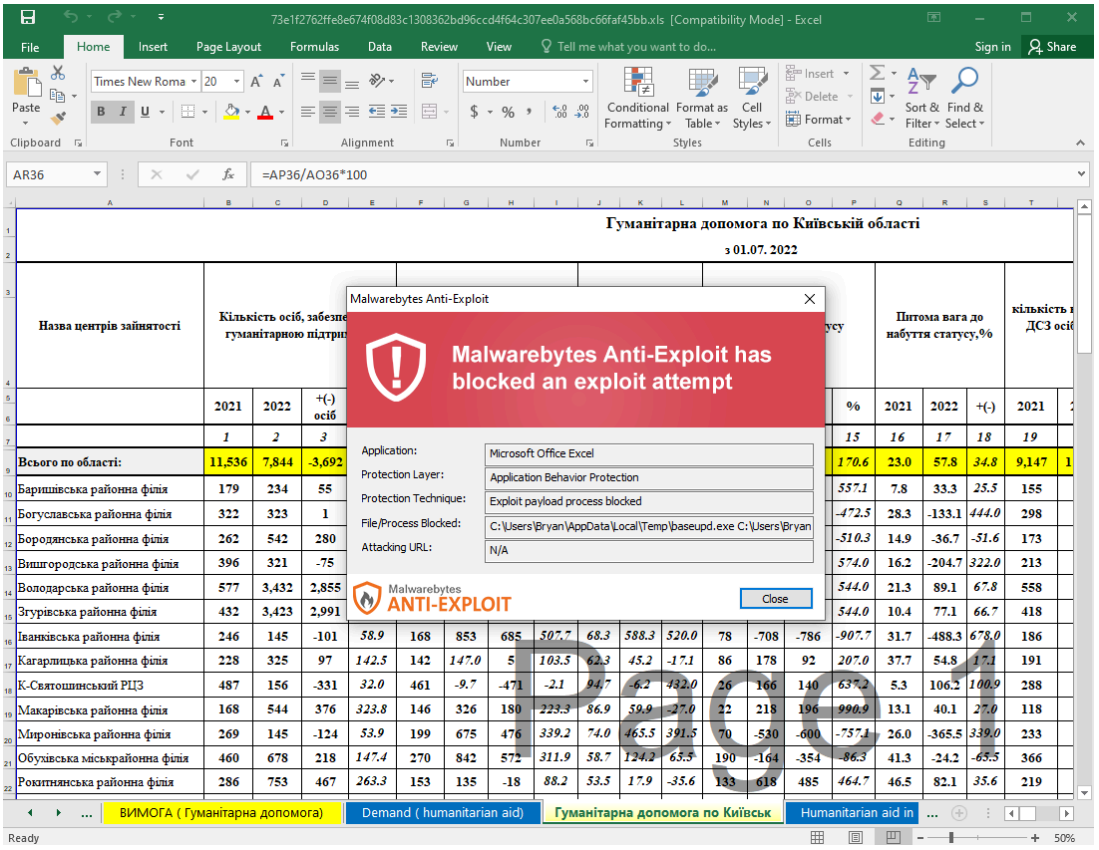
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

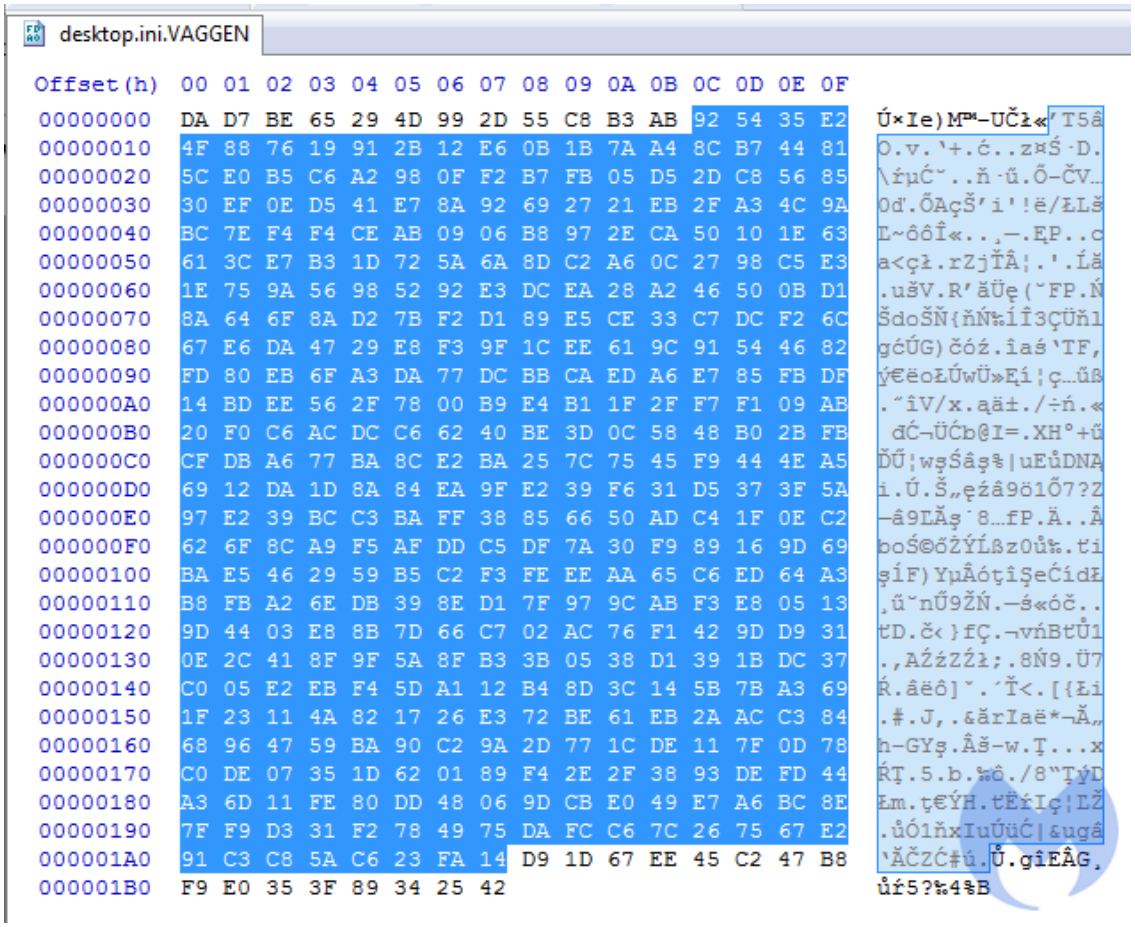
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

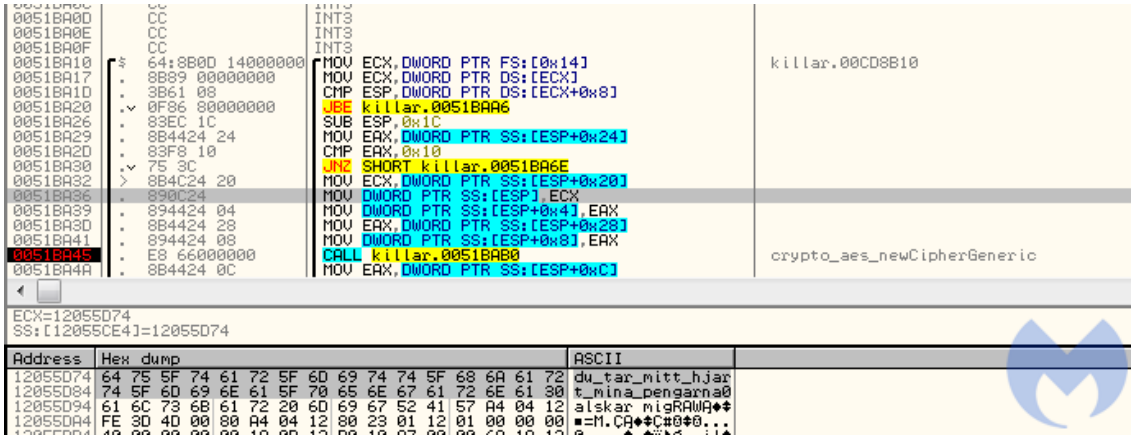


The content of the output file (with .VAGGEN extension) contains:

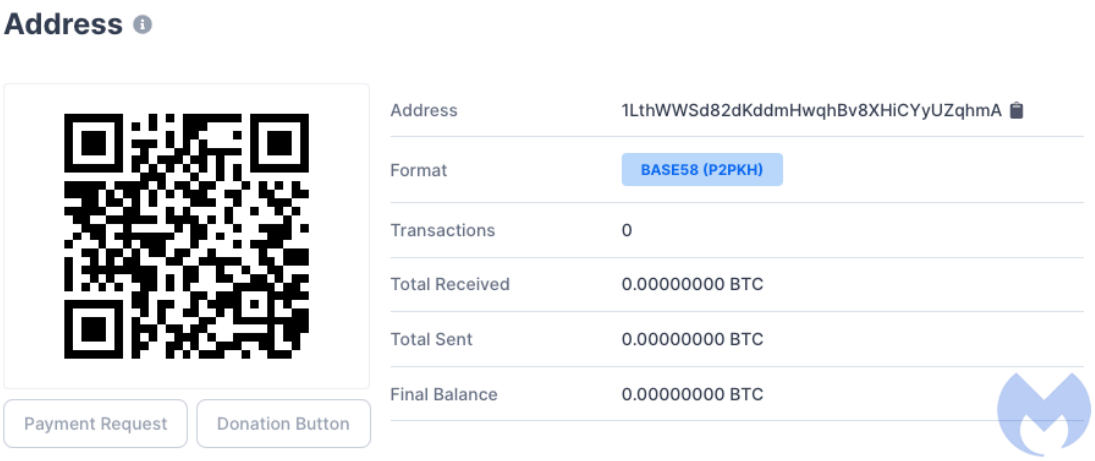
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

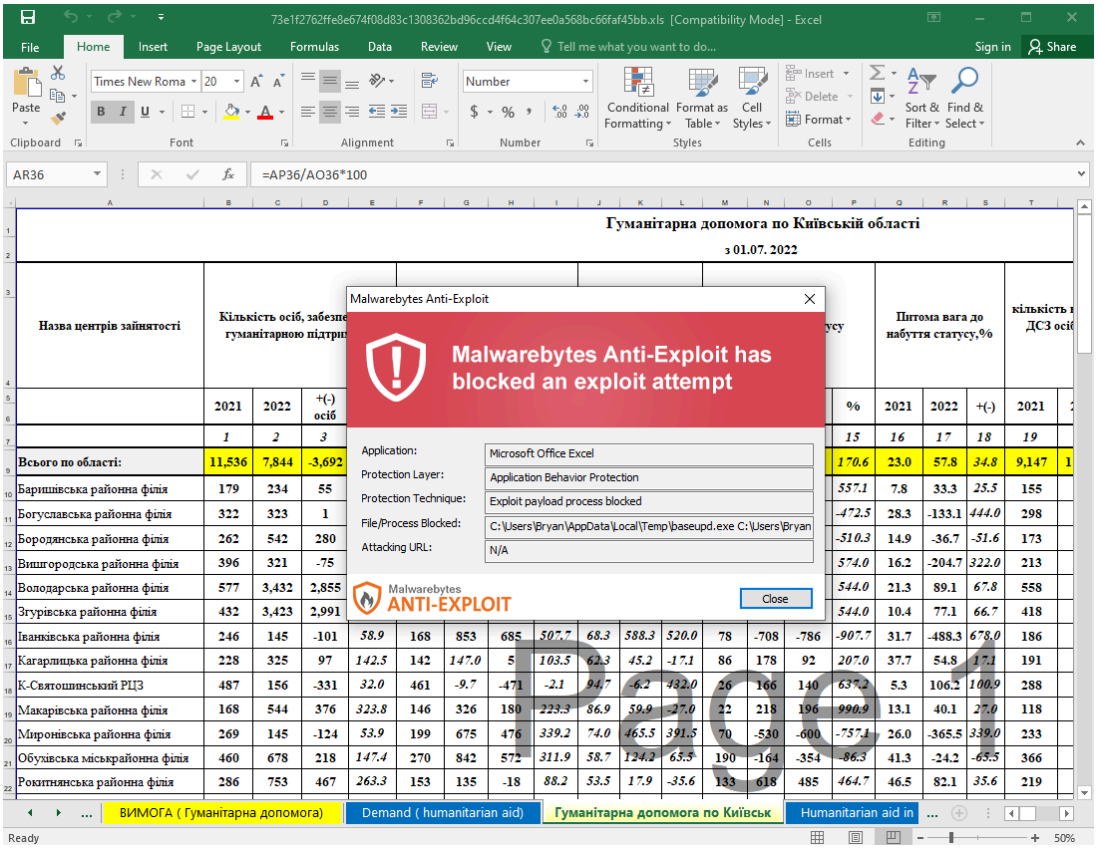
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

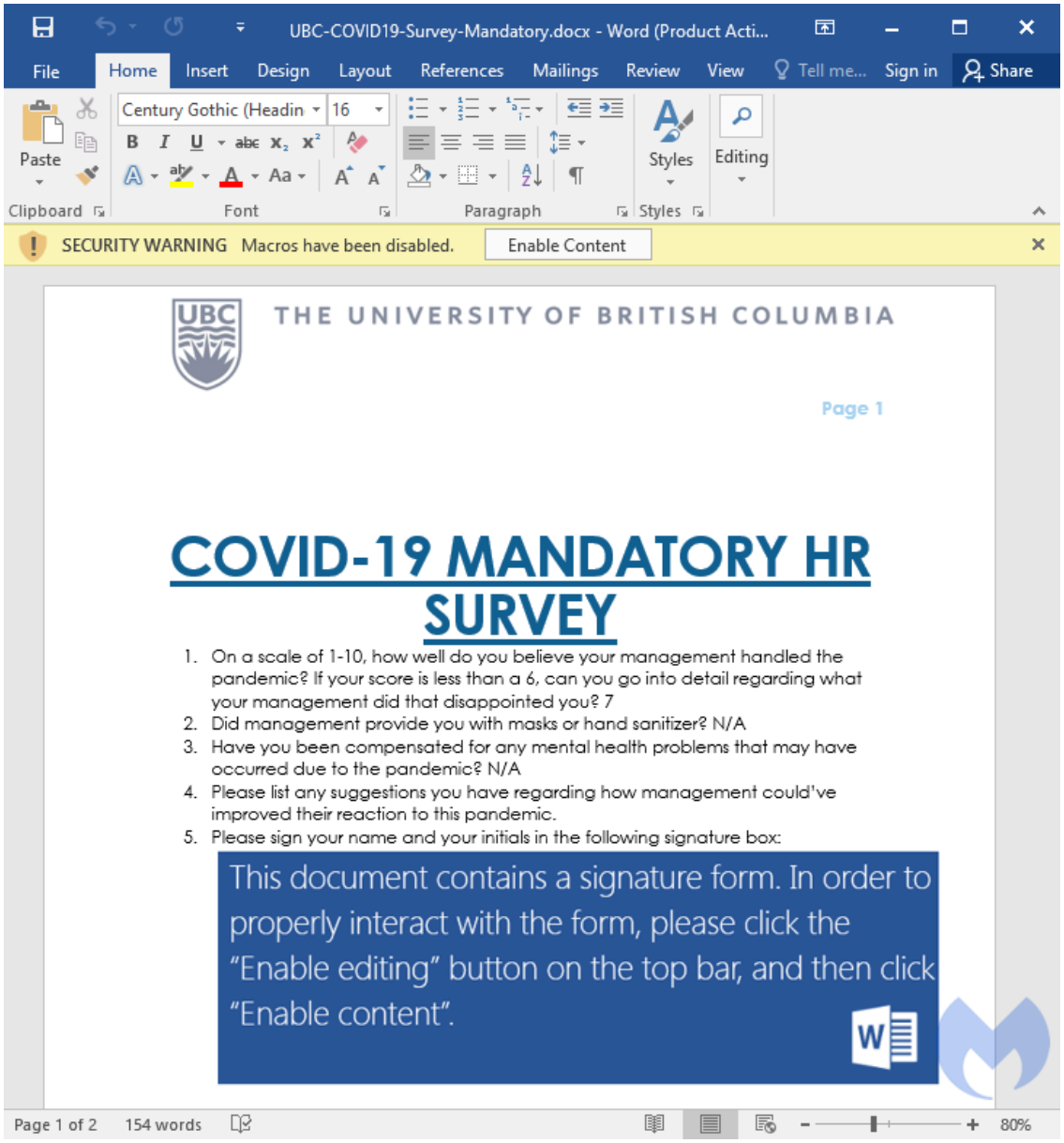
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

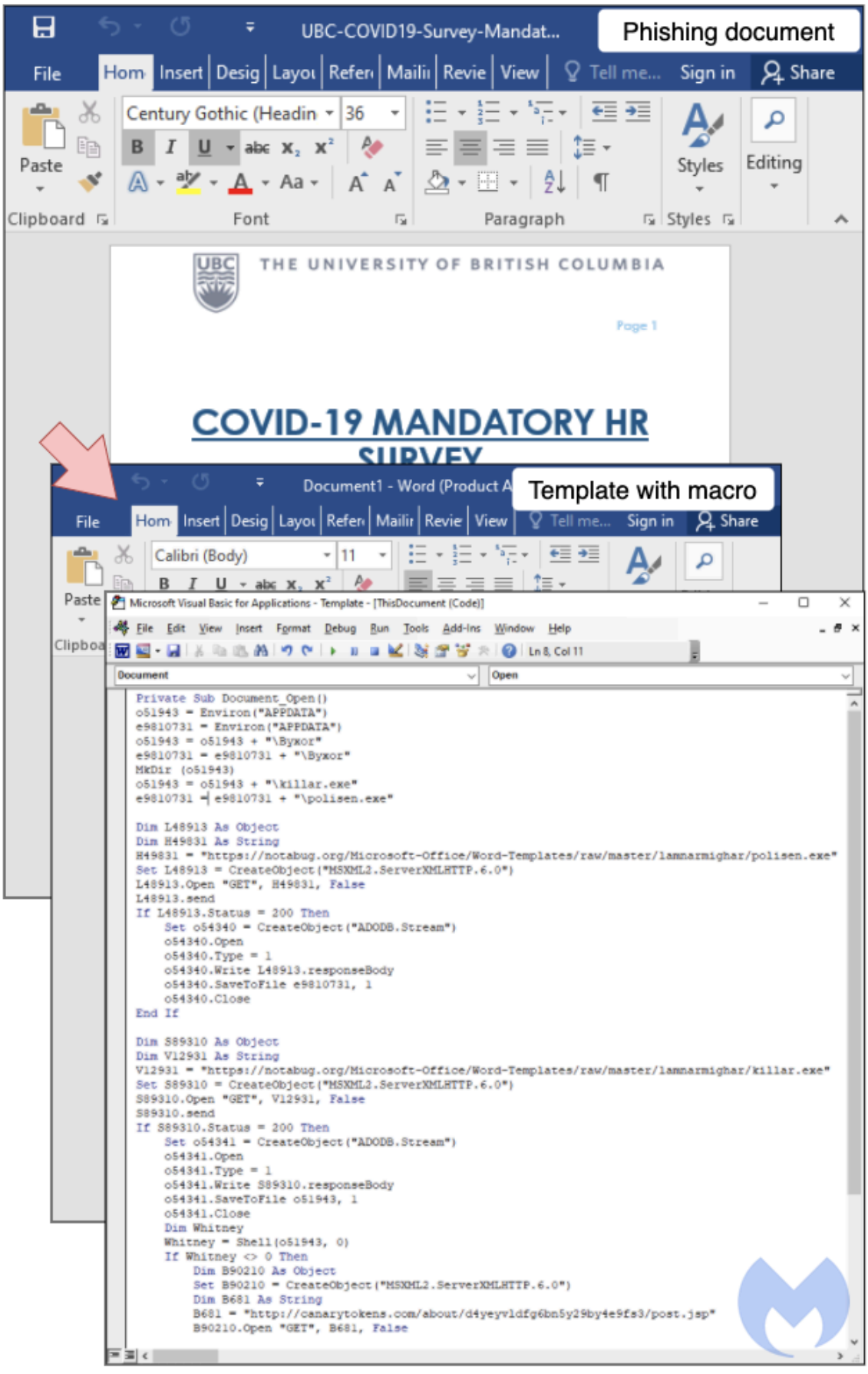
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

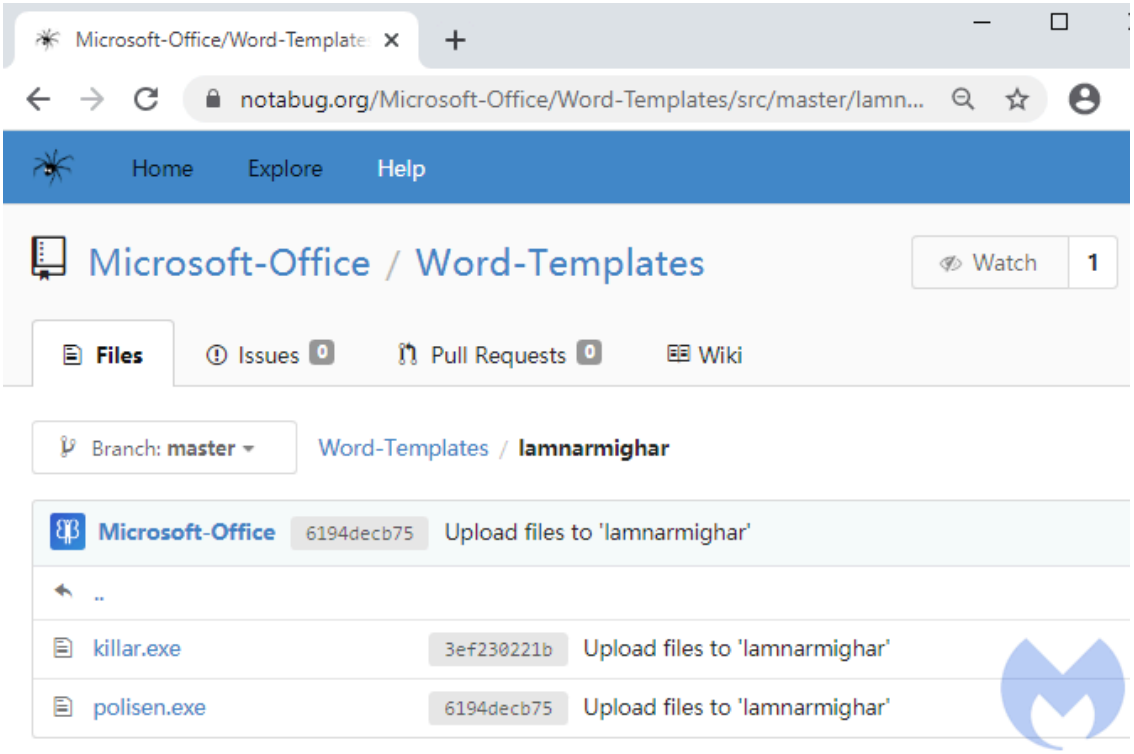
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



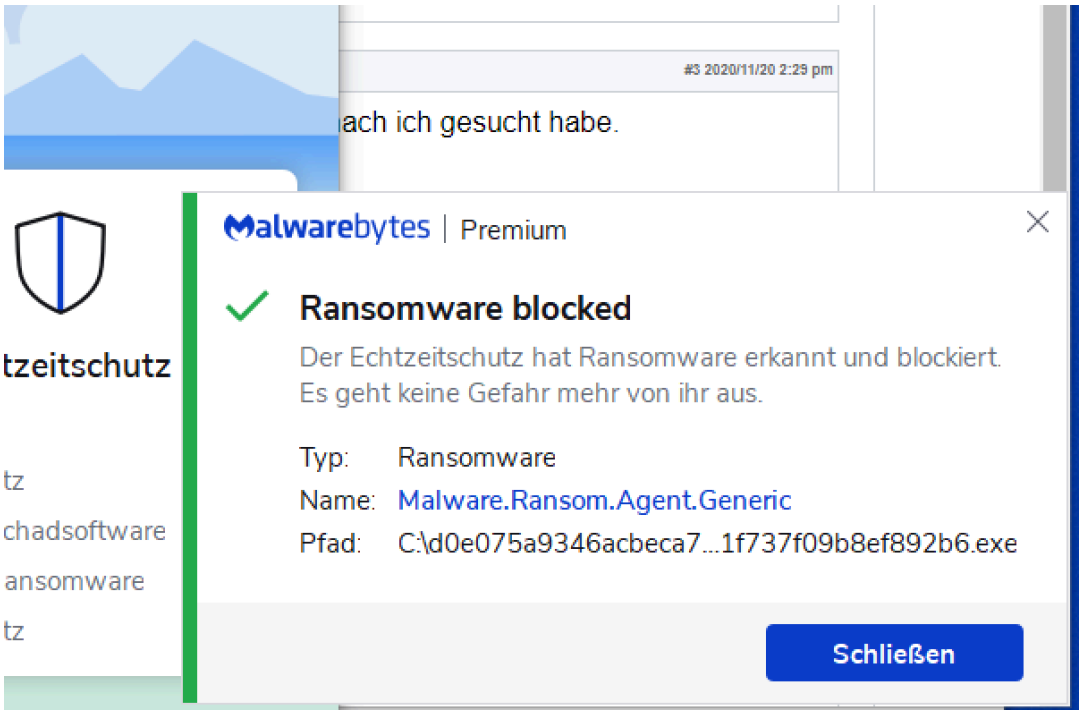
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

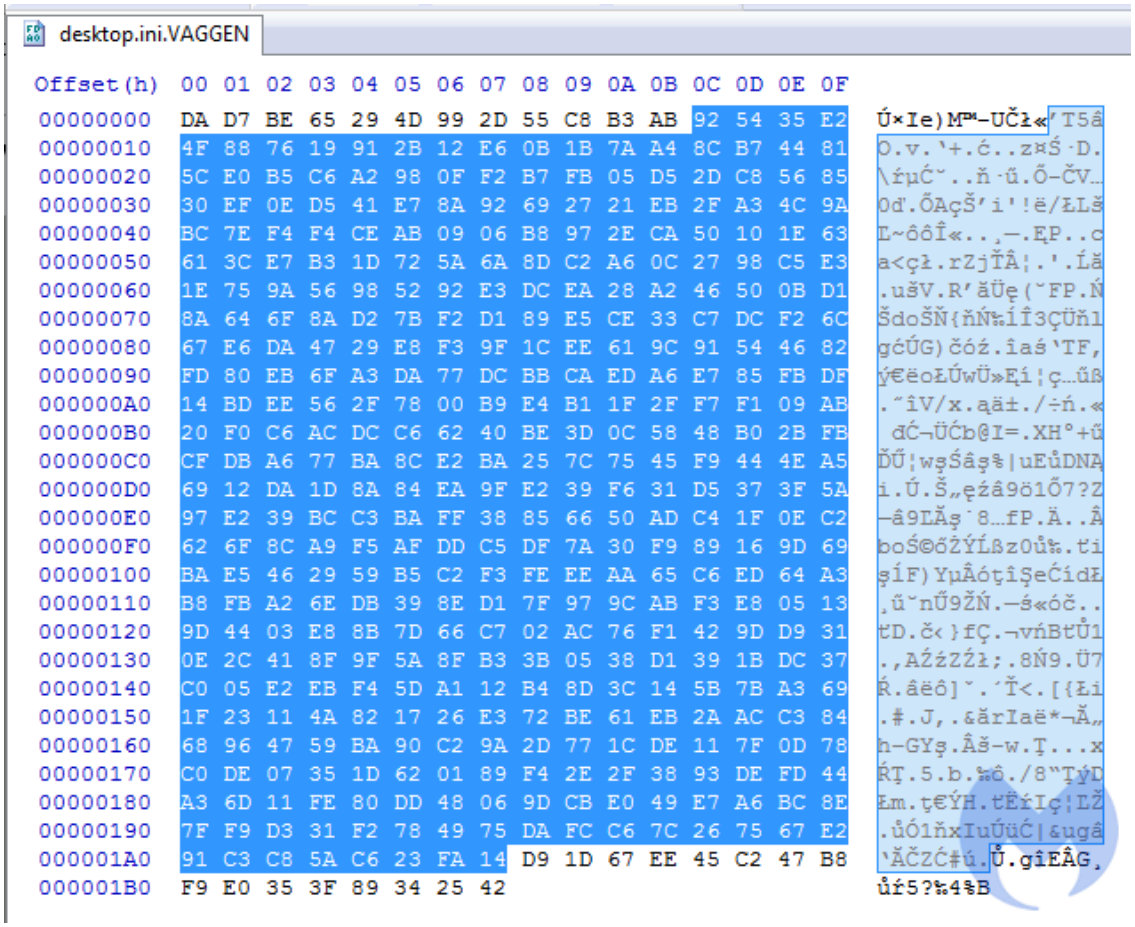
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



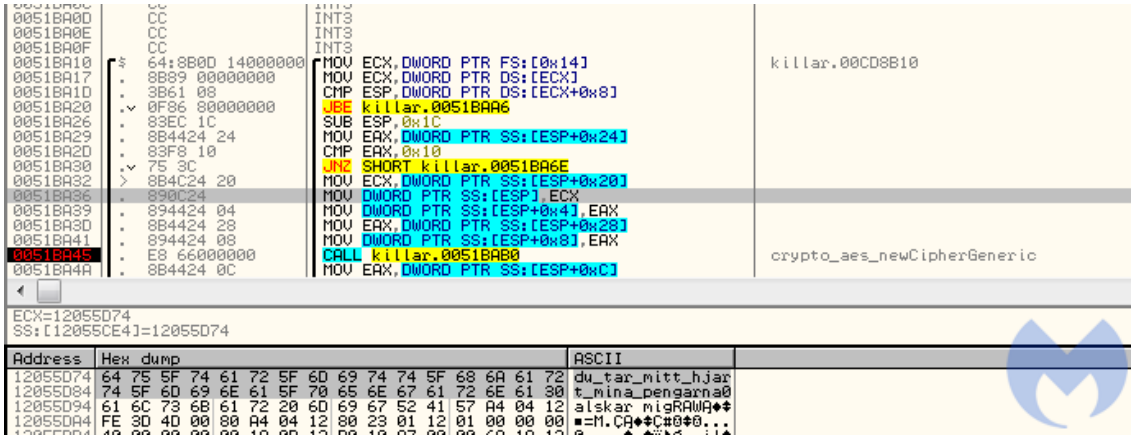
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

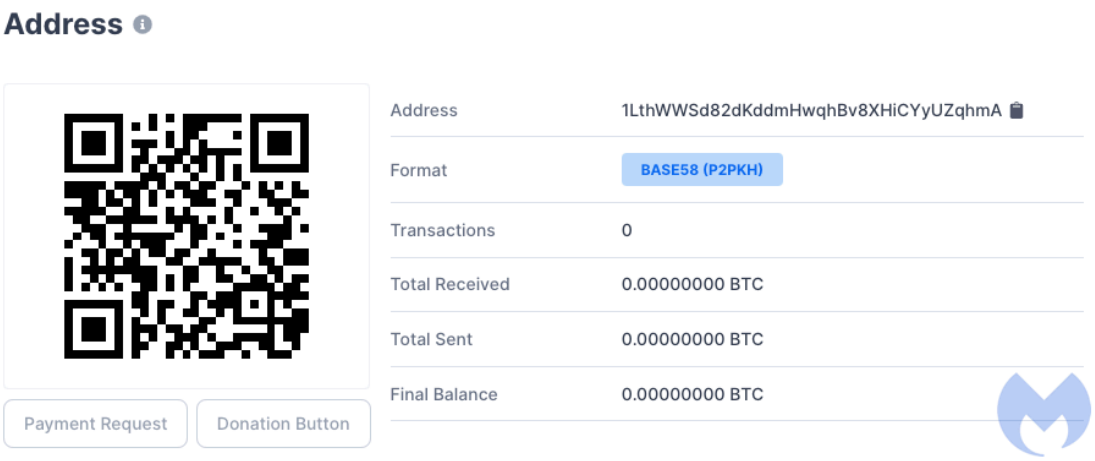
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

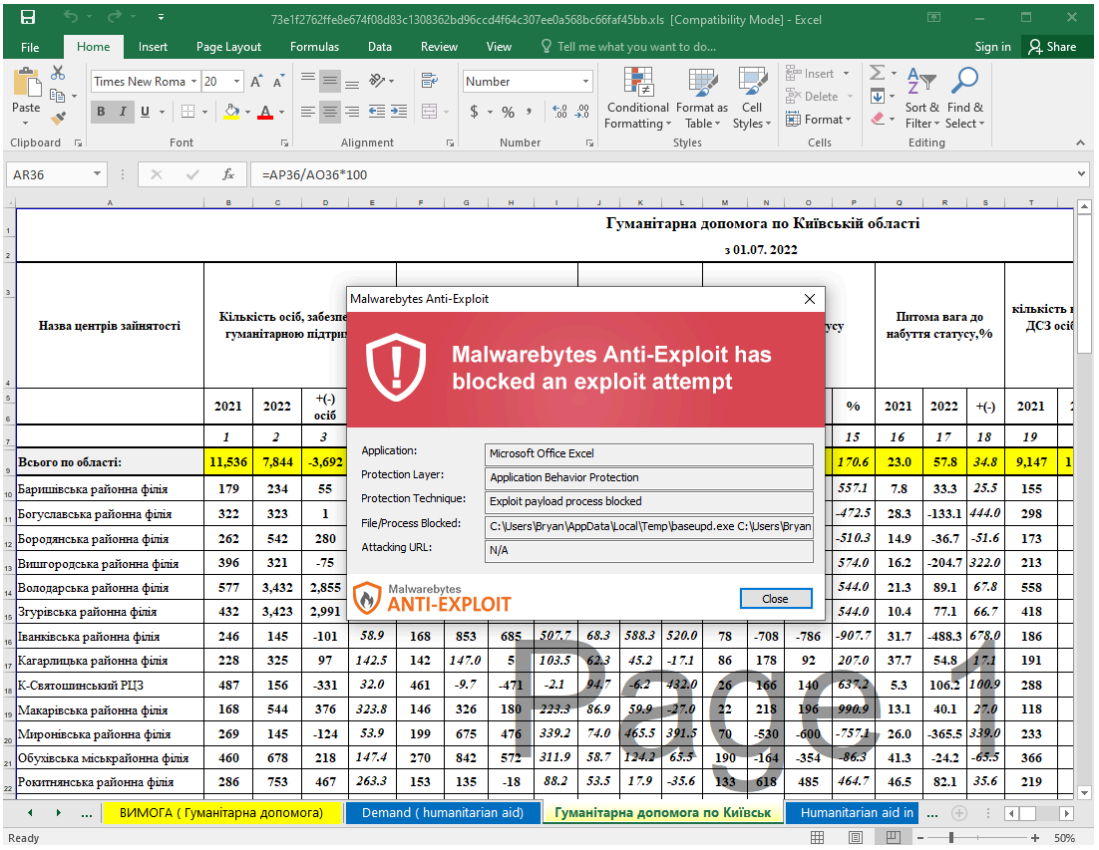
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

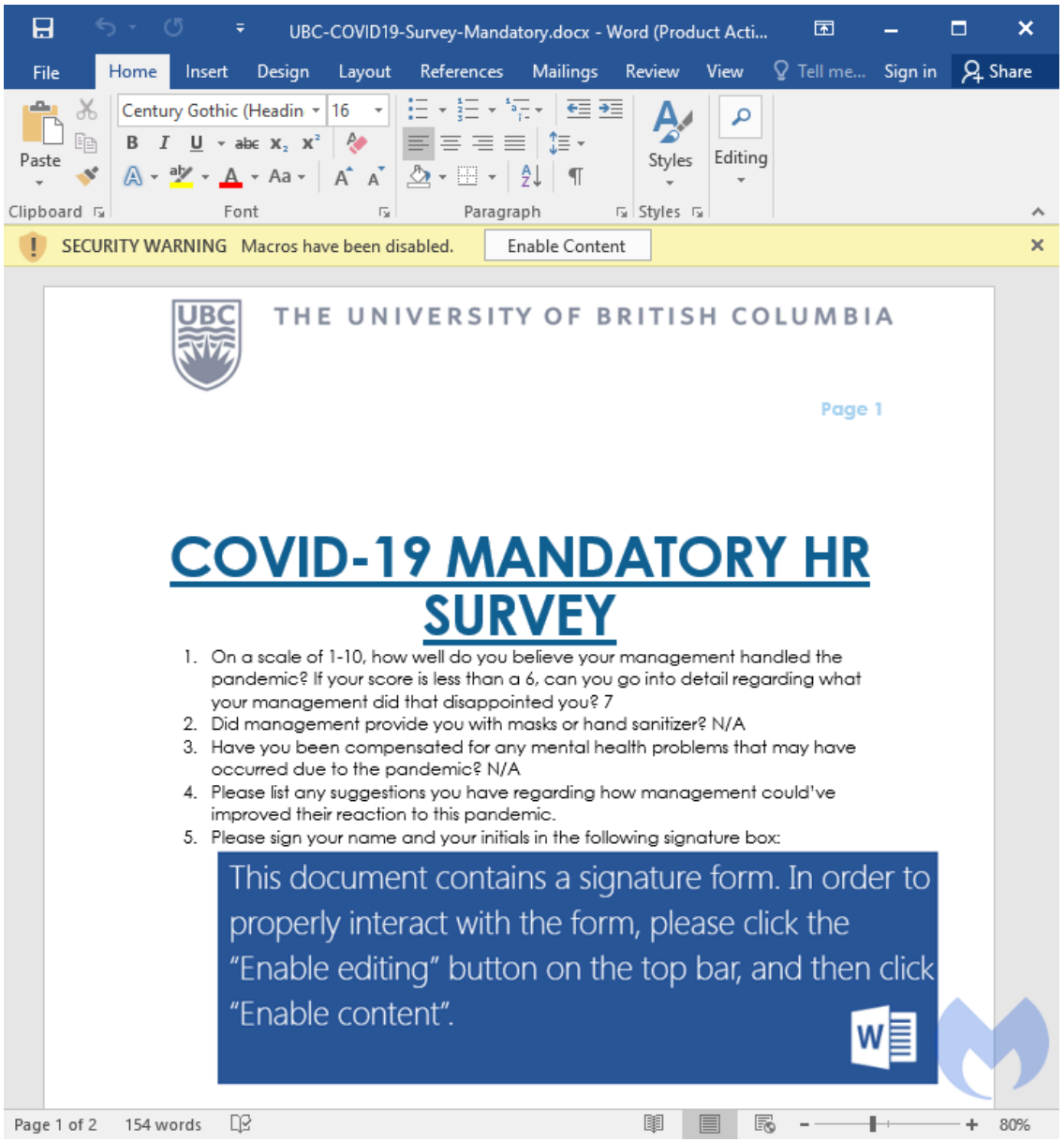
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

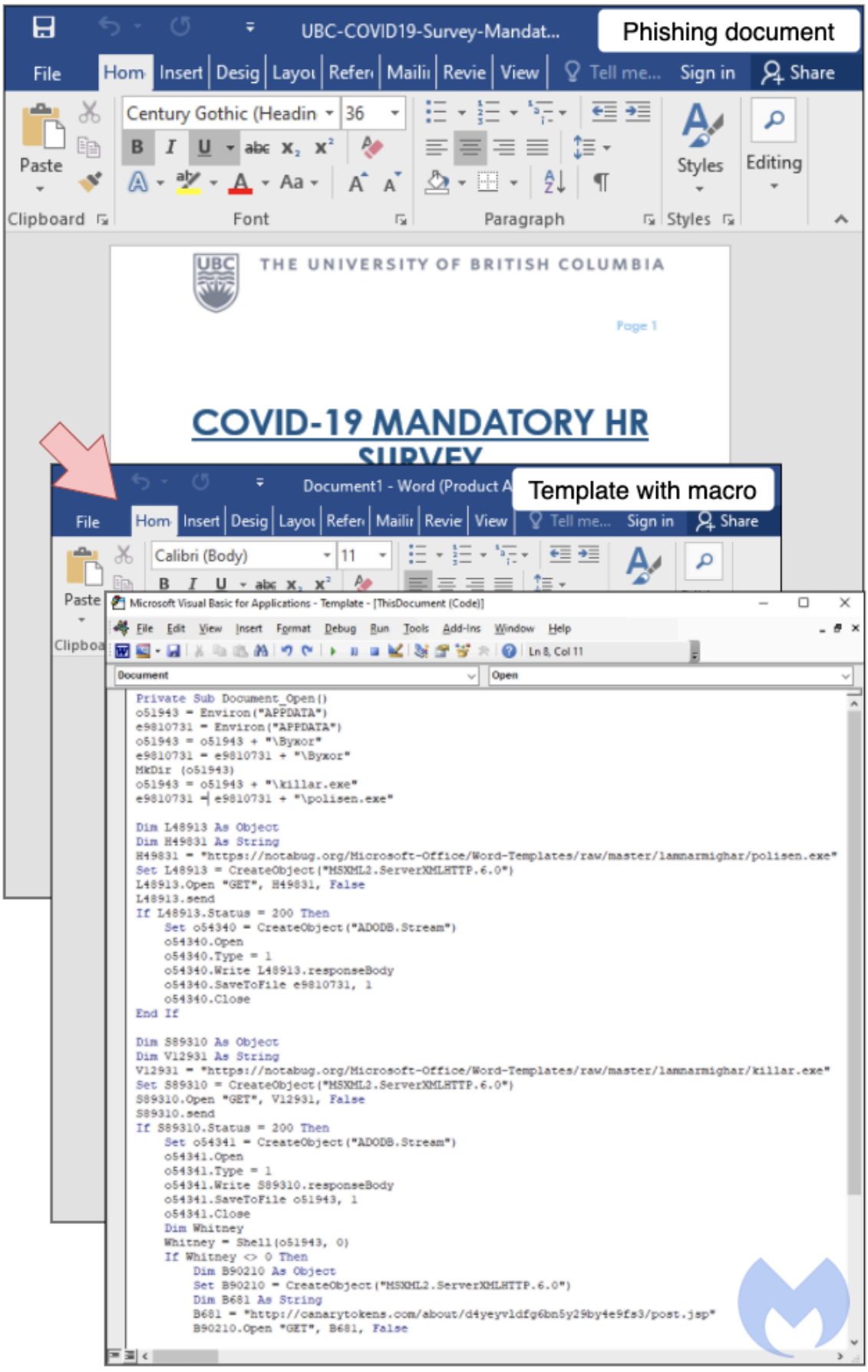
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

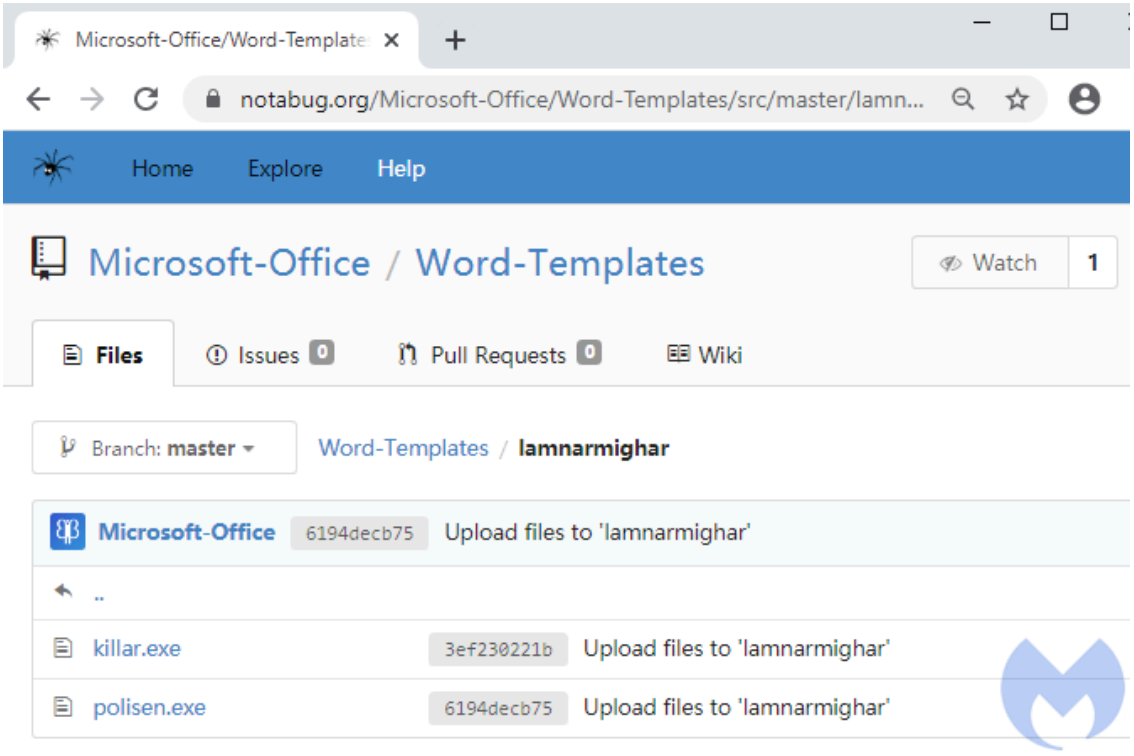
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



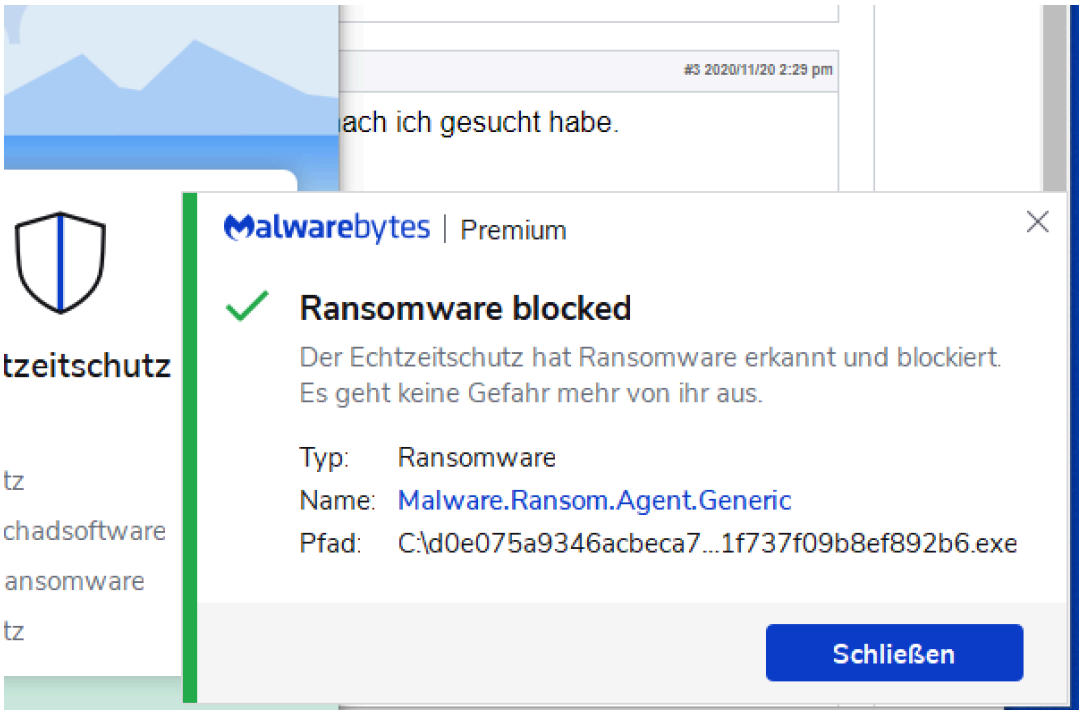
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

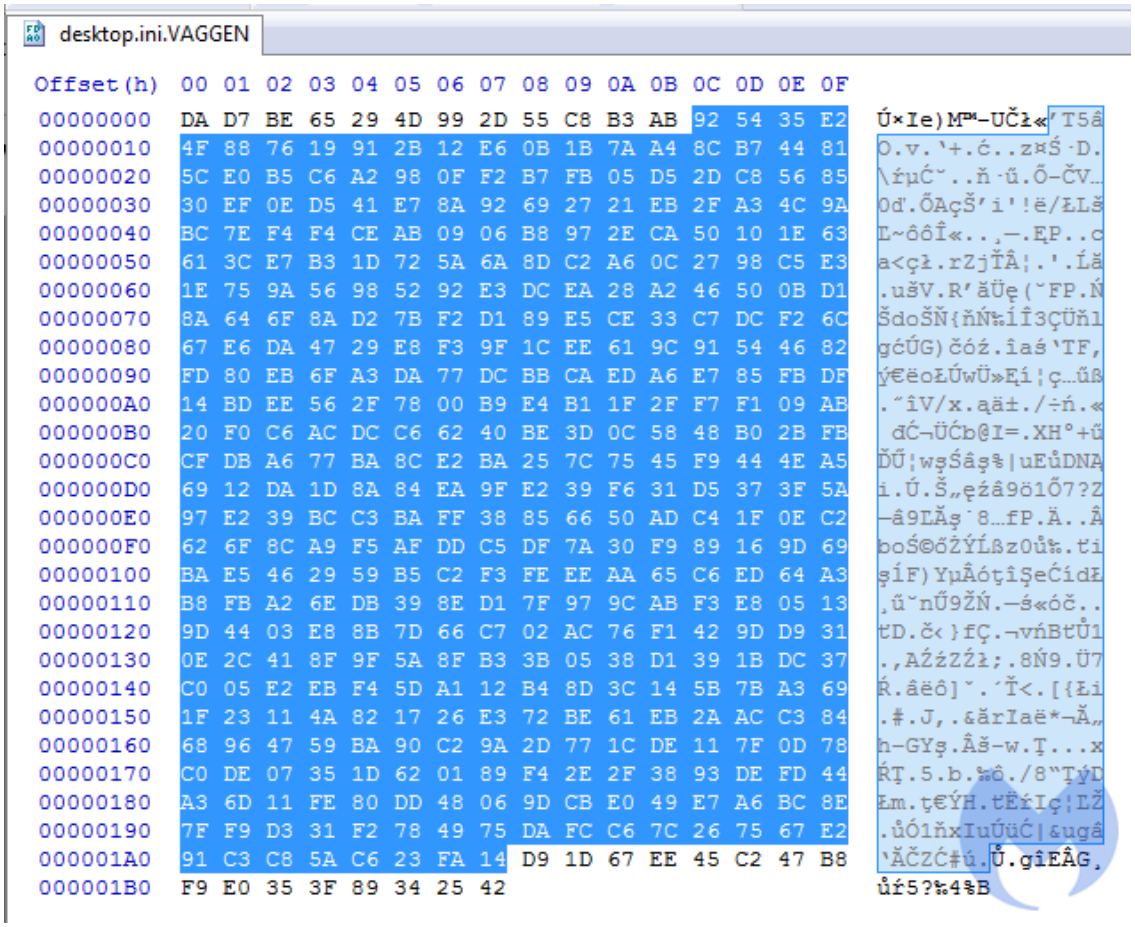
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

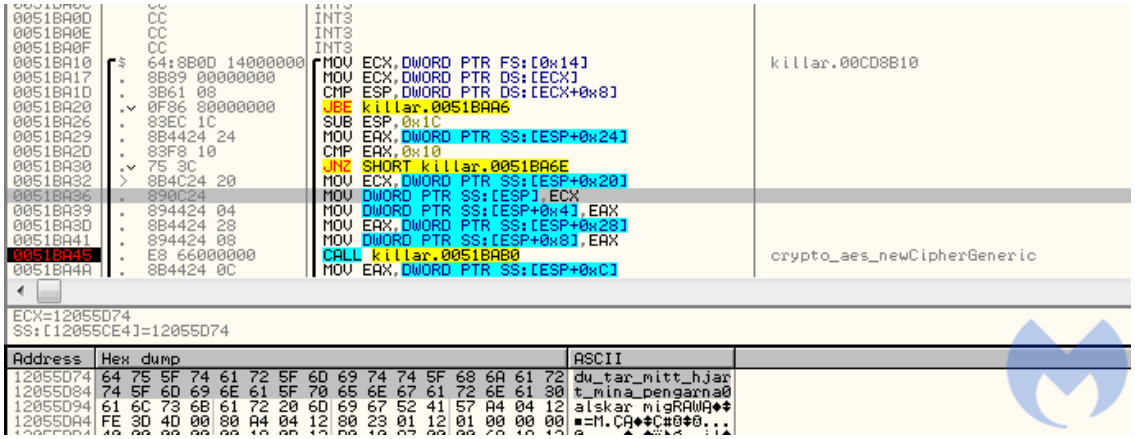


The content of the output file (with .VAGGEN extension) contains:

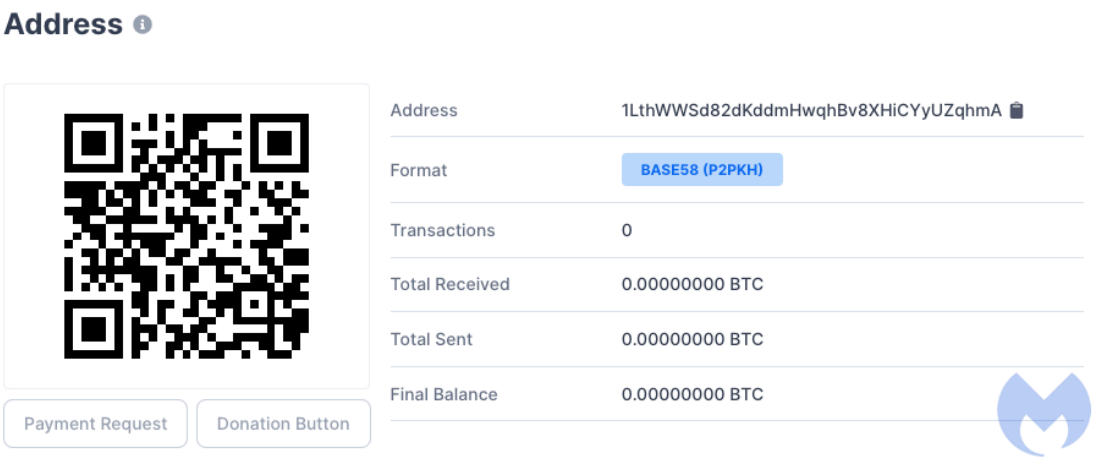
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

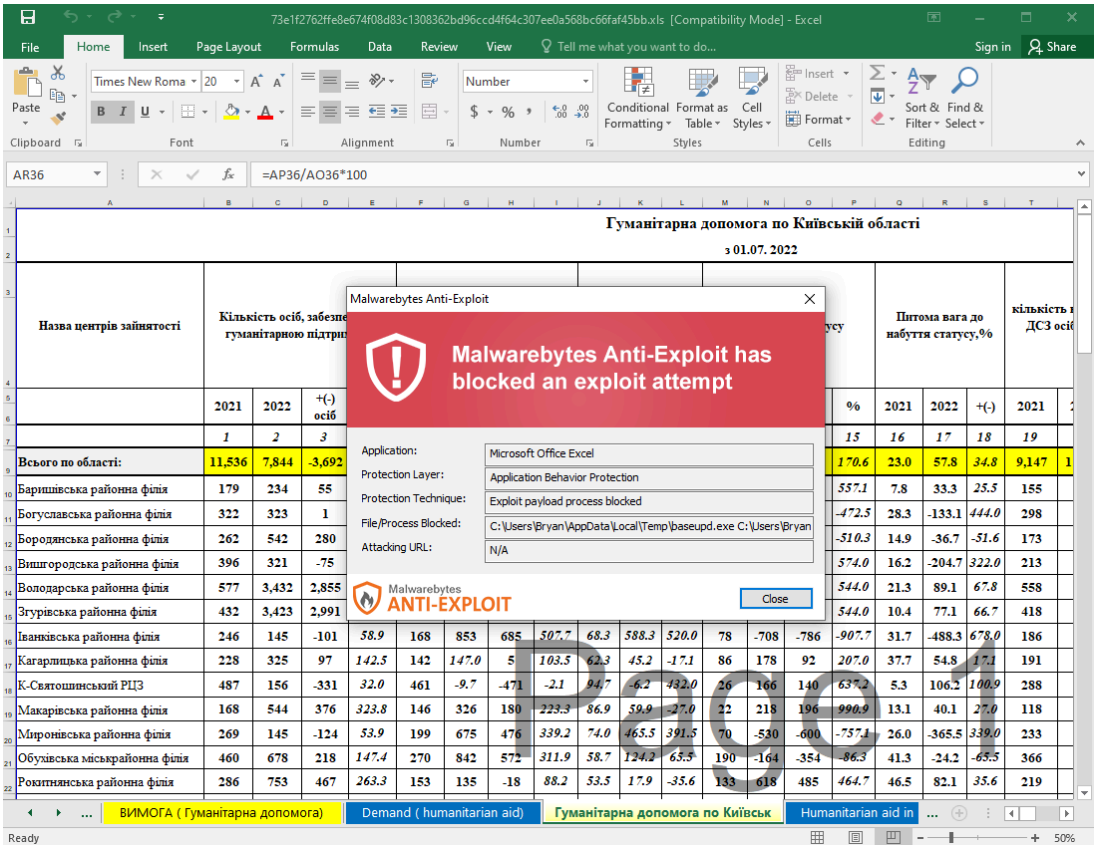
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

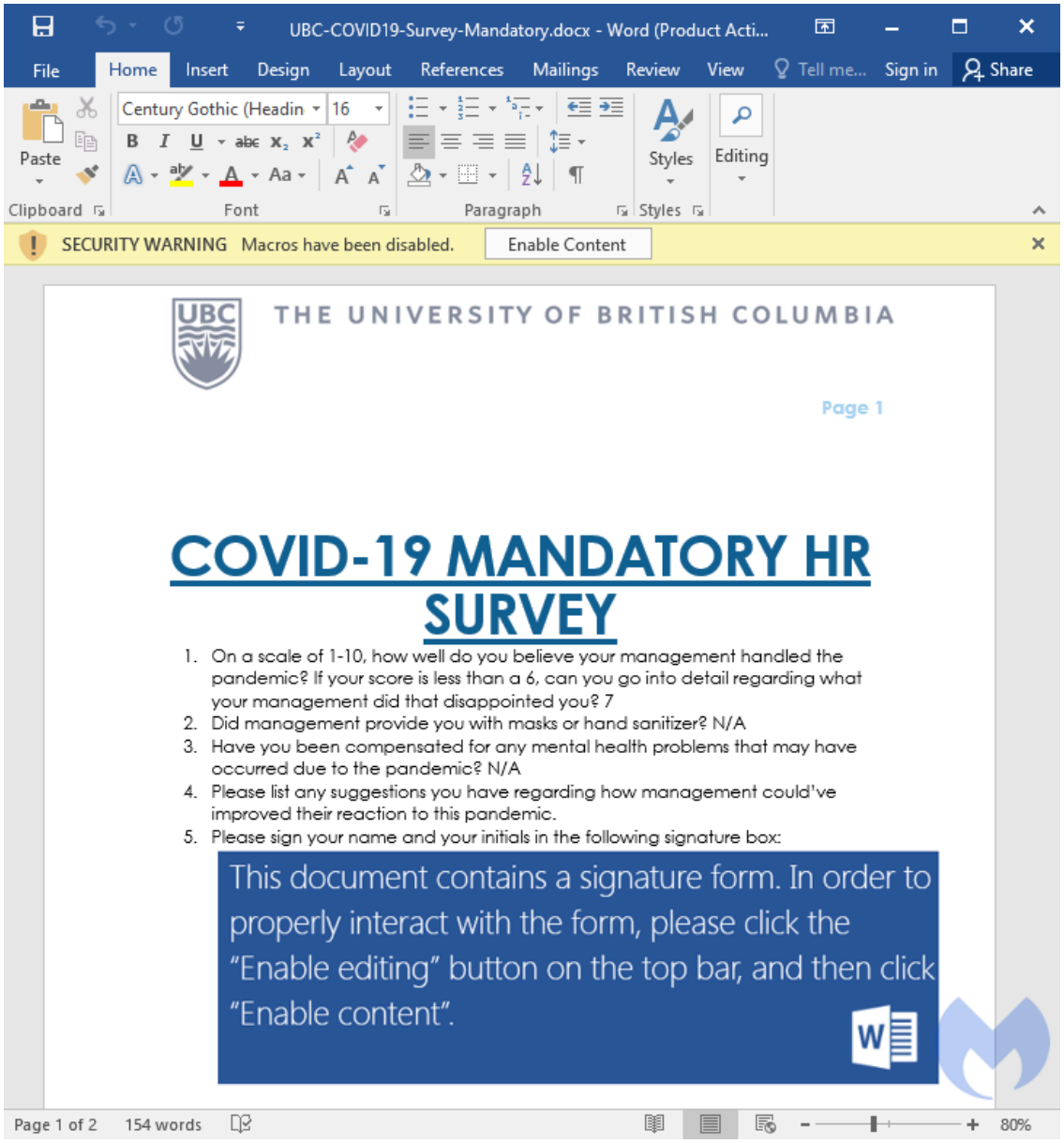
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

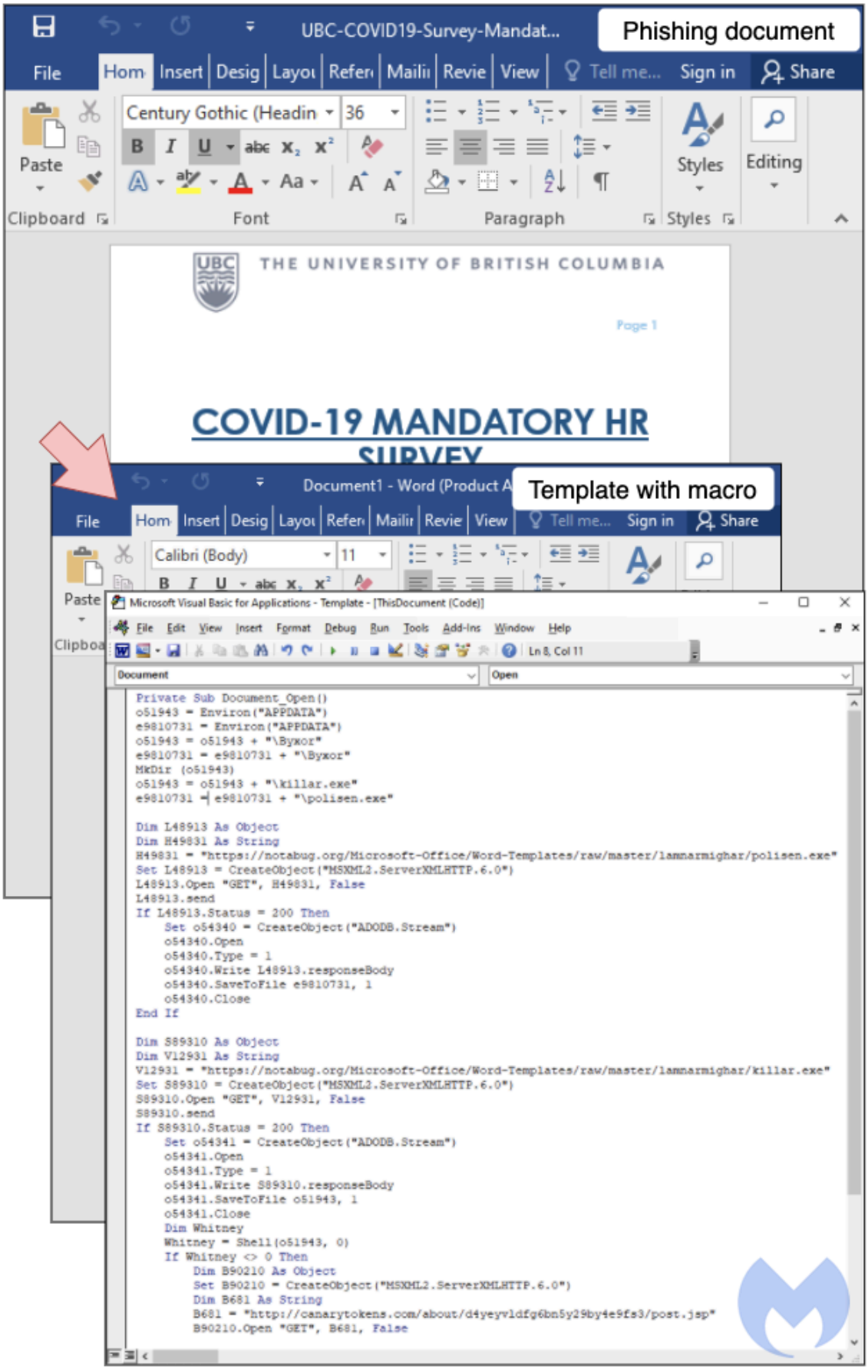
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

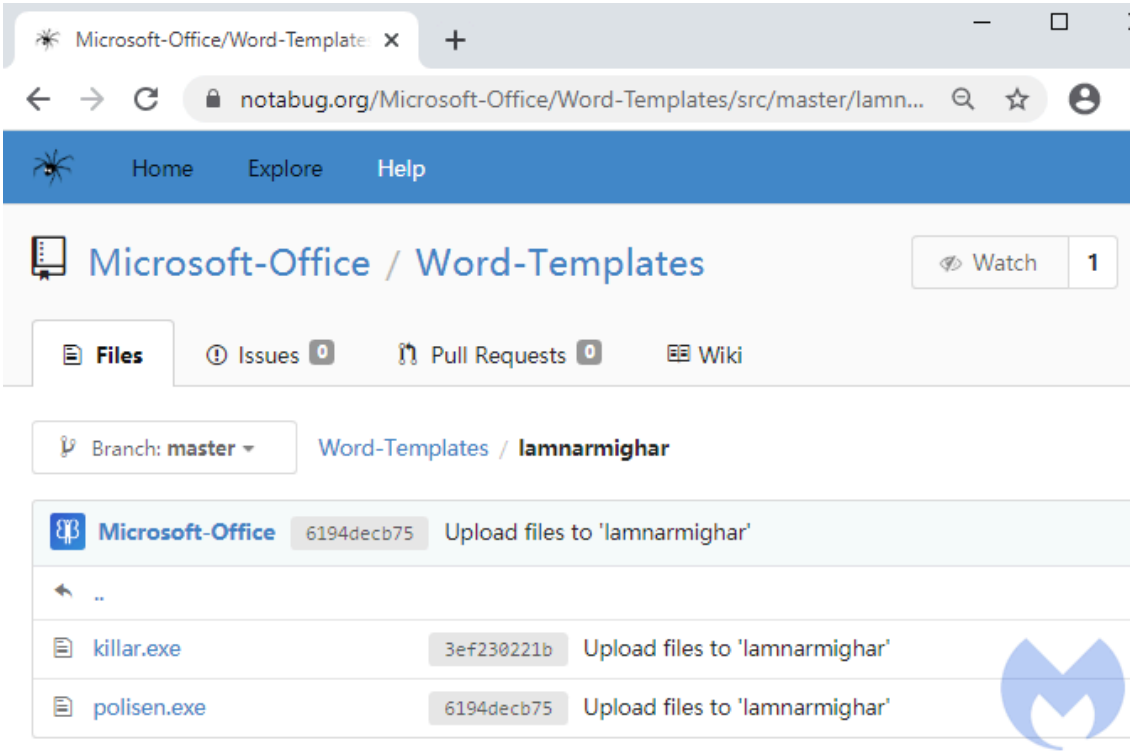
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



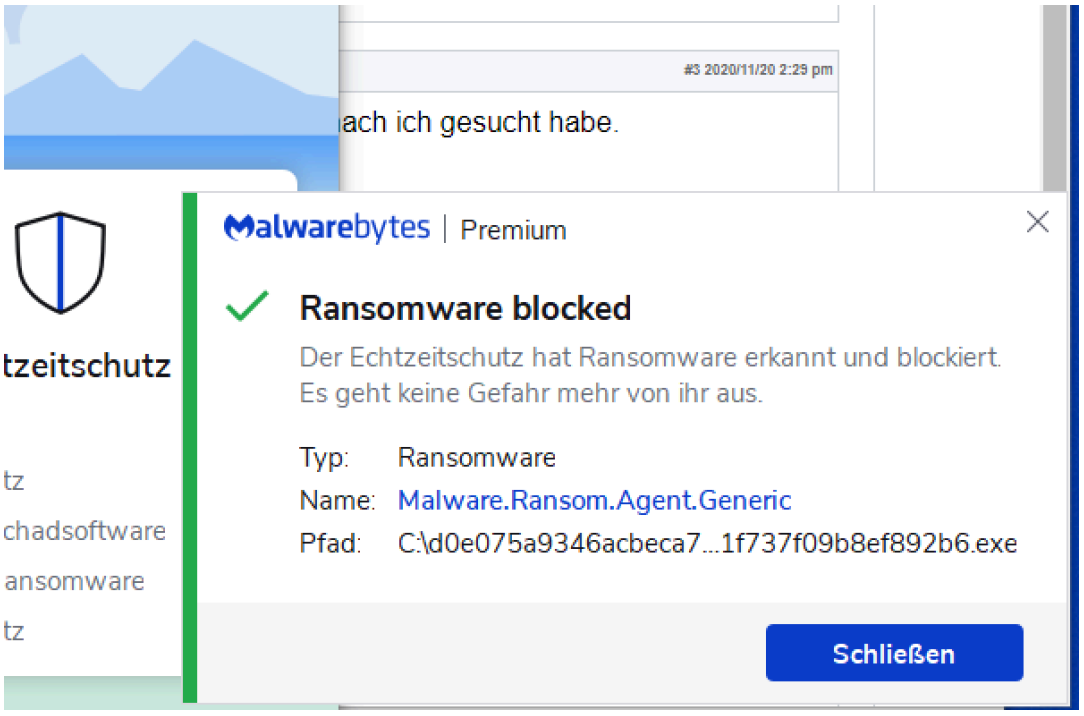
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

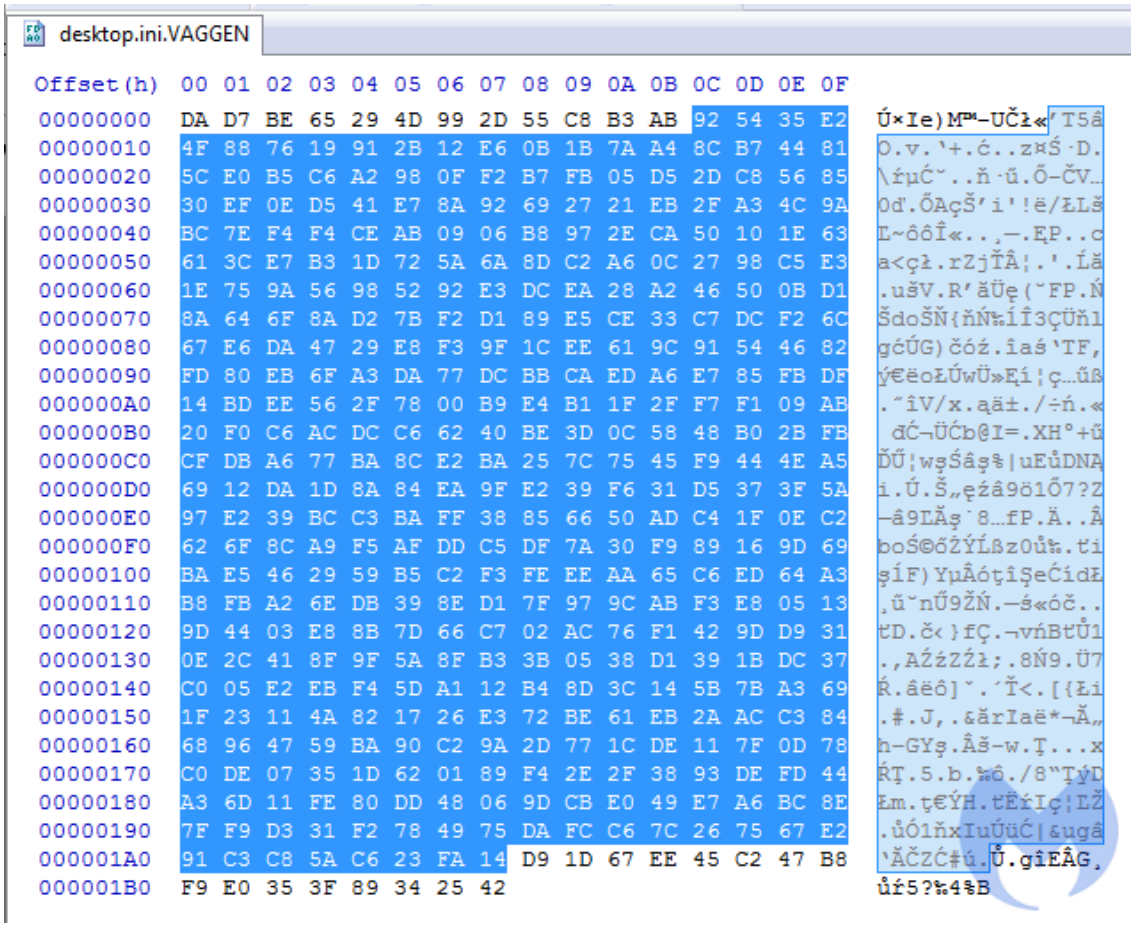
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

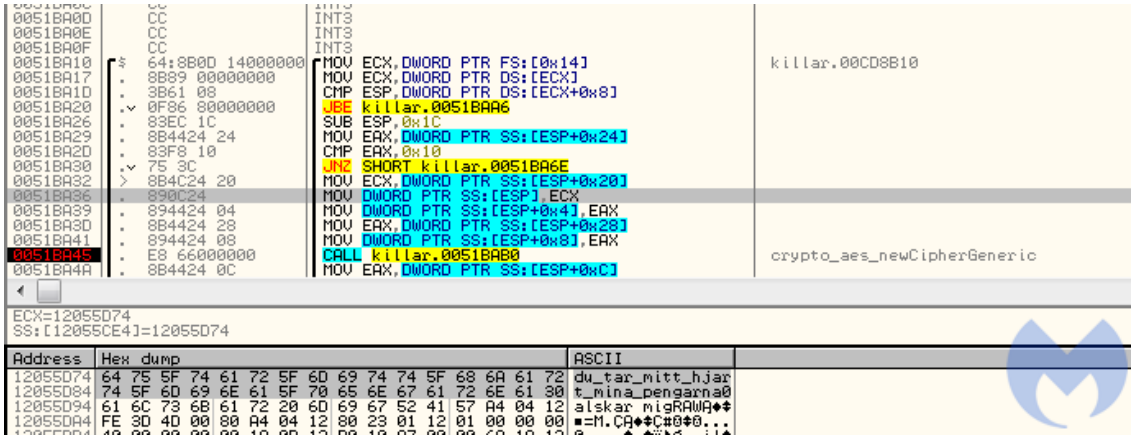


The content of the output file (with .VAGGEN extension) contains:

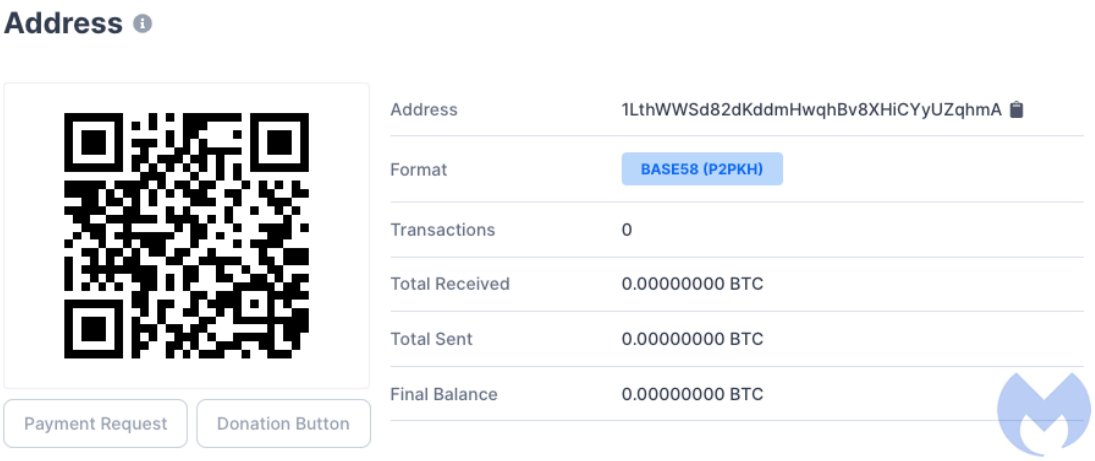
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

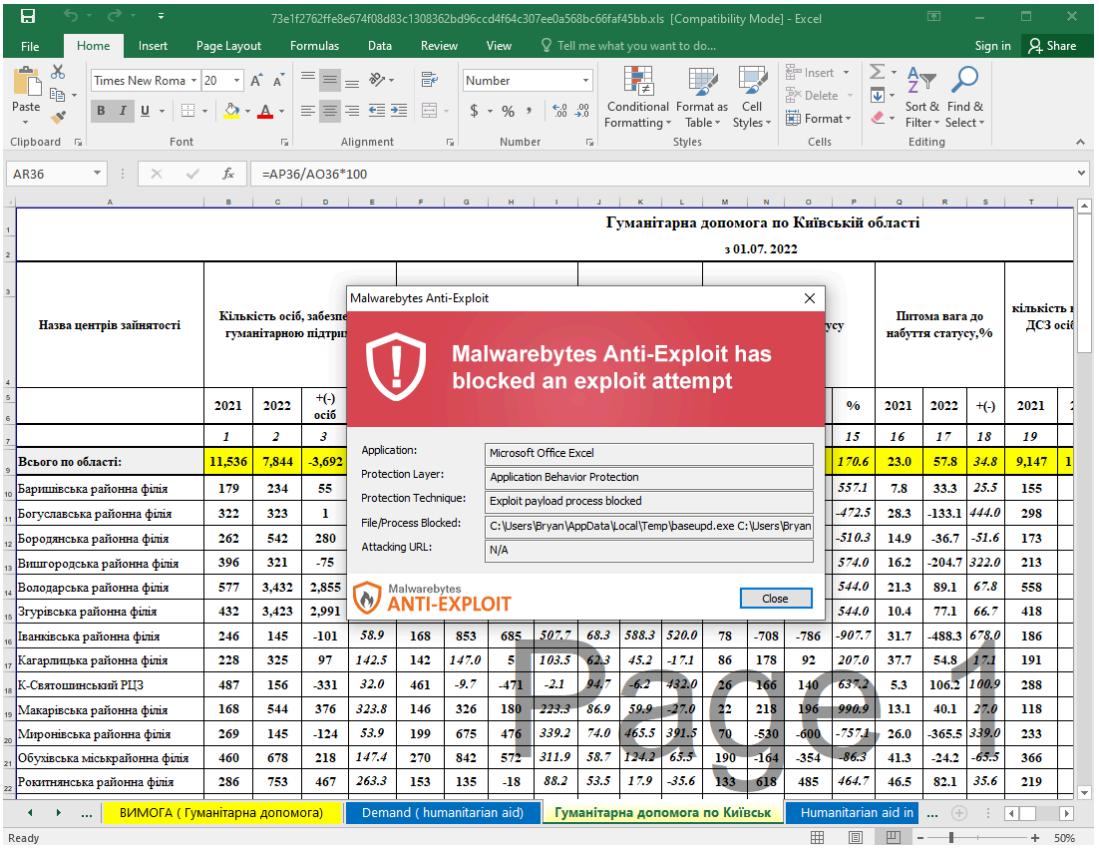
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

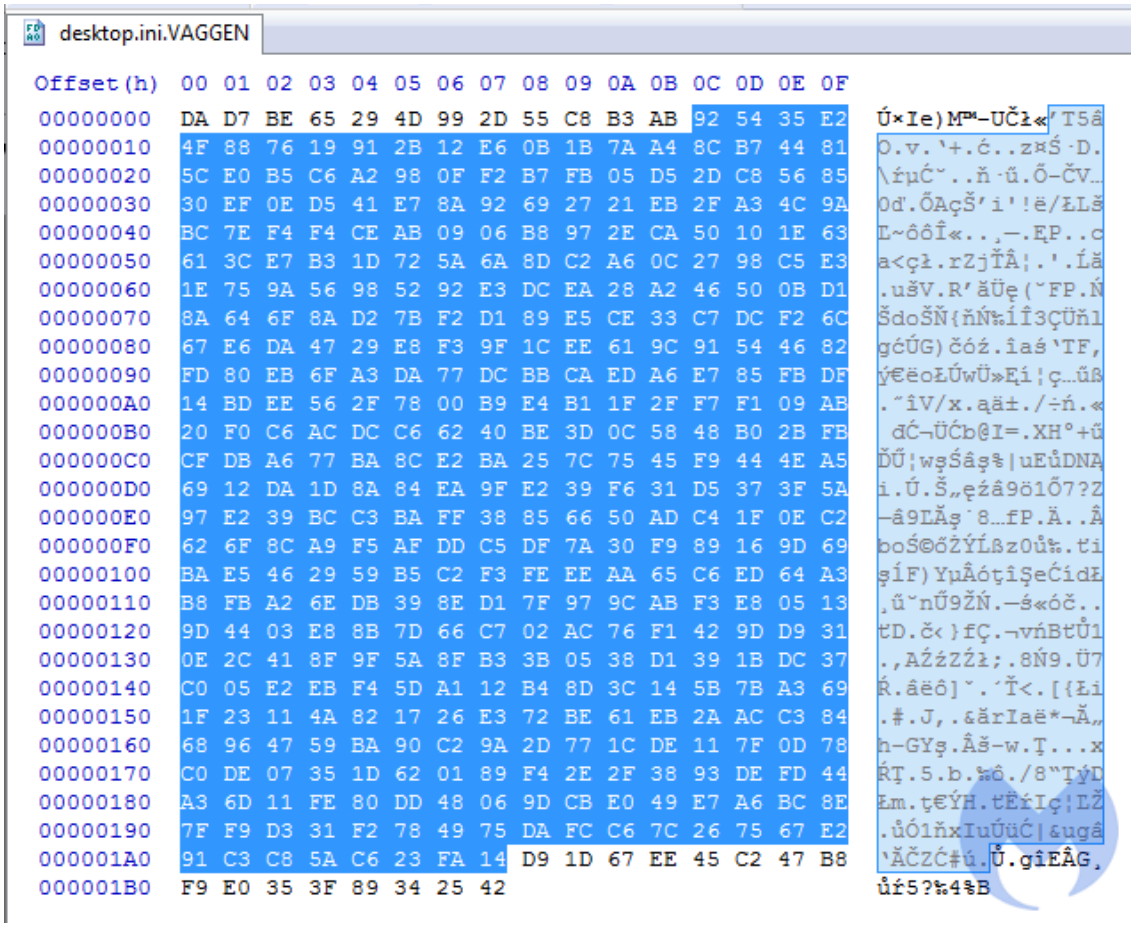
```
46  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
47    runtime_morestack_noctxt();
48  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
49  ((void (*)(void))loc_45CDA0)();
50  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
51  v38 = v4;
52  v30 = v5;
53  v31 = v6;
54  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
55  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
56  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
57  v32 = v10;
58  v11 = *(void (__cdecl **)(int))(v10 + 16);
59  v39 = v12;
60  v11(v12);
61  v29 = v13;
62  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
63  v40 = v15;
64  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
65  v17 = v27;
66  if ( v27 )
67  {
68    if ( v27 )
69      v17 = *(_DWORD *)(v27 + 4);
70    v43 = v17;
71    v44 = v28;
72    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
73  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

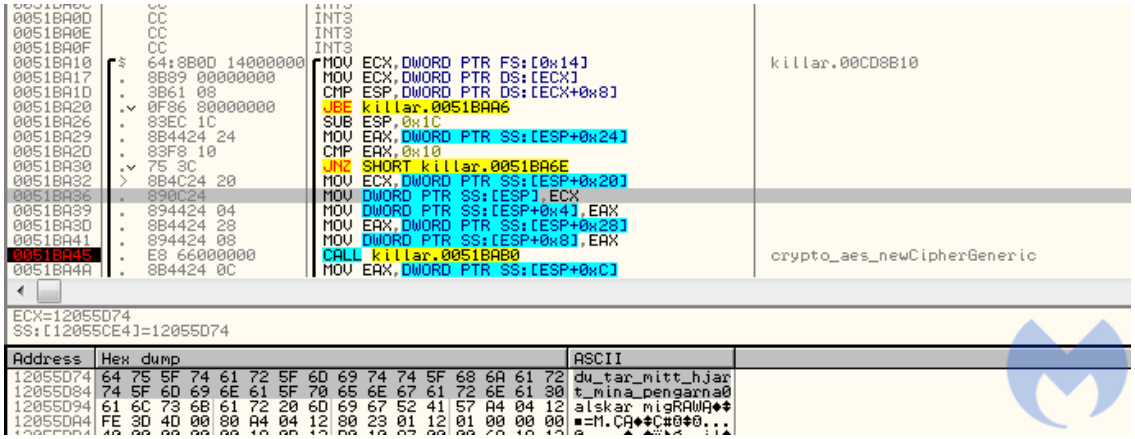


The content of the output file (with .VAGGEN extension) contains:

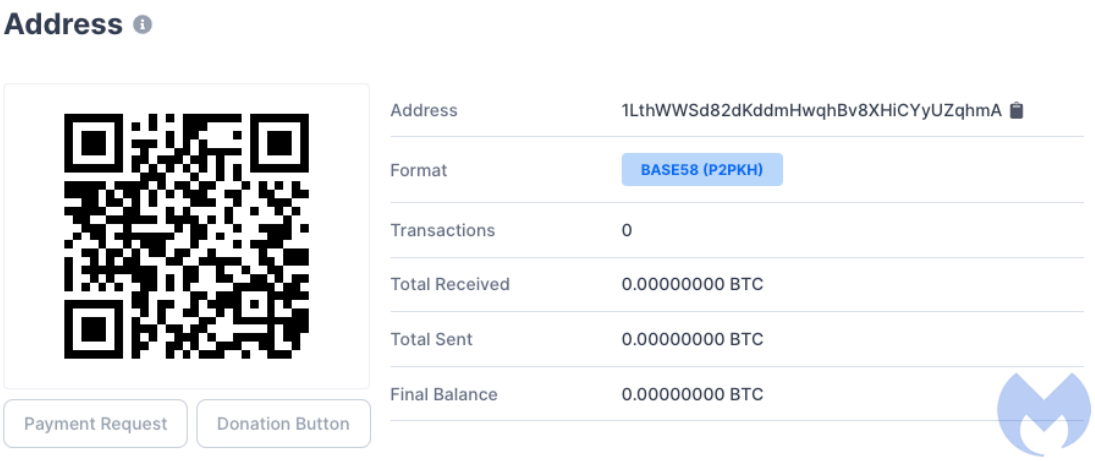
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

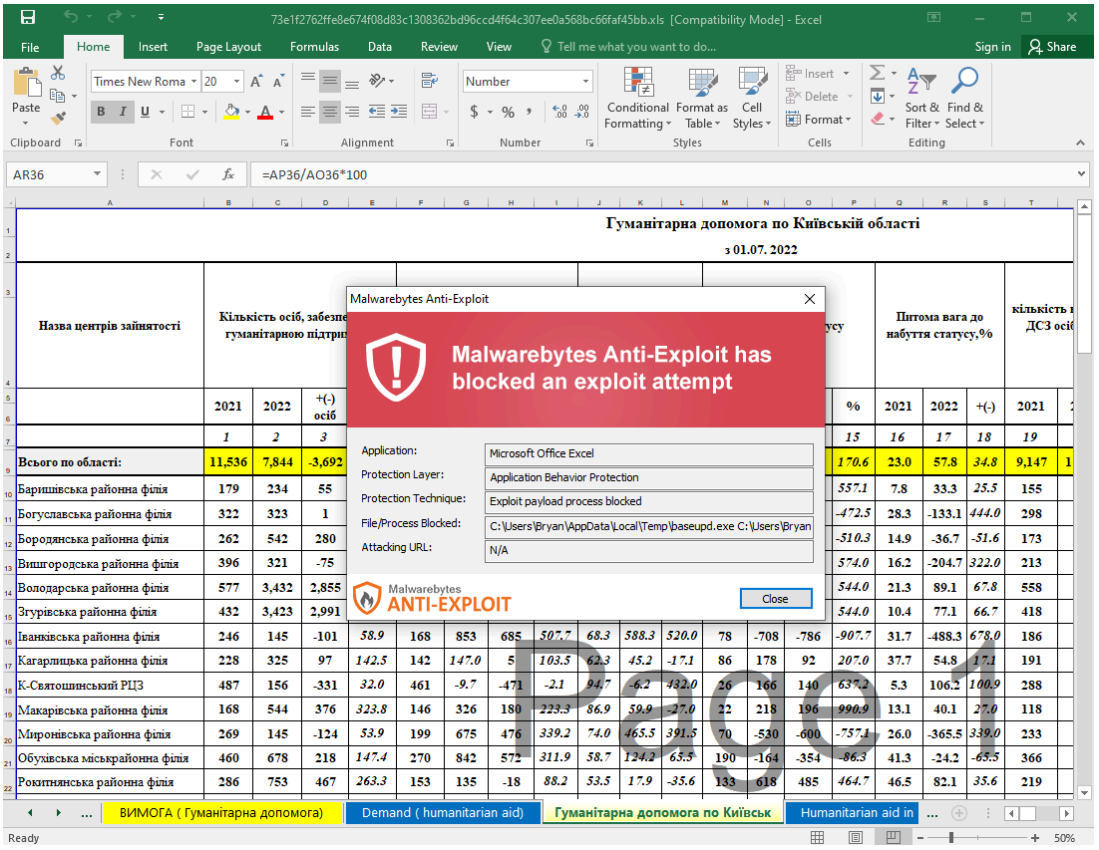
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

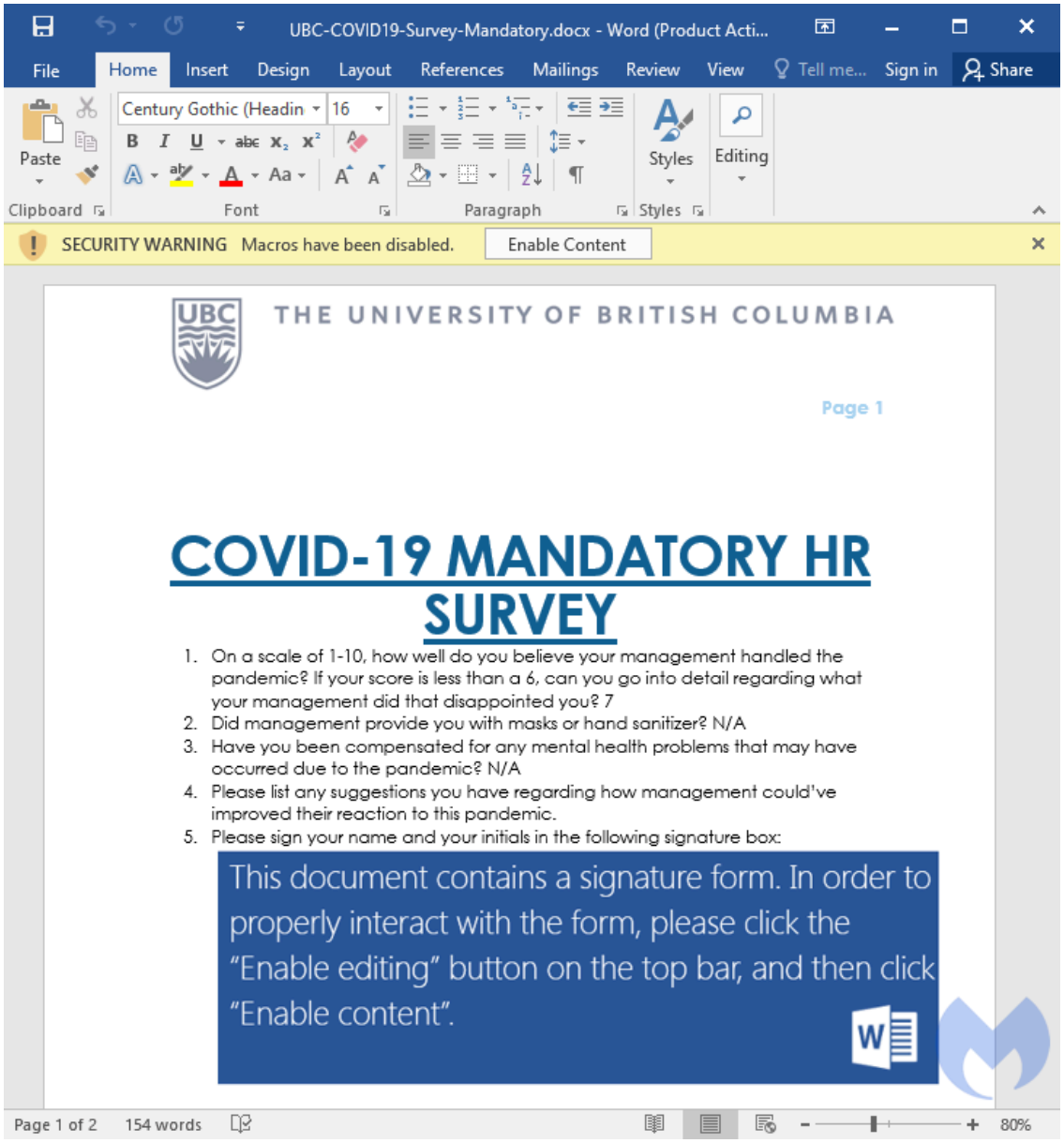
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

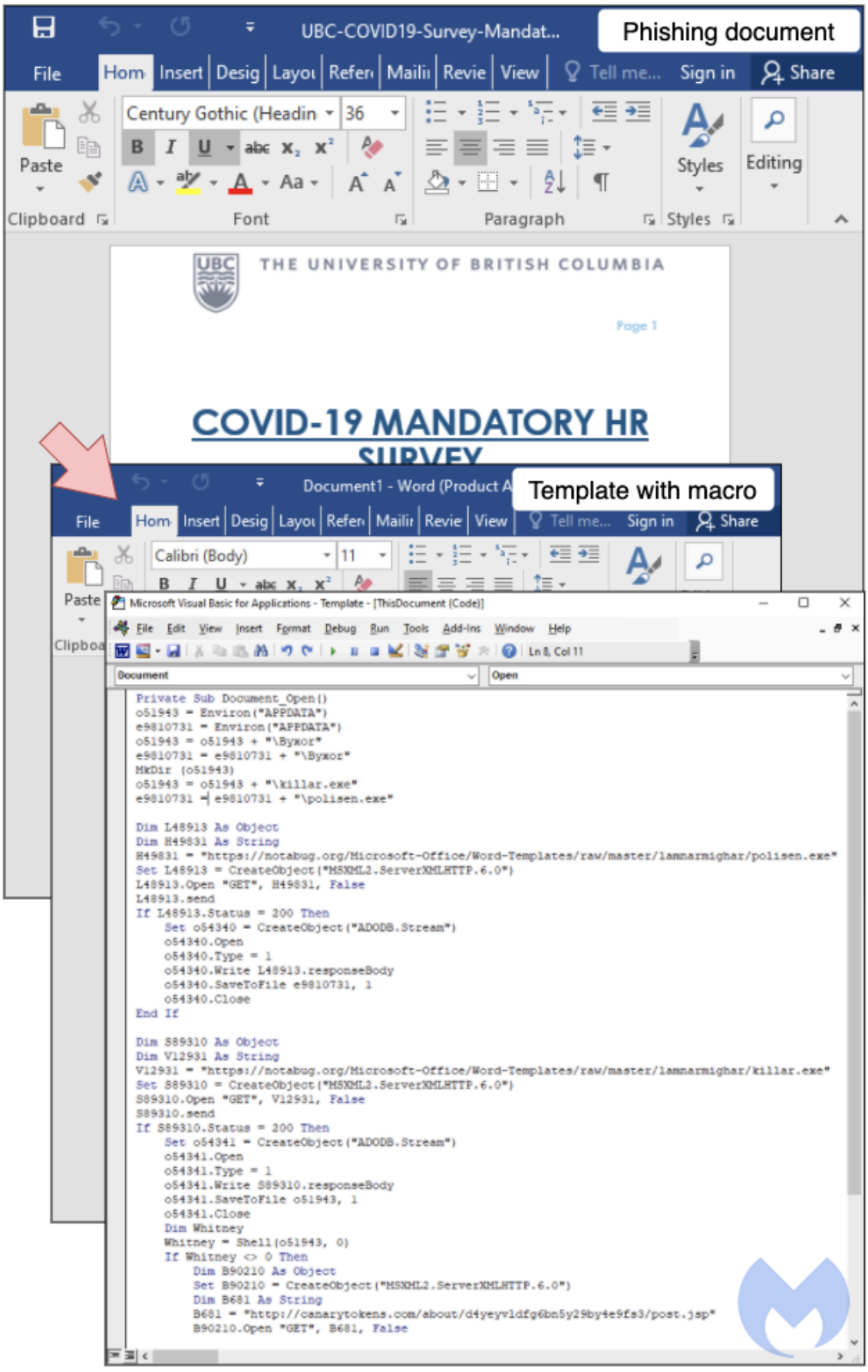
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

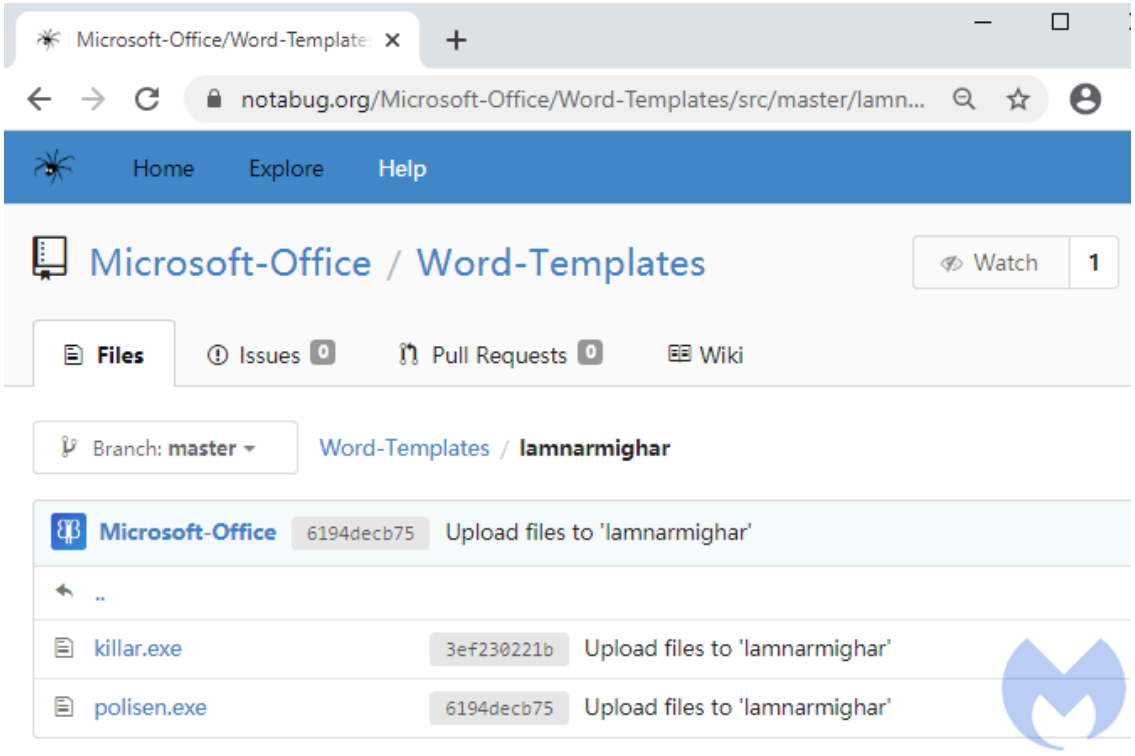
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



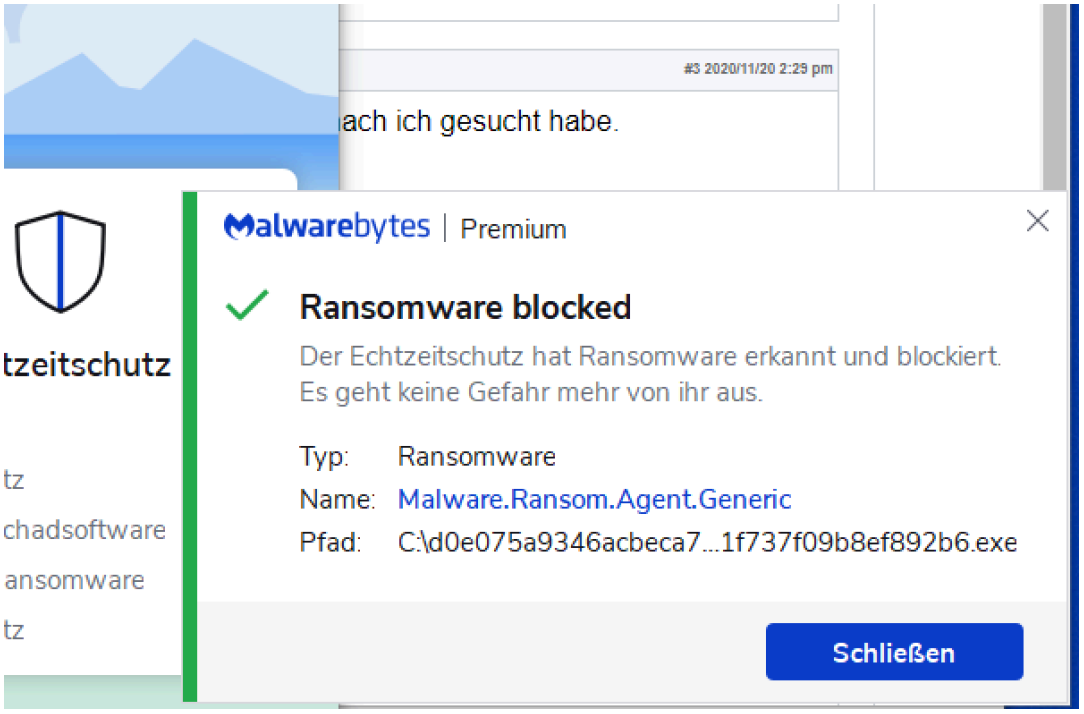
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

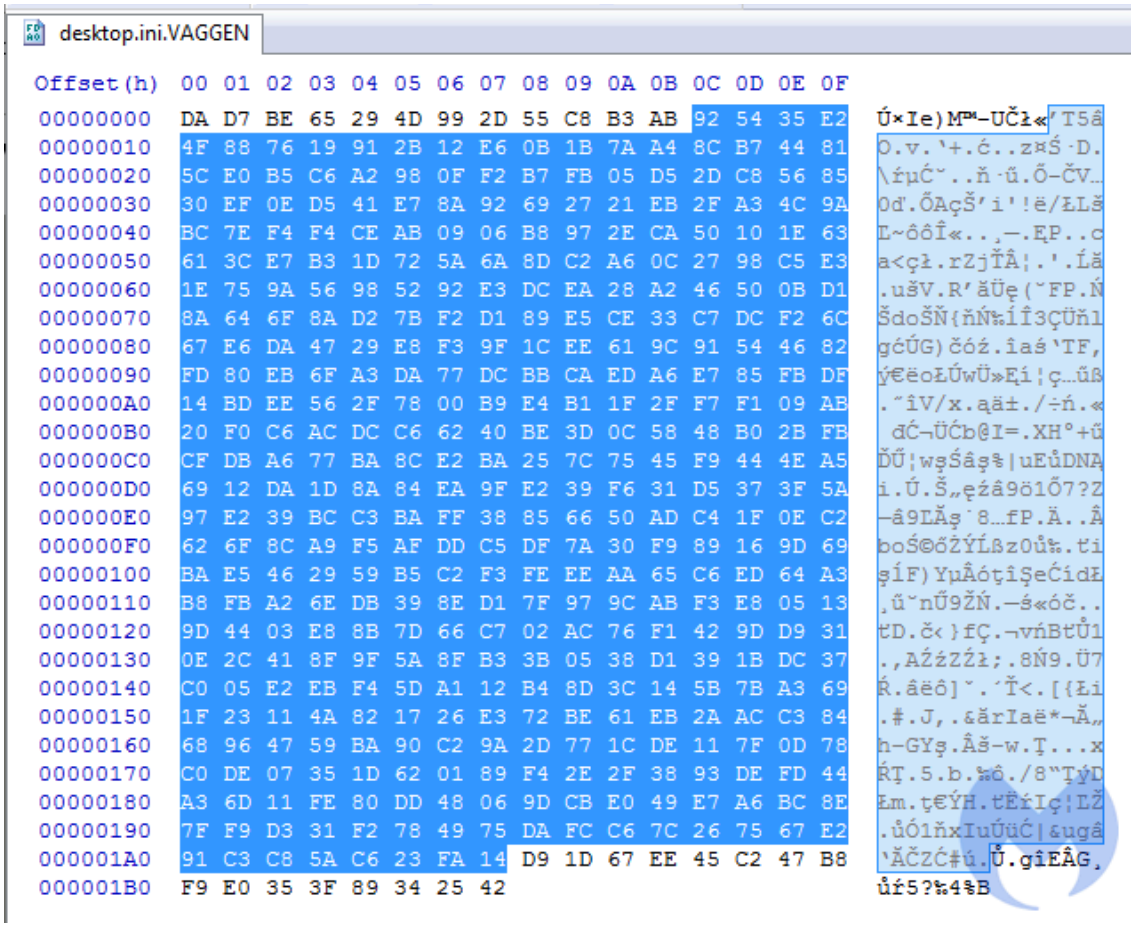
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

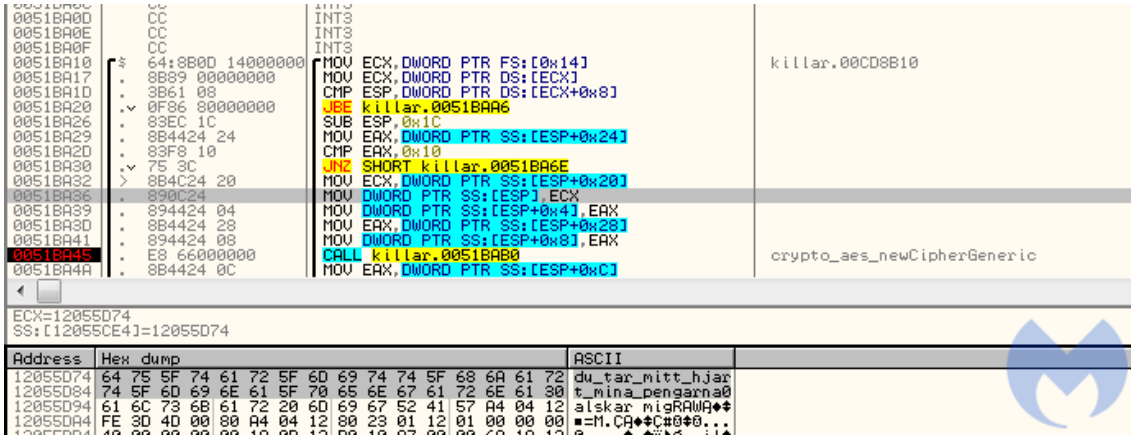


The content of the output file (with .VAGGEN extension) contains:

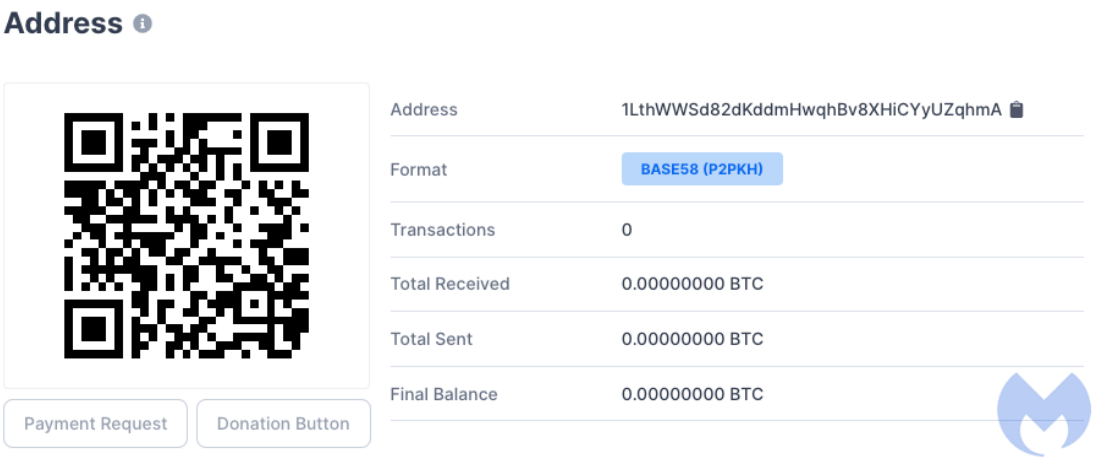
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

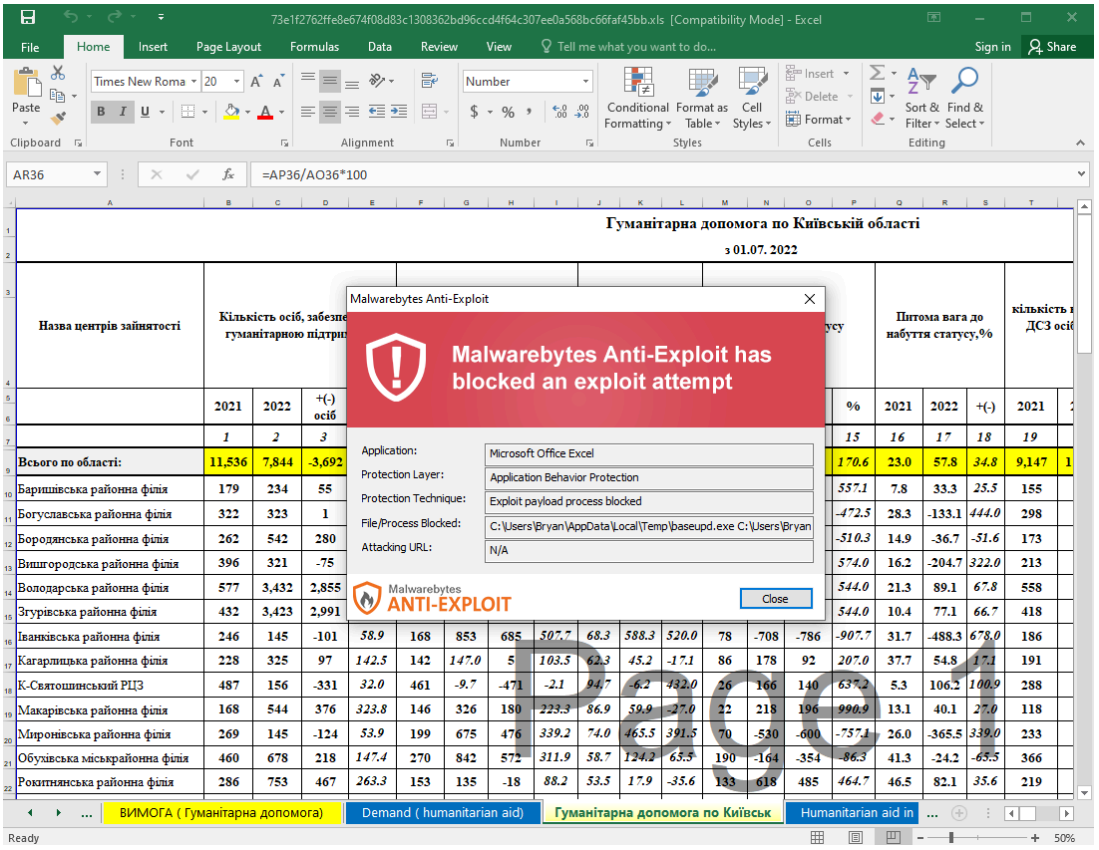
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

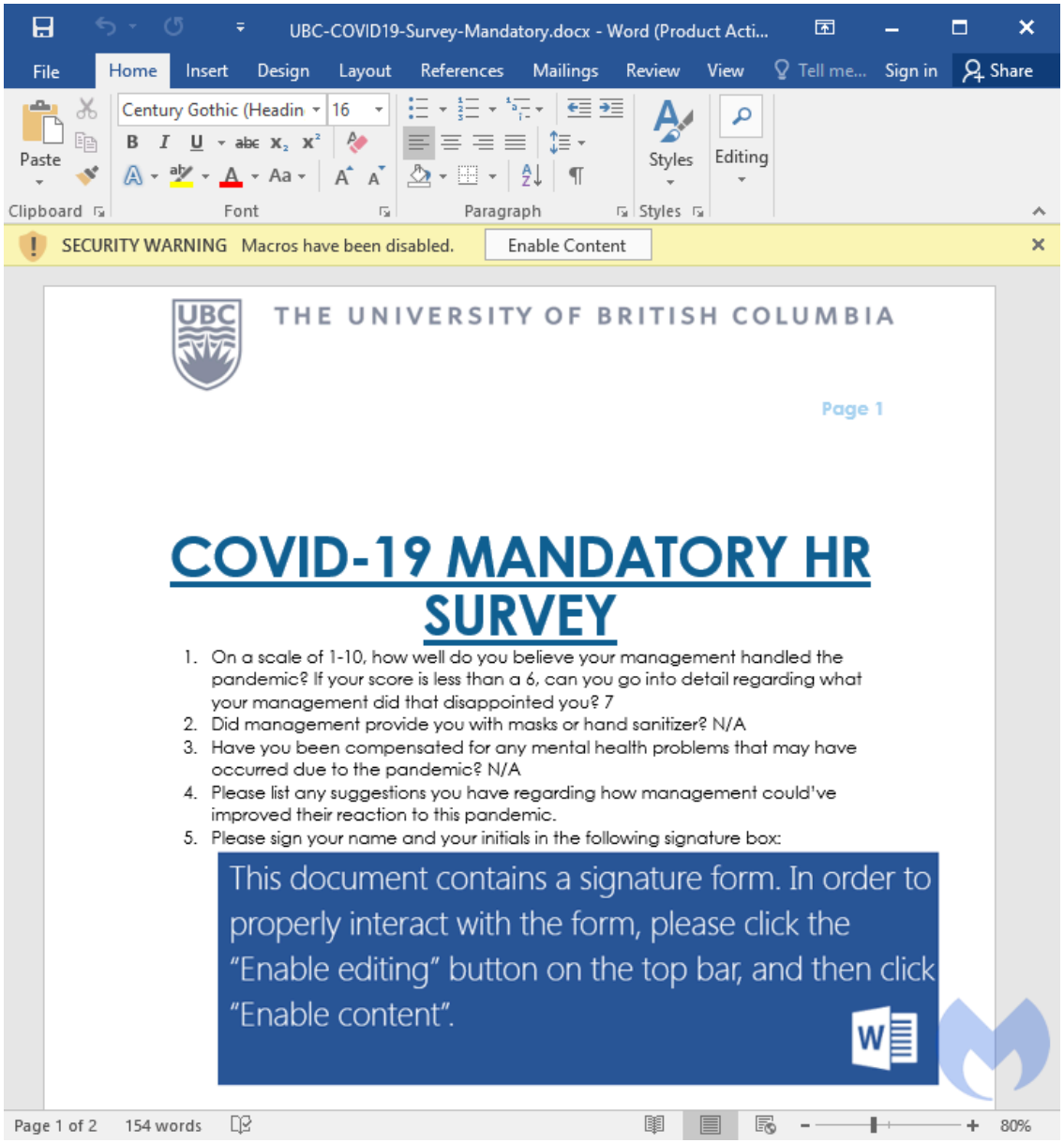
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

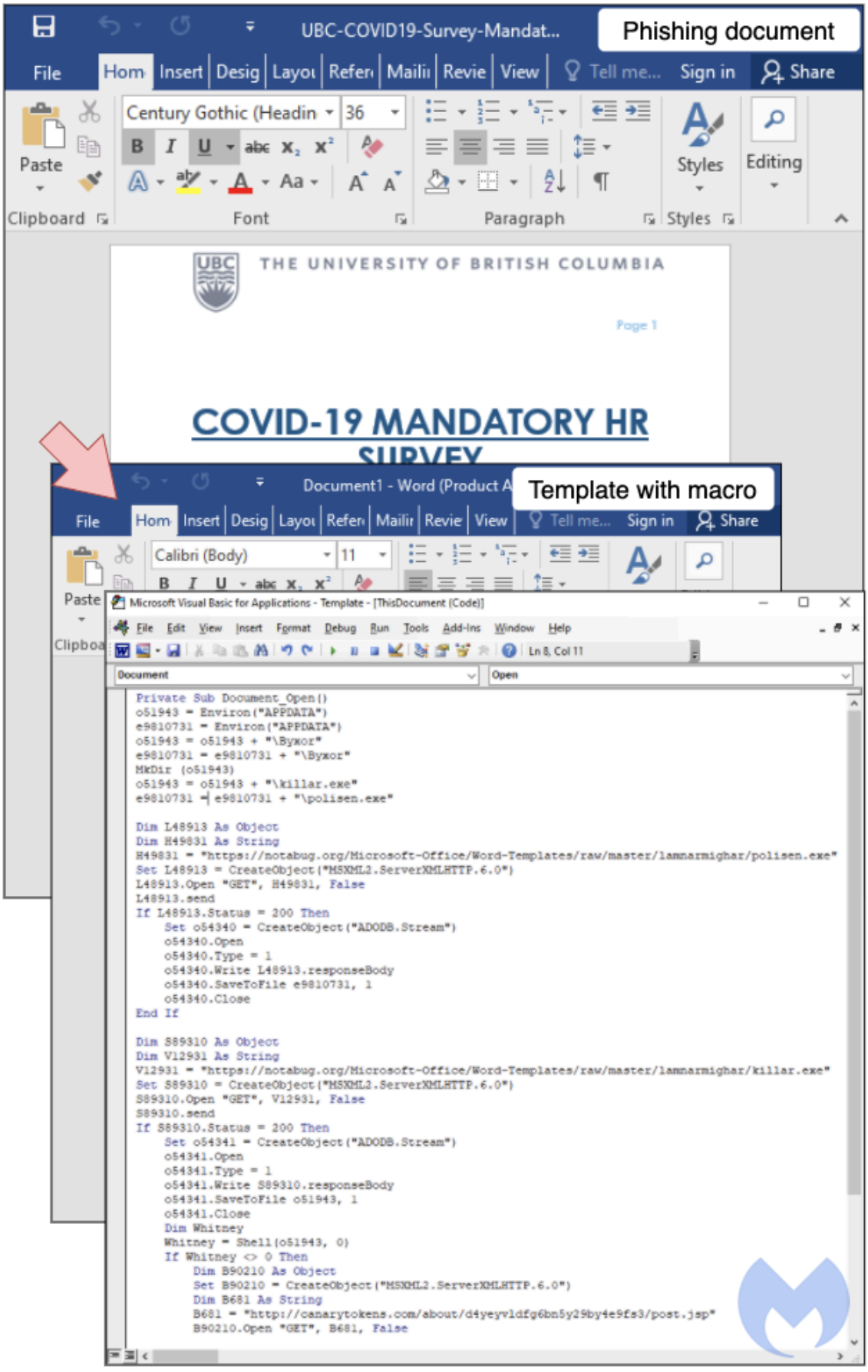
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

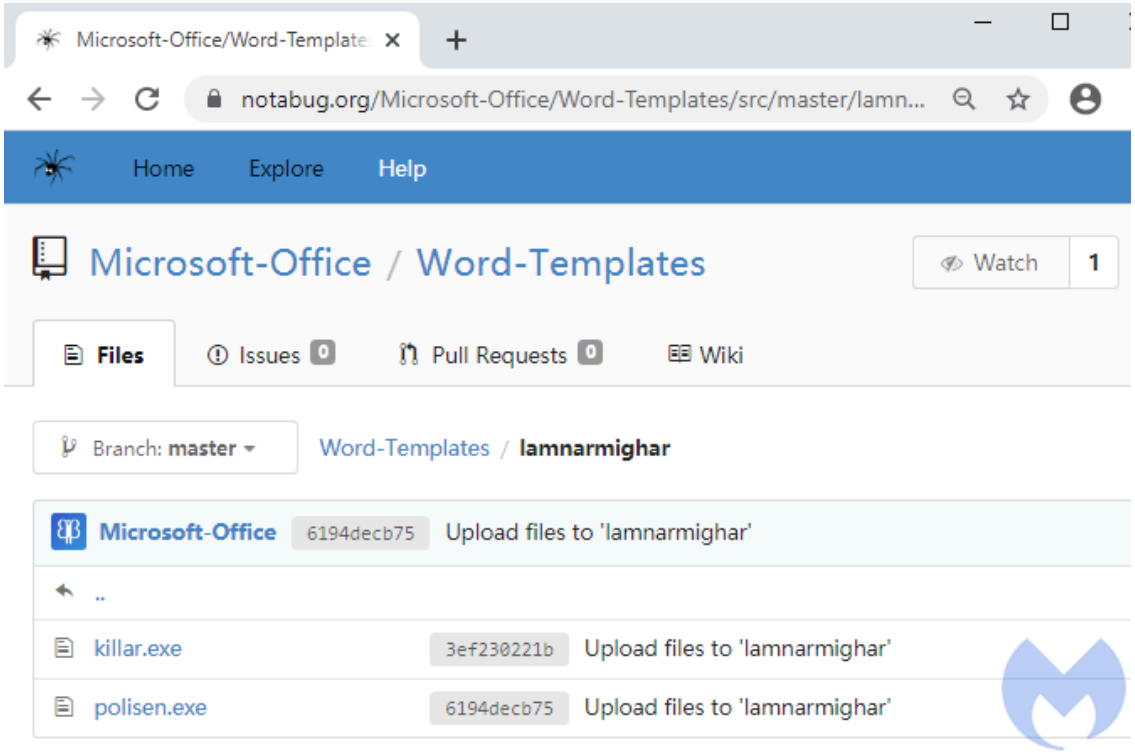
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



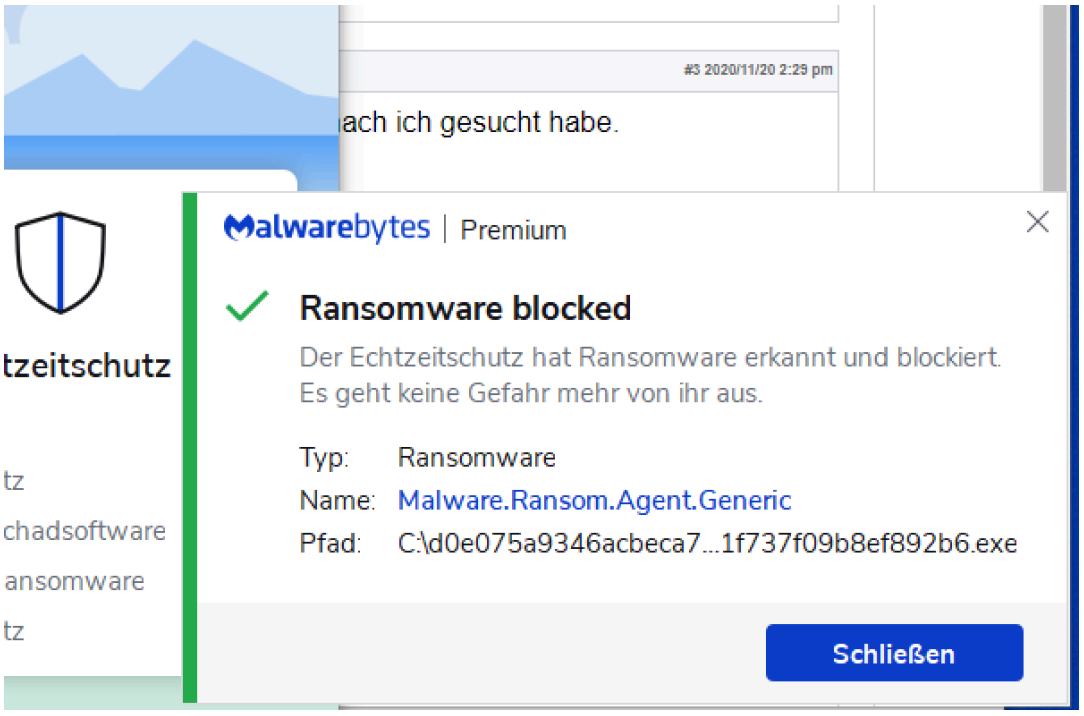
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

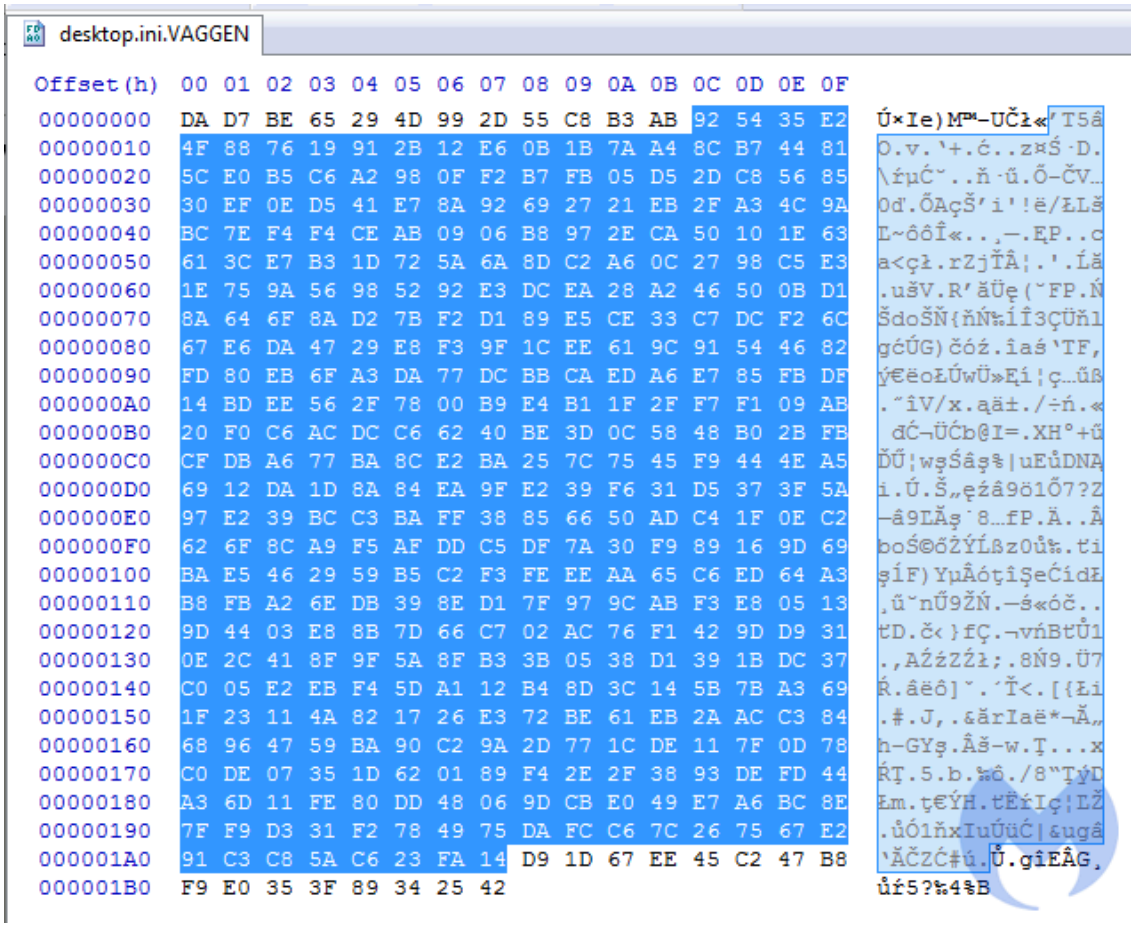
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

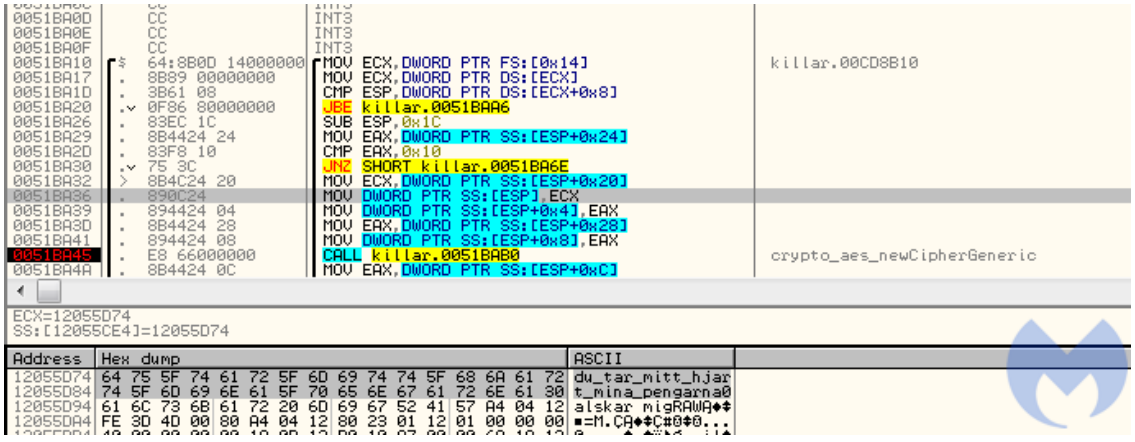


The content of the output file (with .VAGGEN extension) contains:

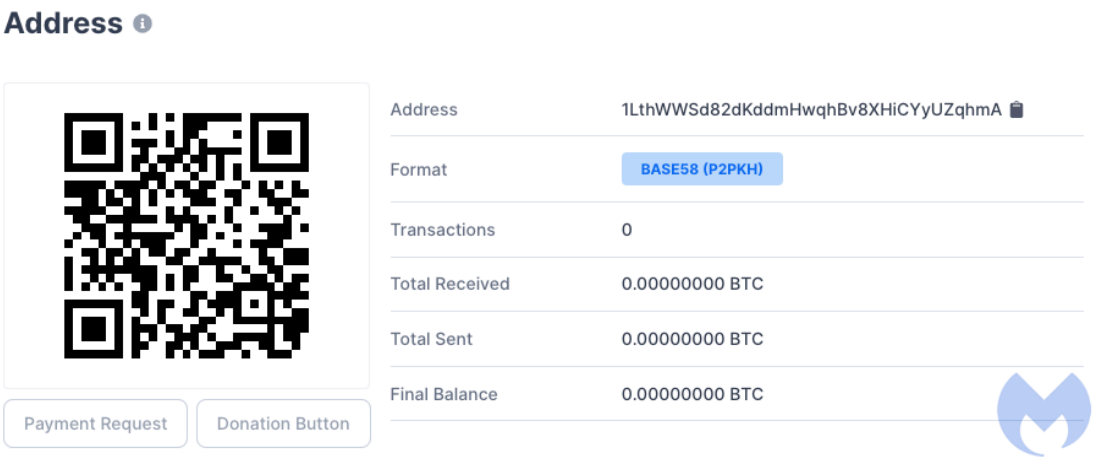
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

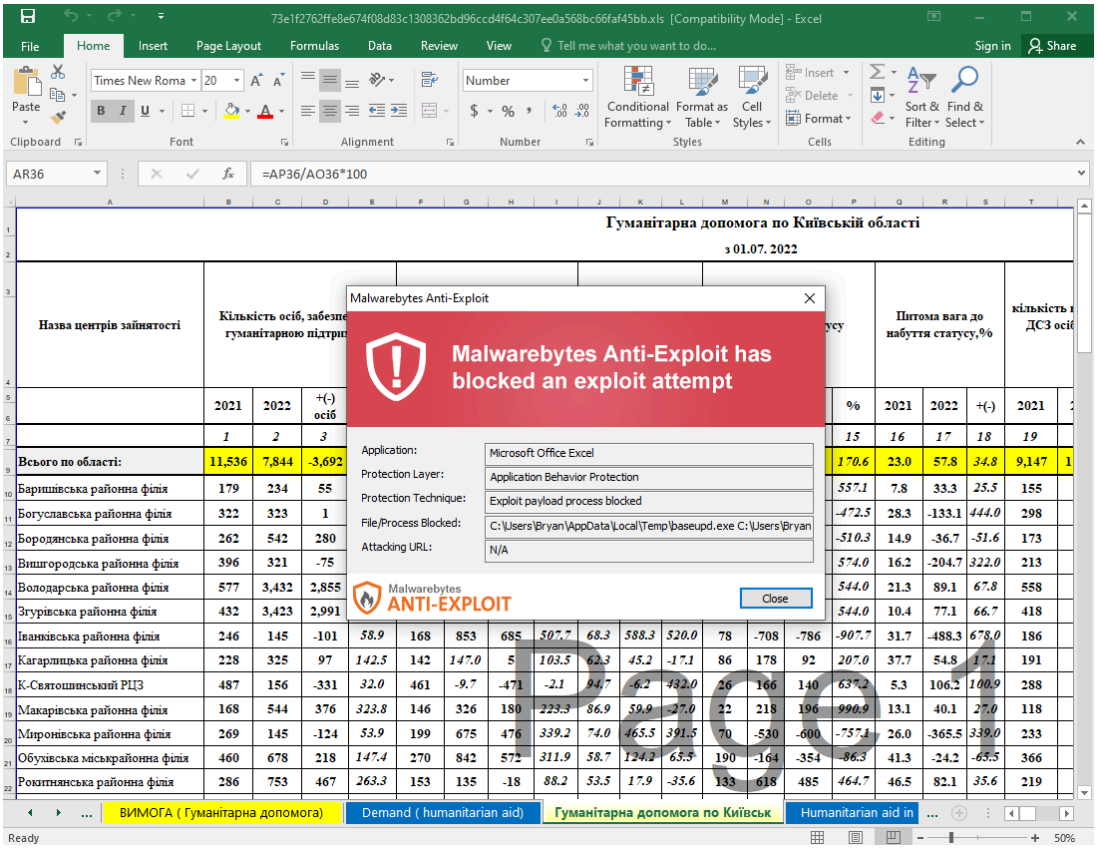
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

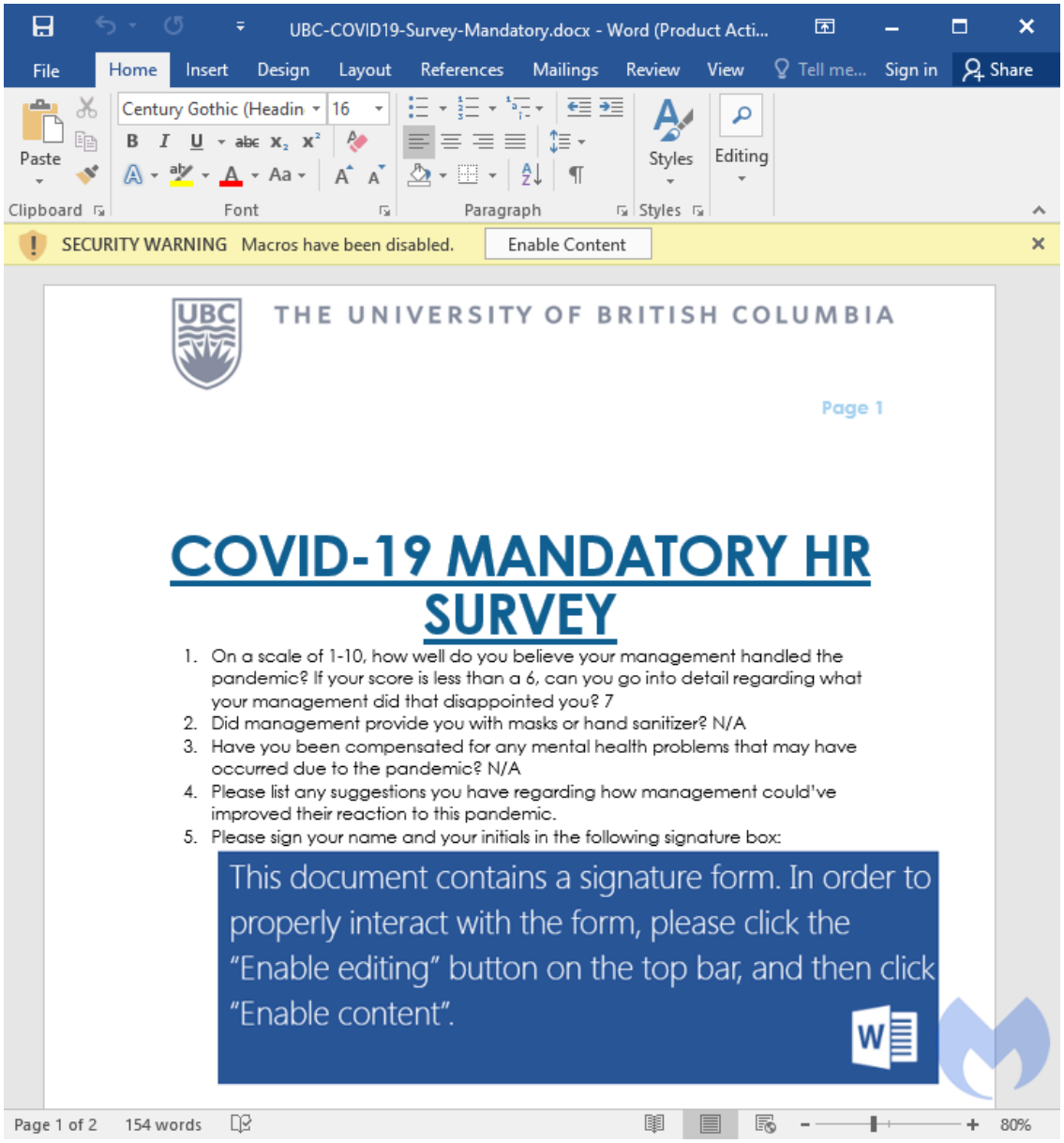
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

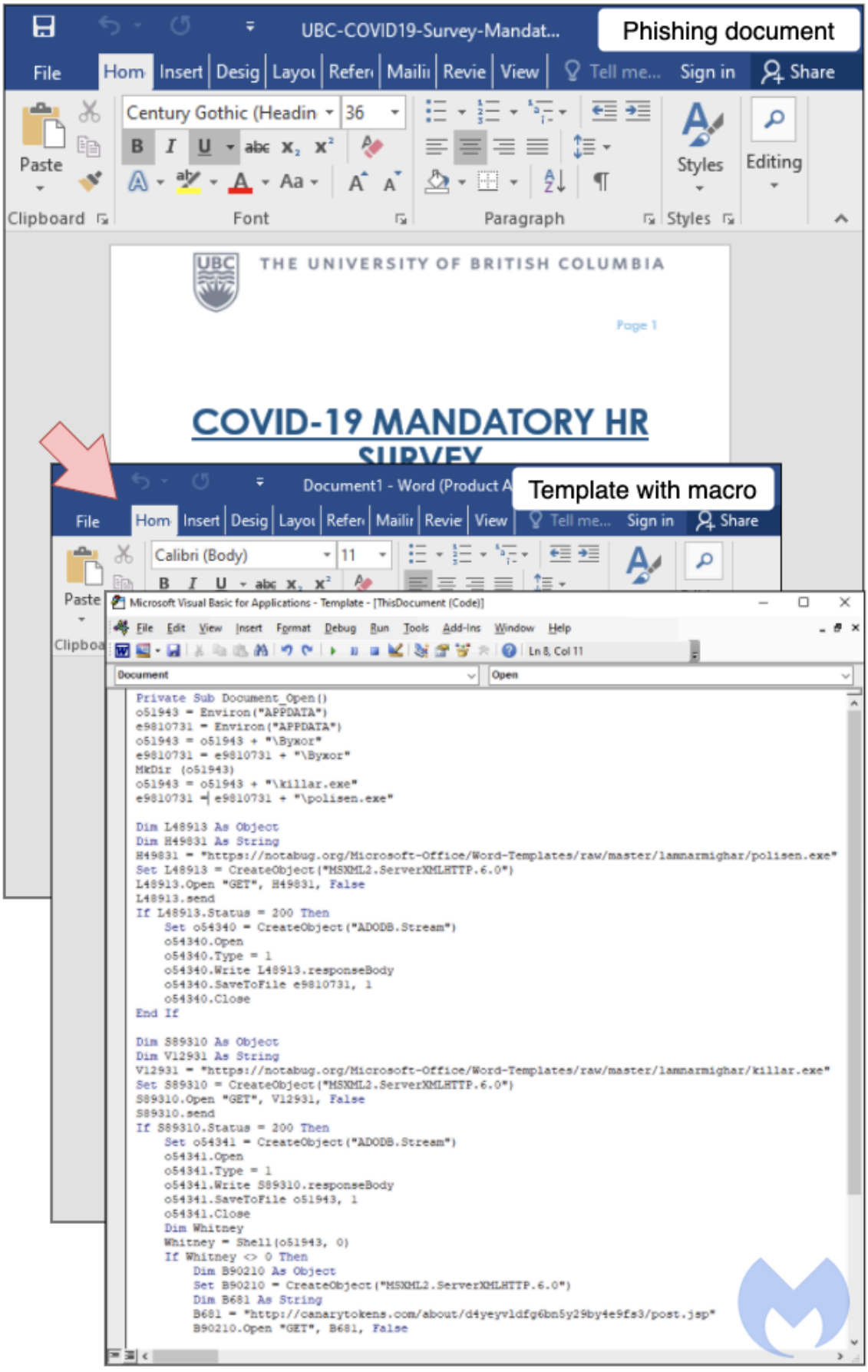
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

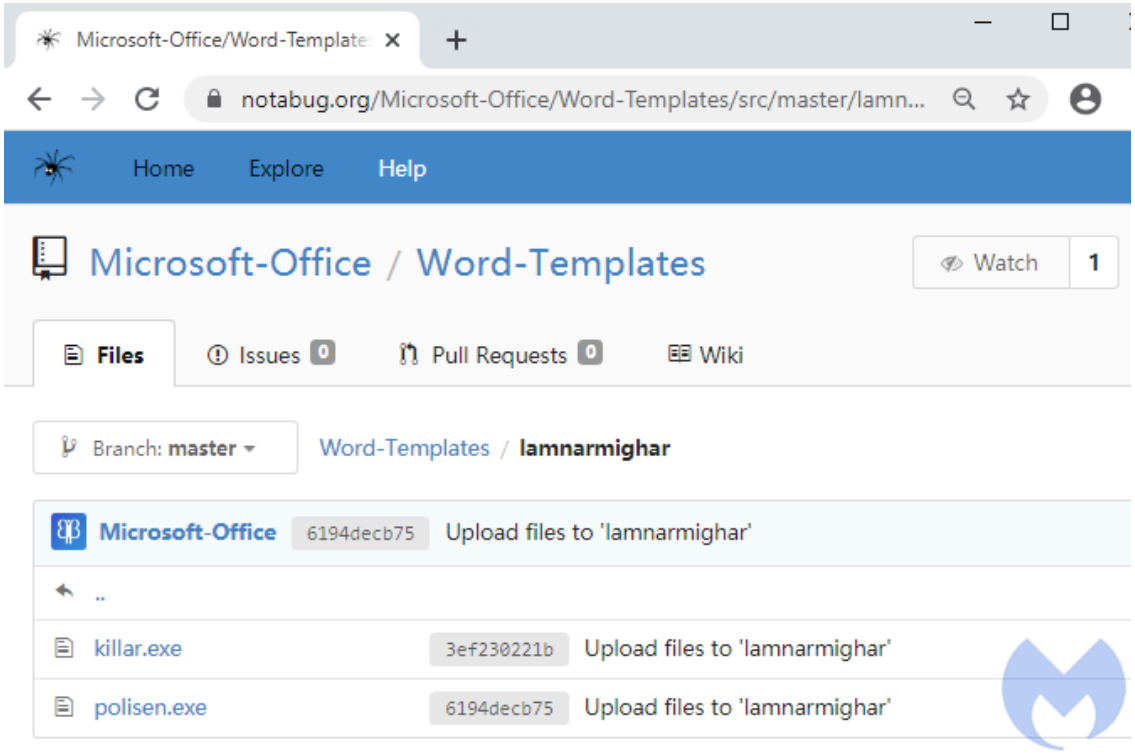
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



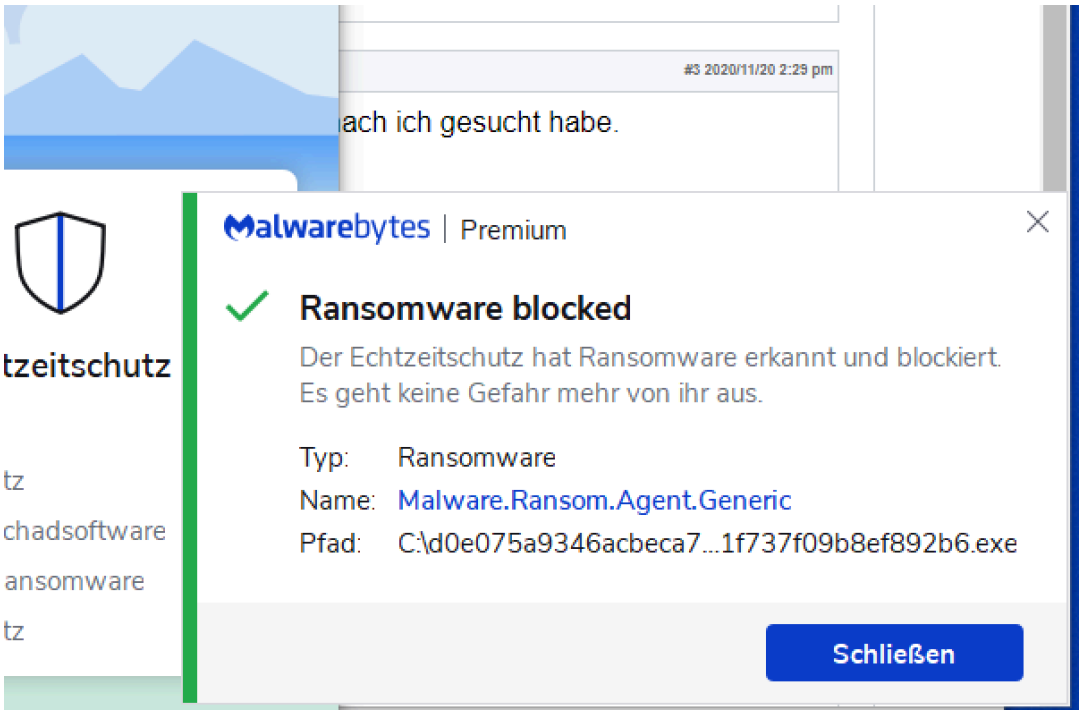
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

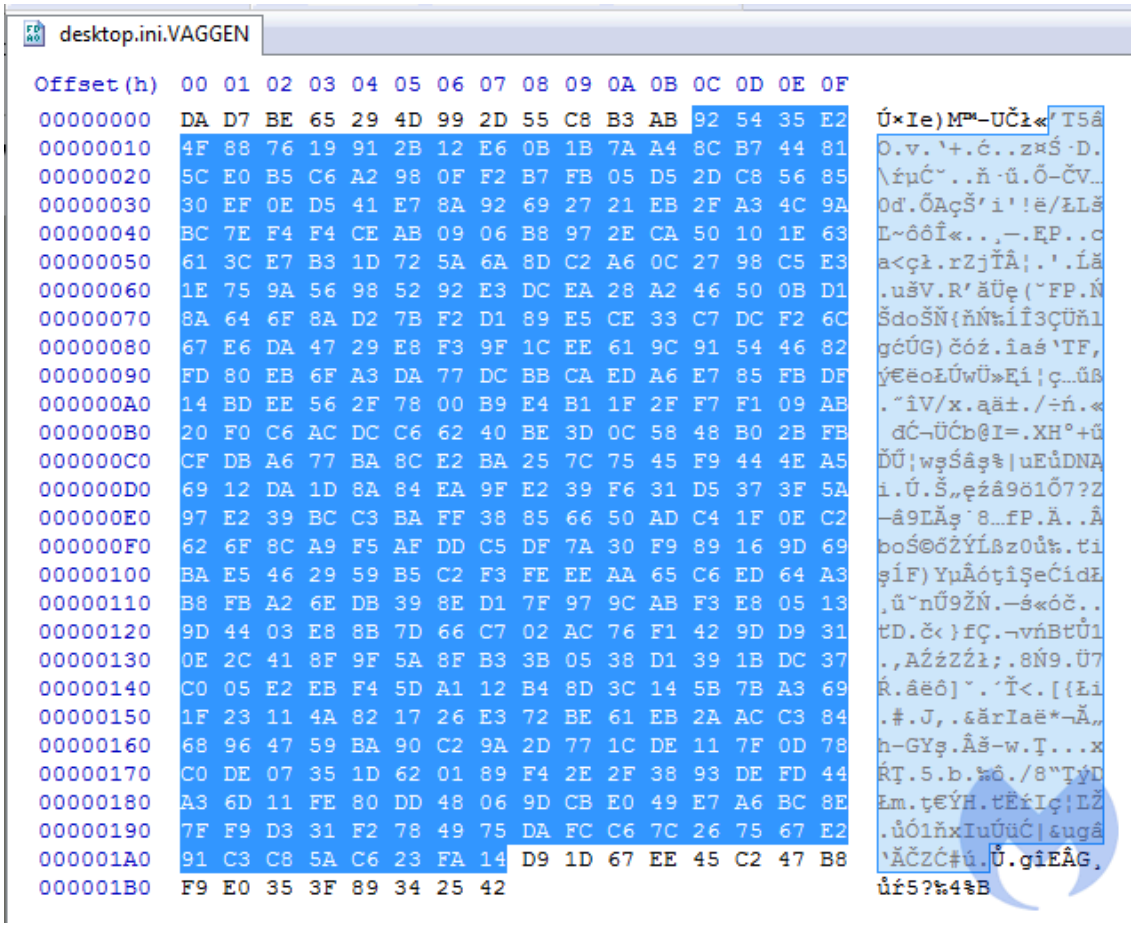
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

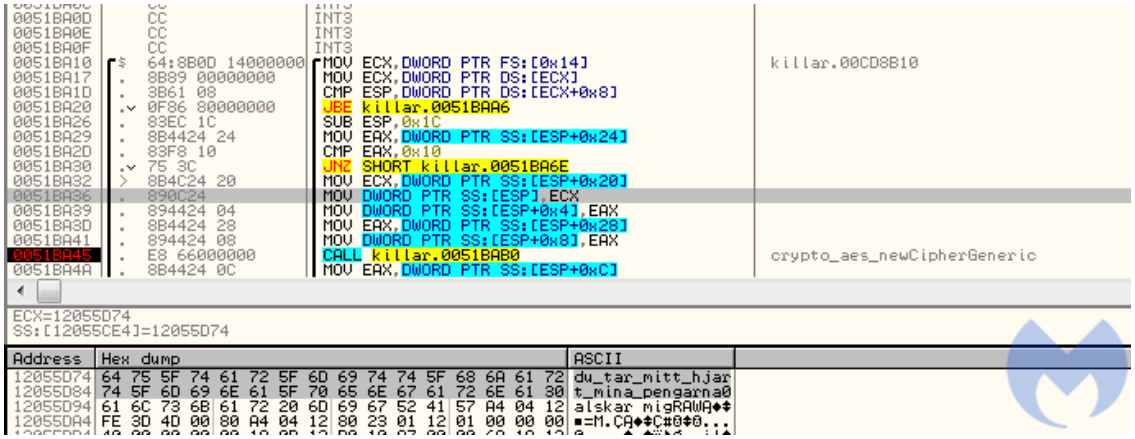


The content of the output file (with .VAGGEN extension) contains:

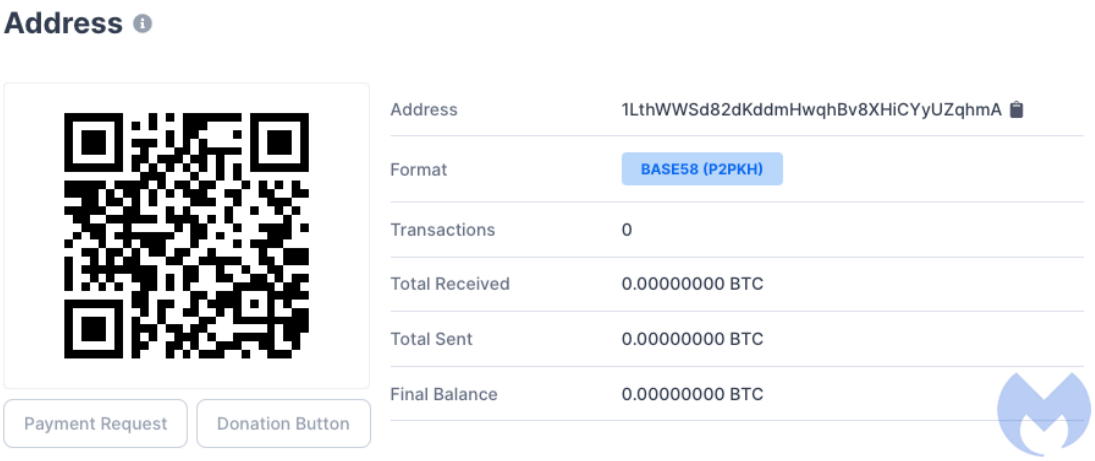
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

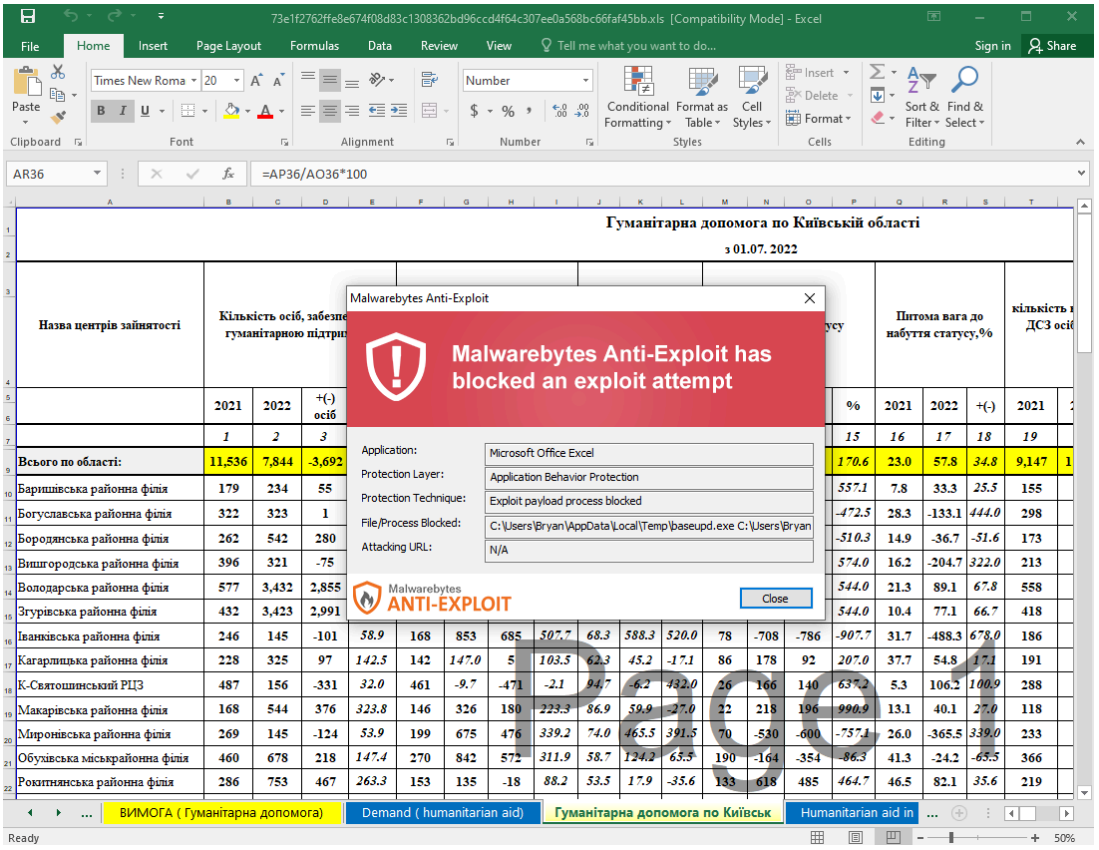
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

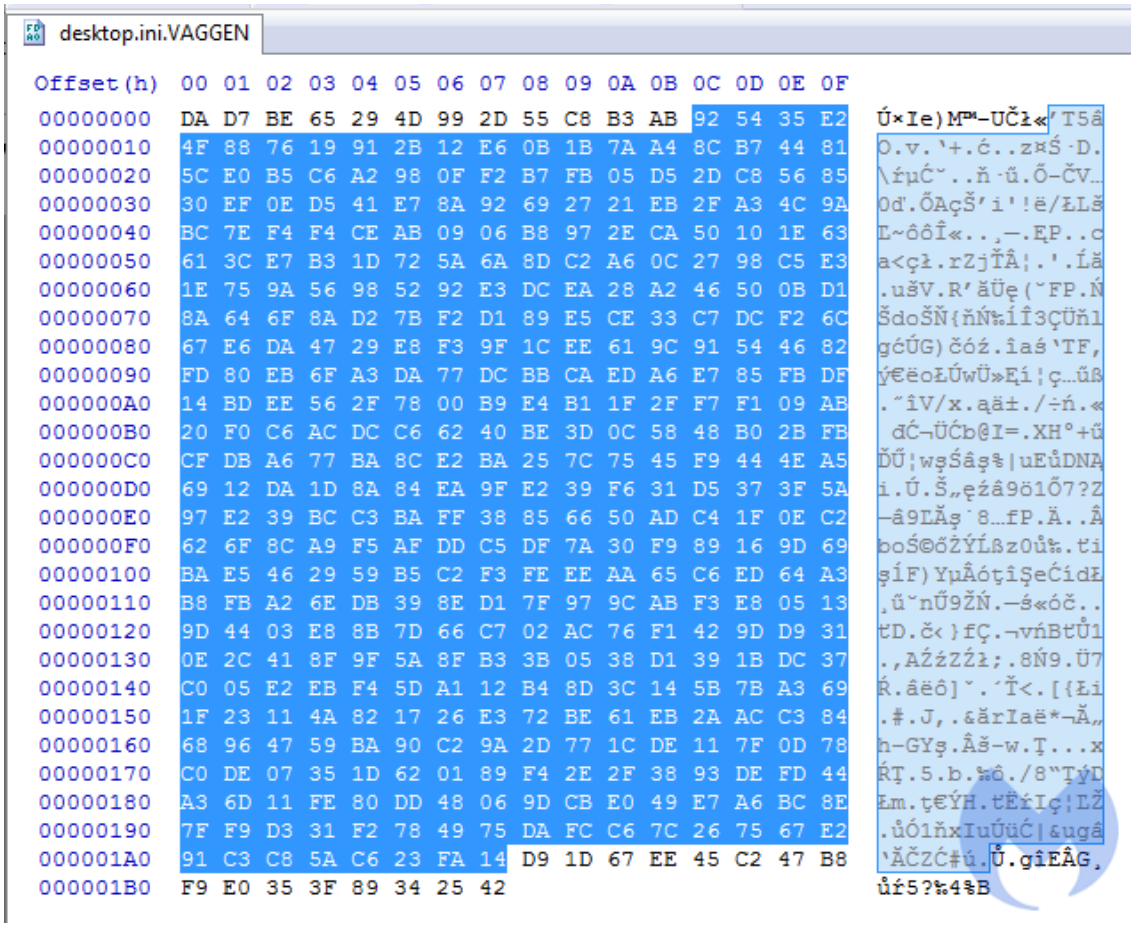
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



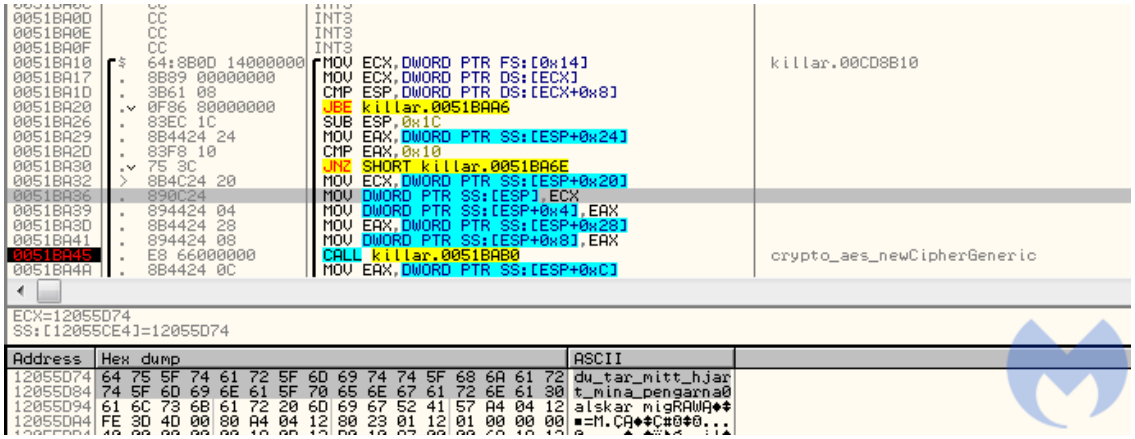
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

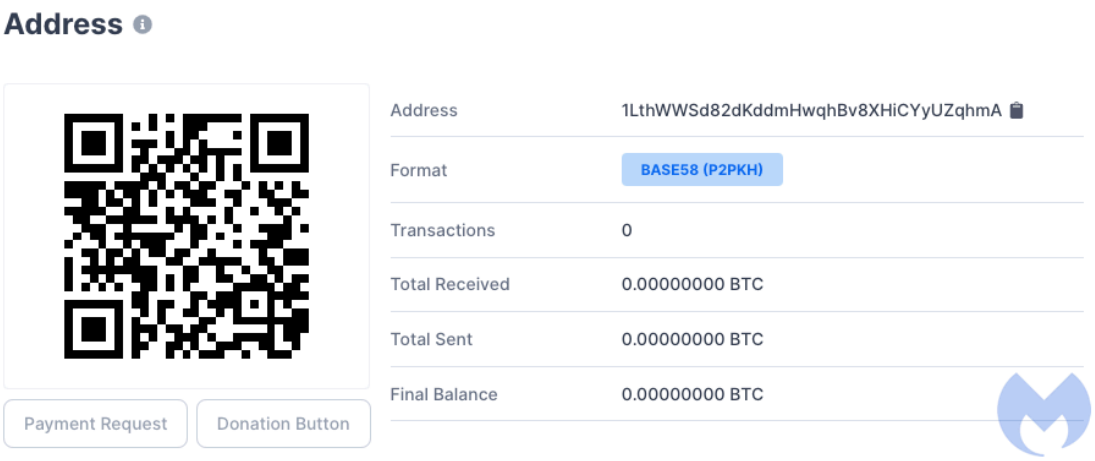
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

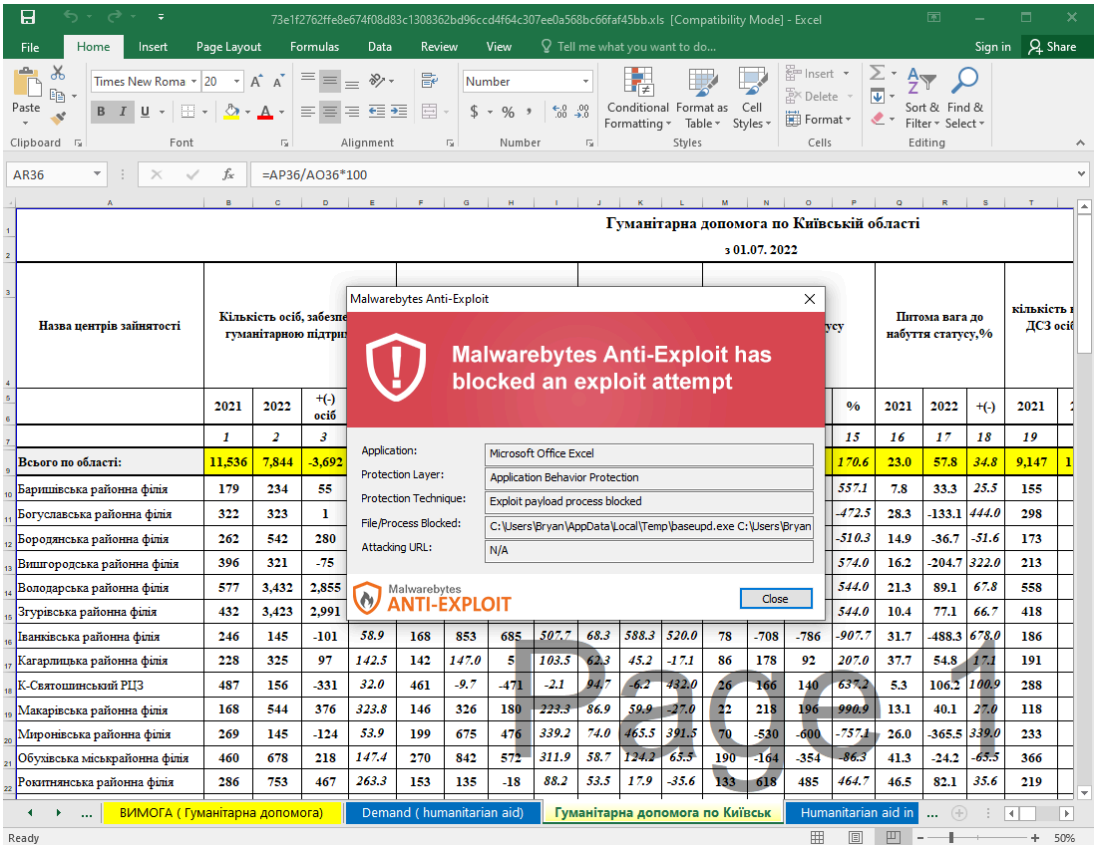
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

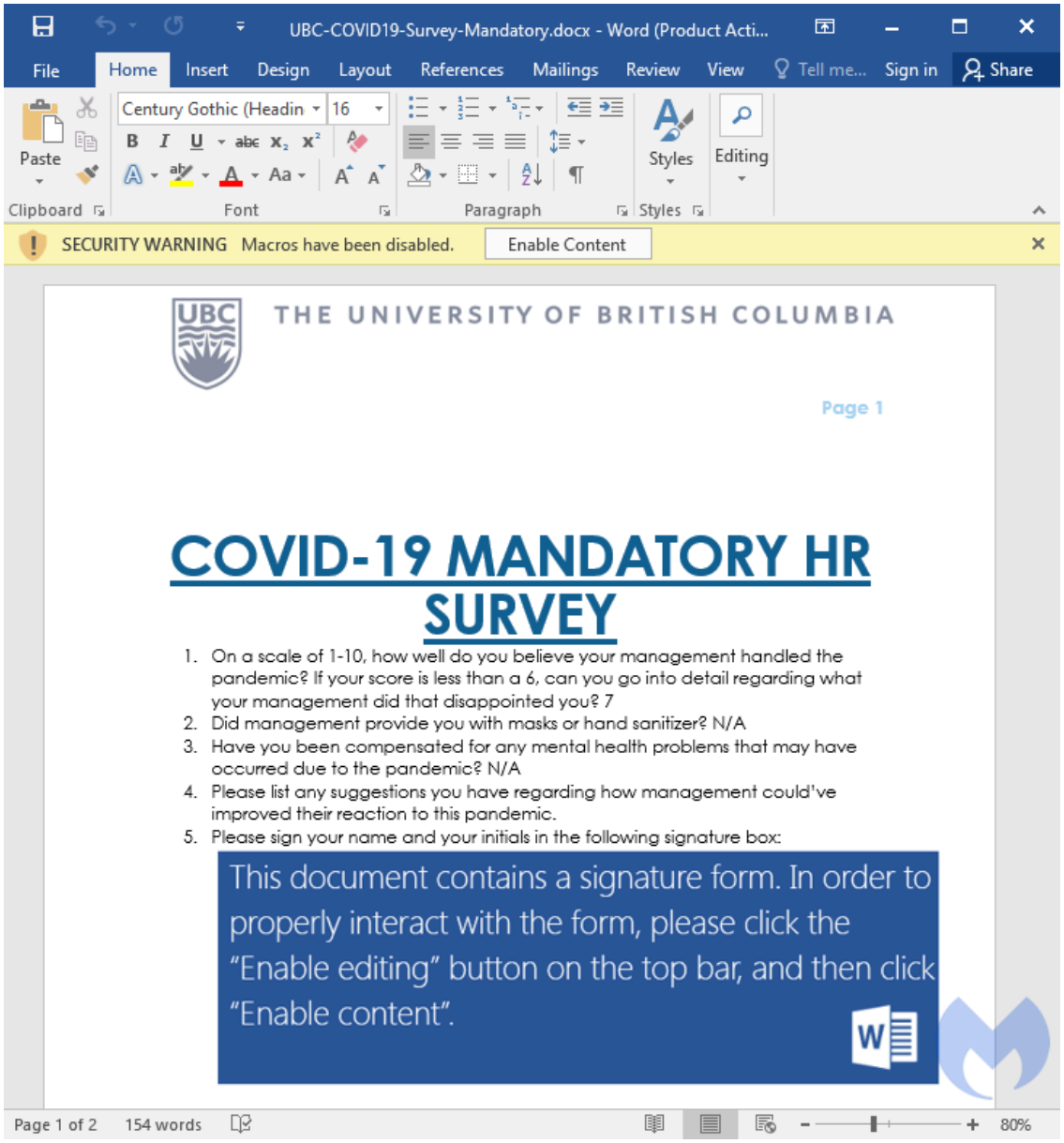
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

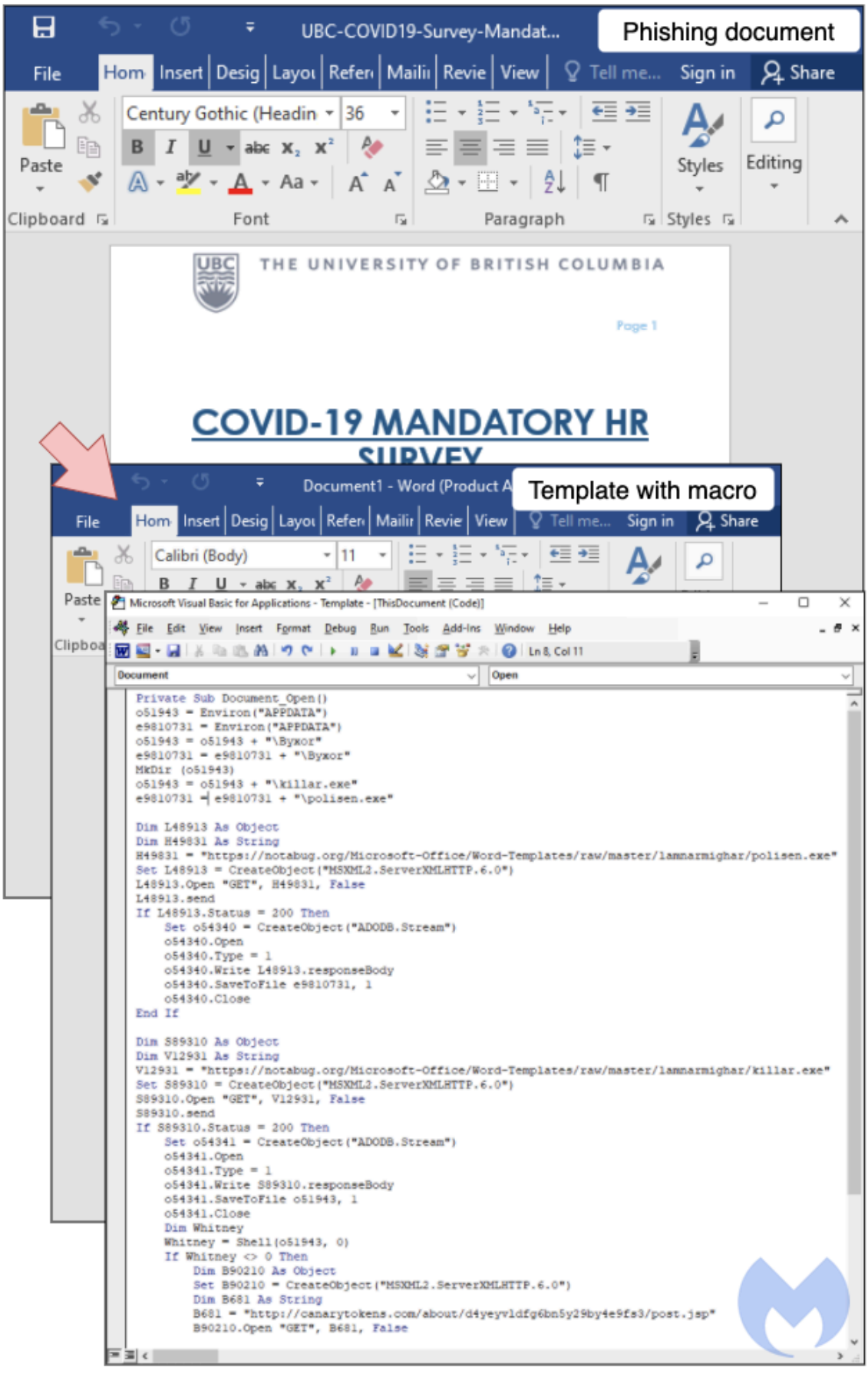
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

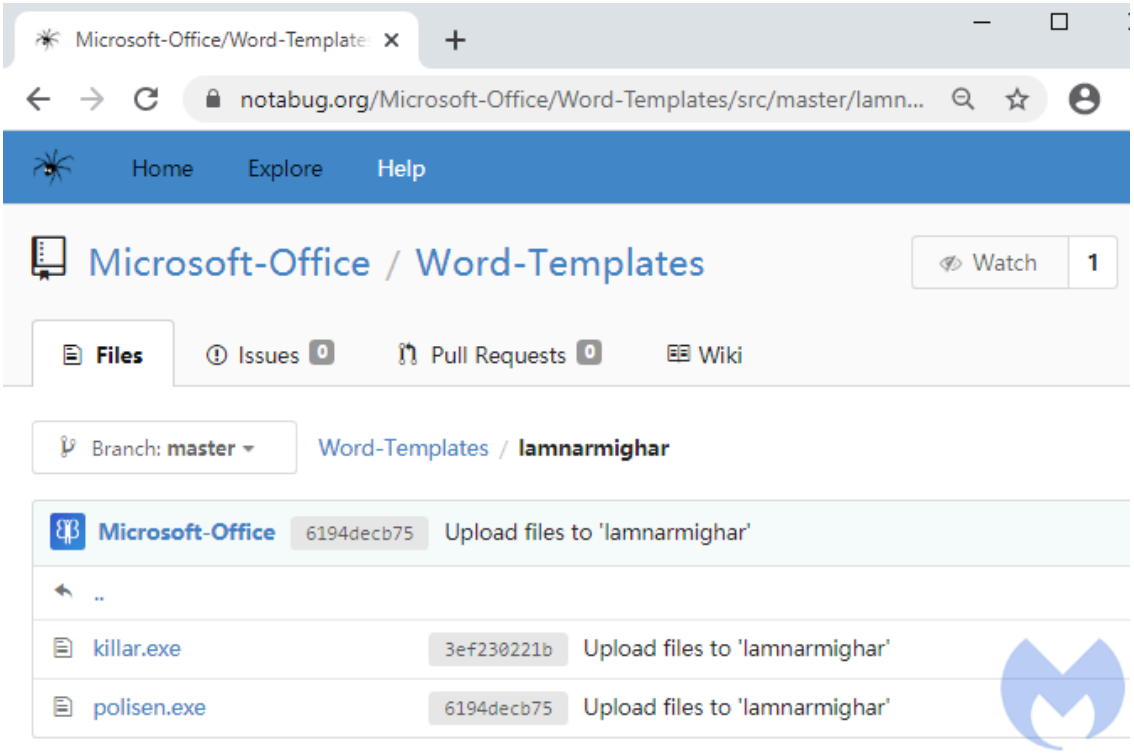
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



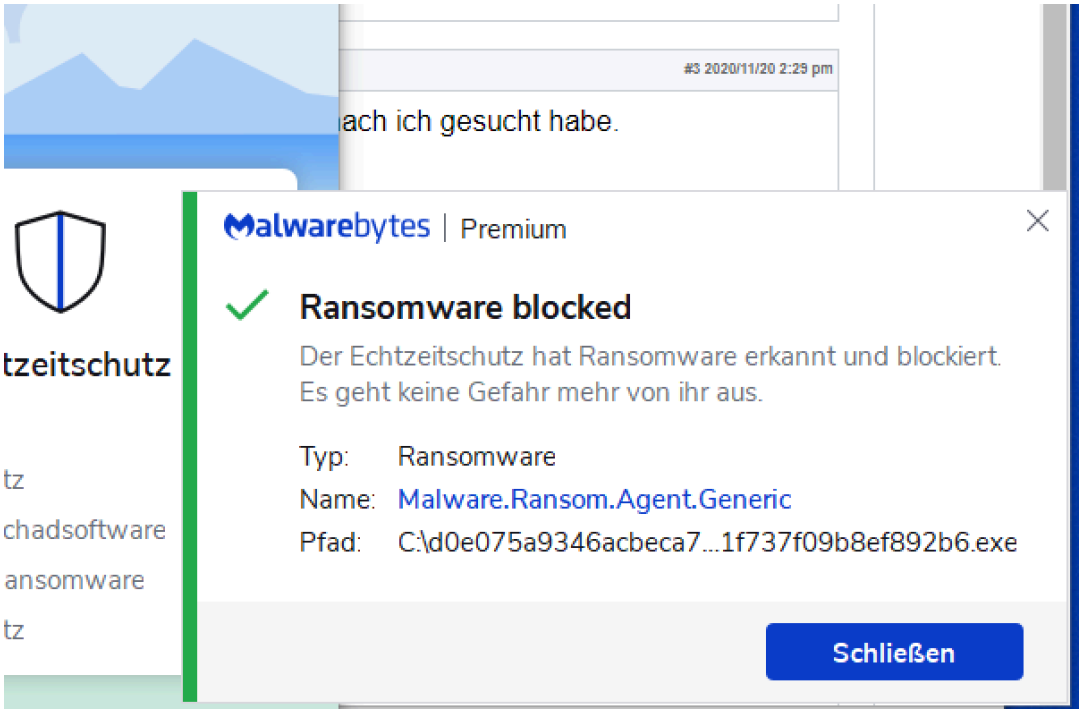
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

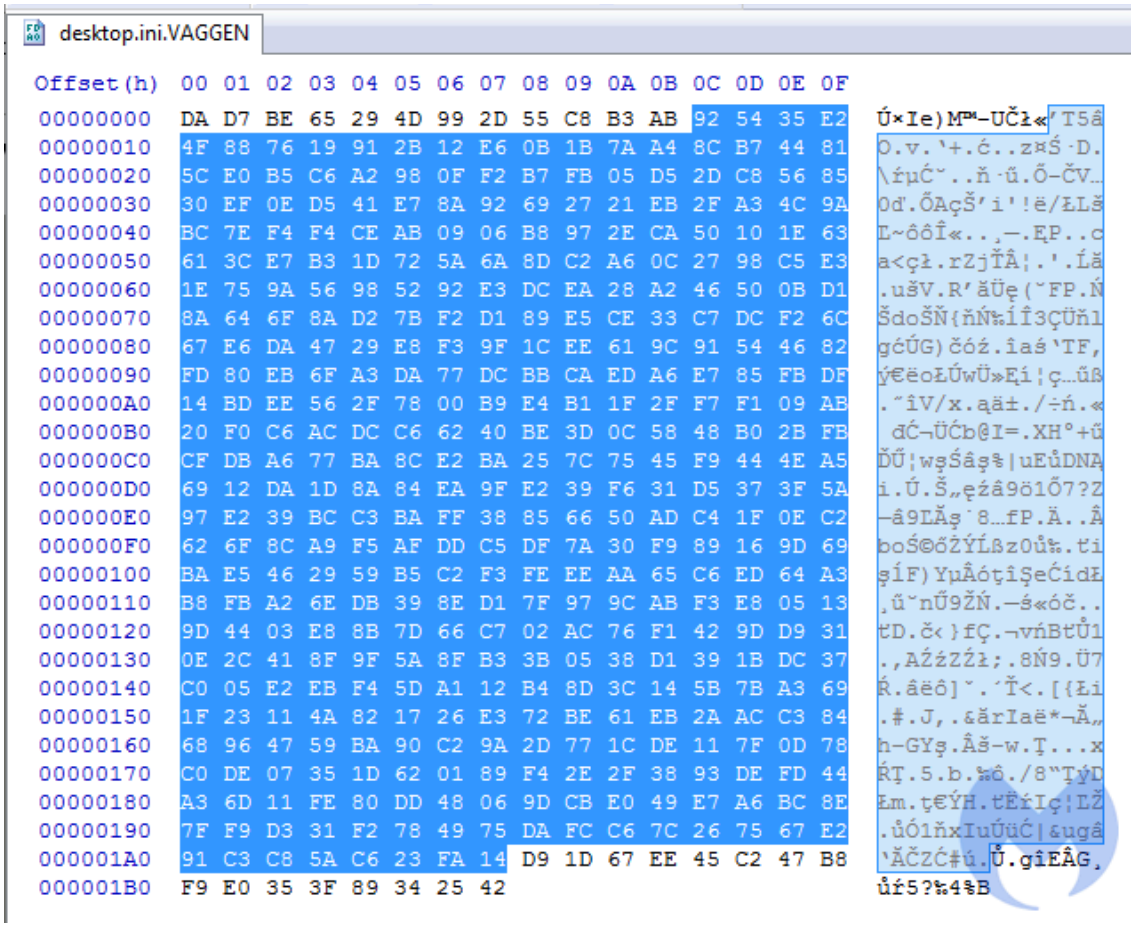
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

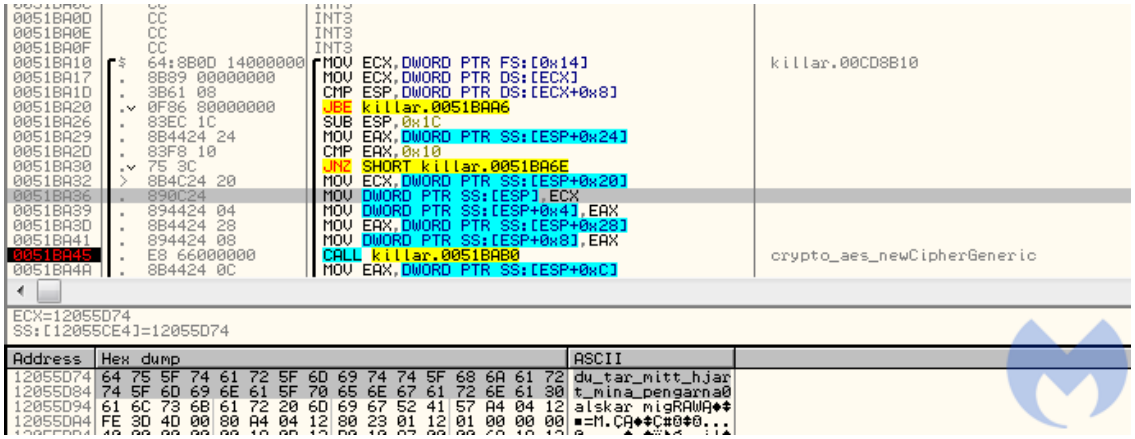


The content of the output file (with .VAGGEN extension) contains:

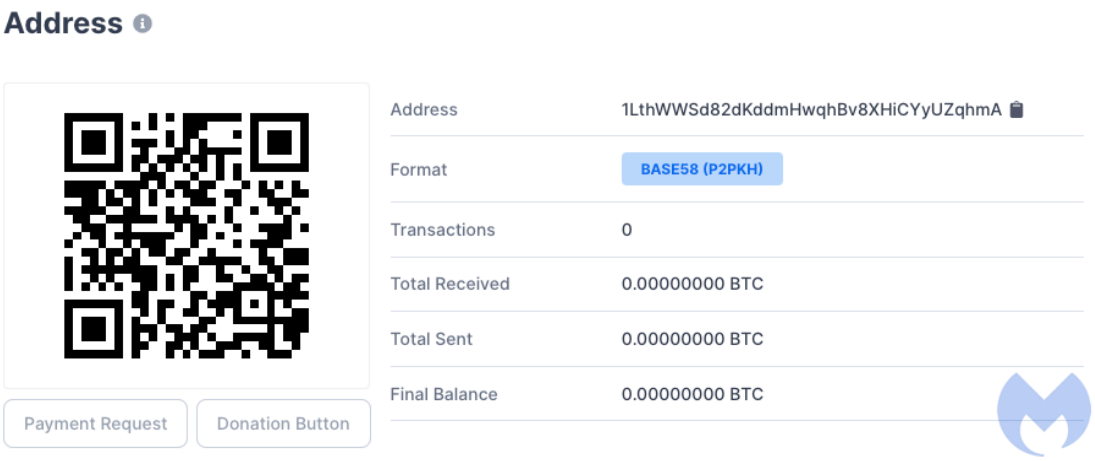
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

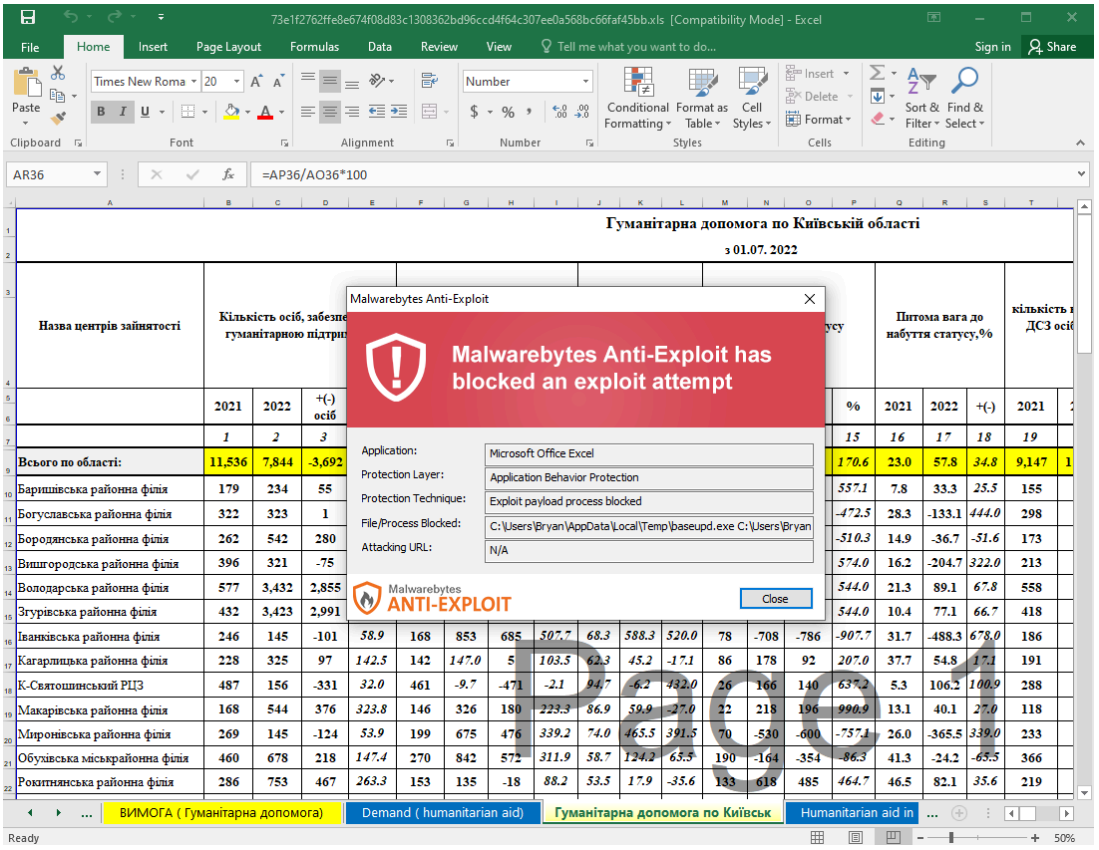
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

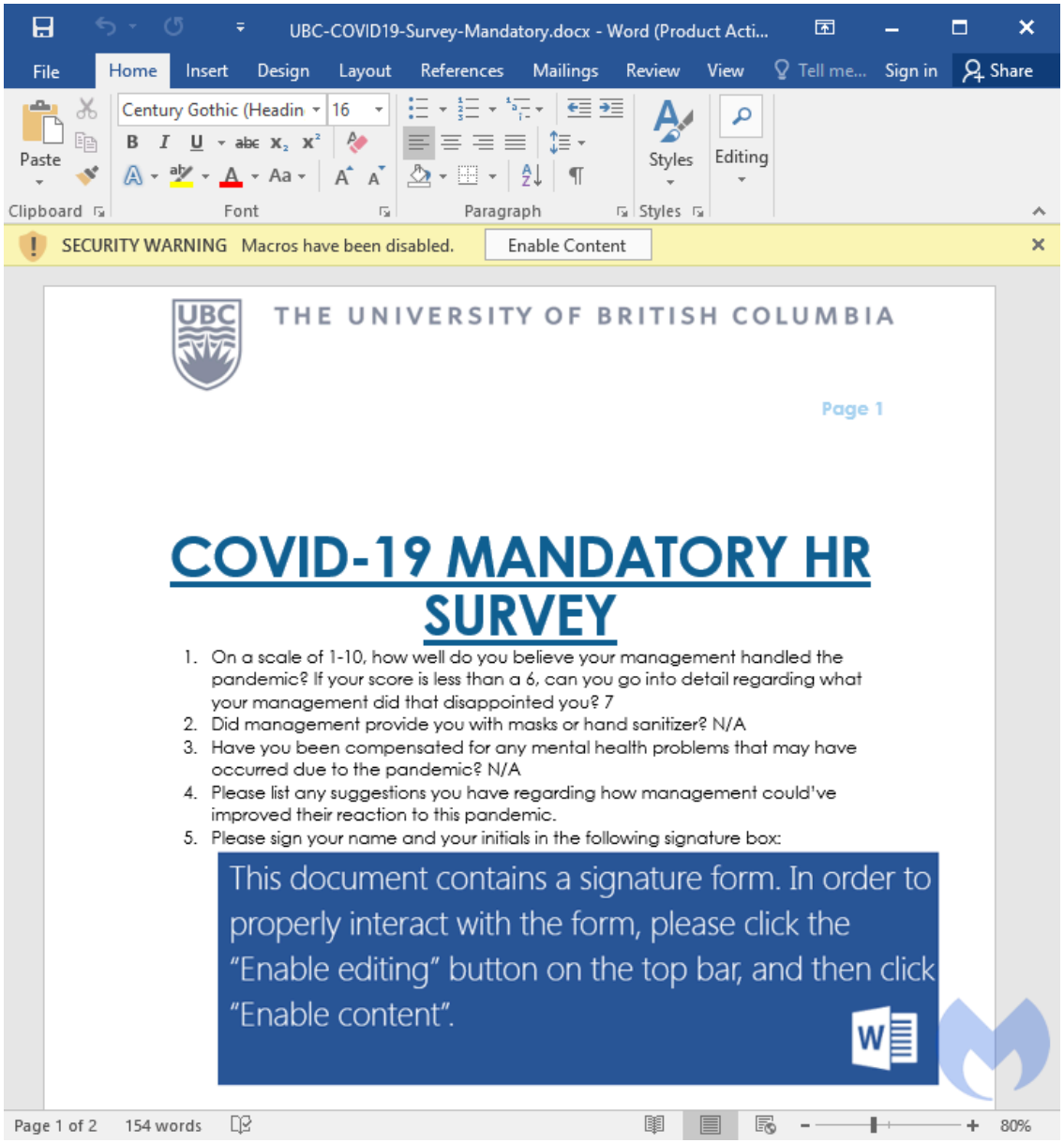
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

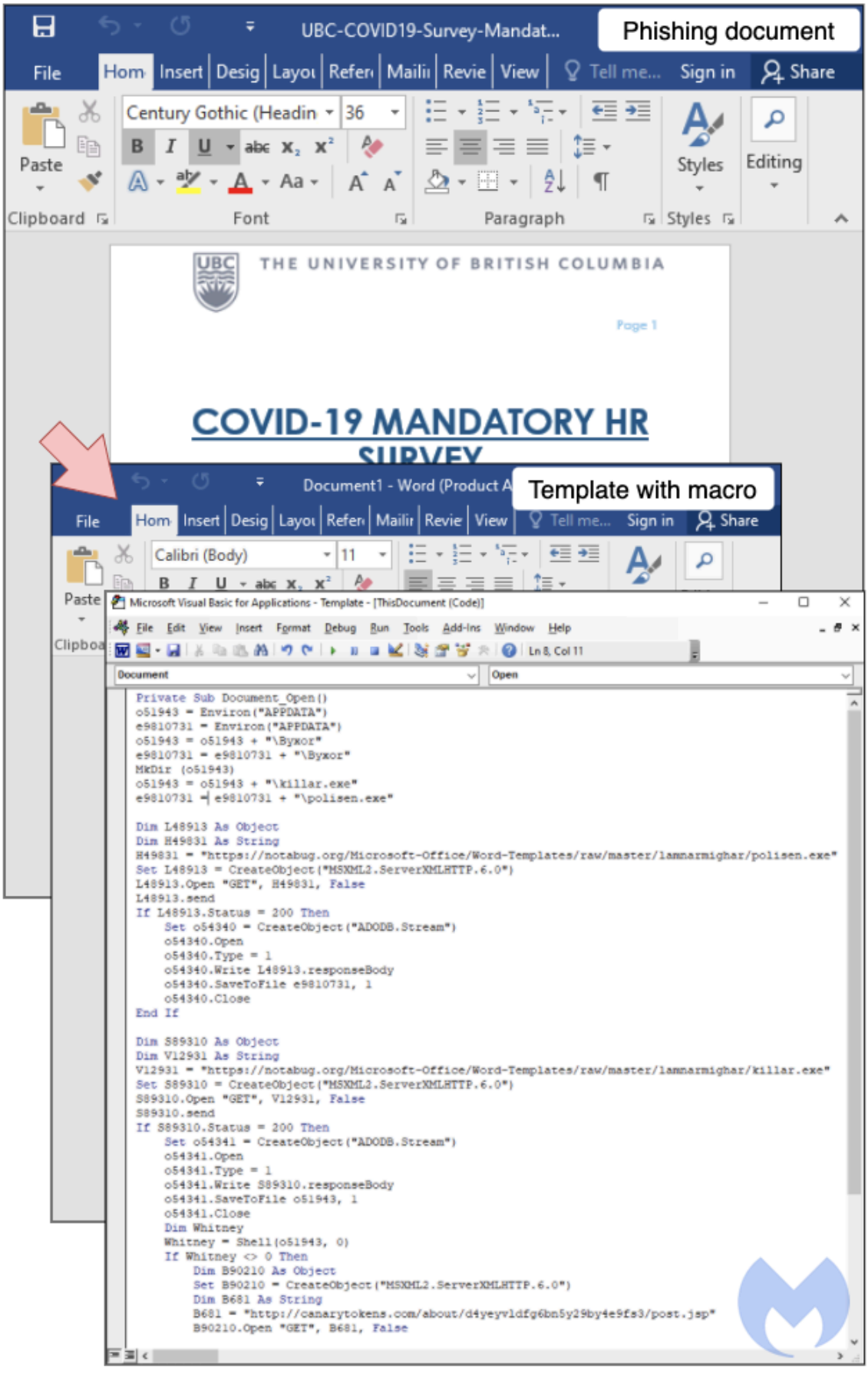
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

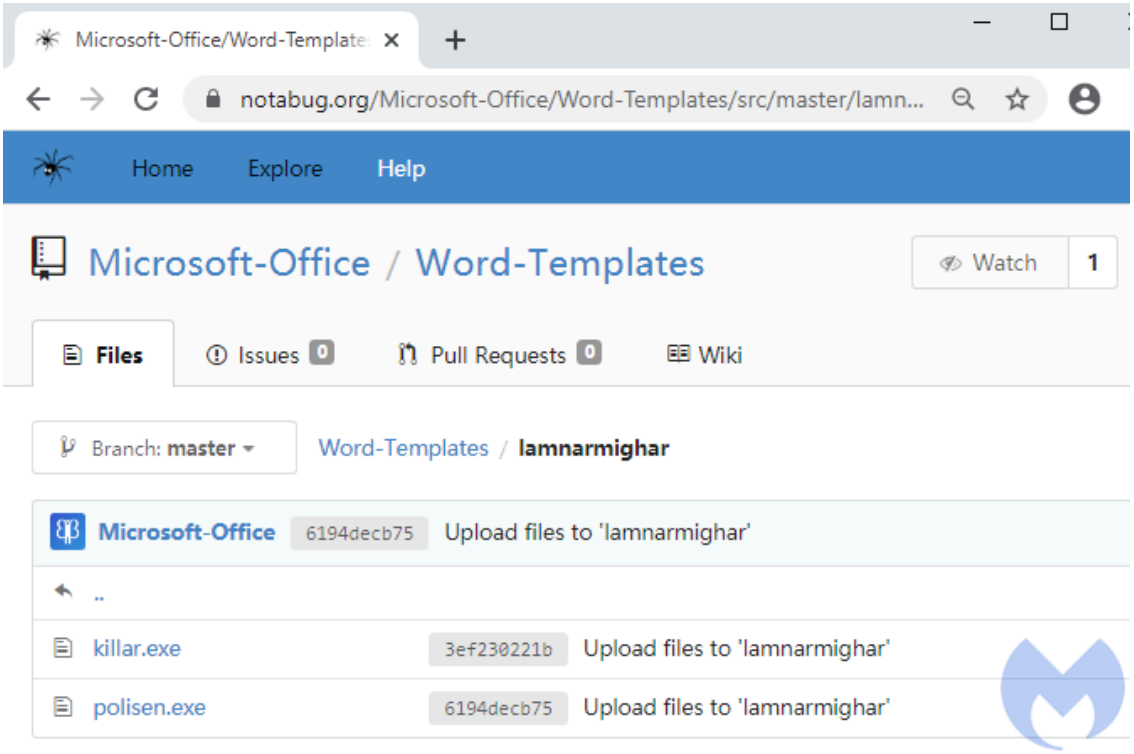
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



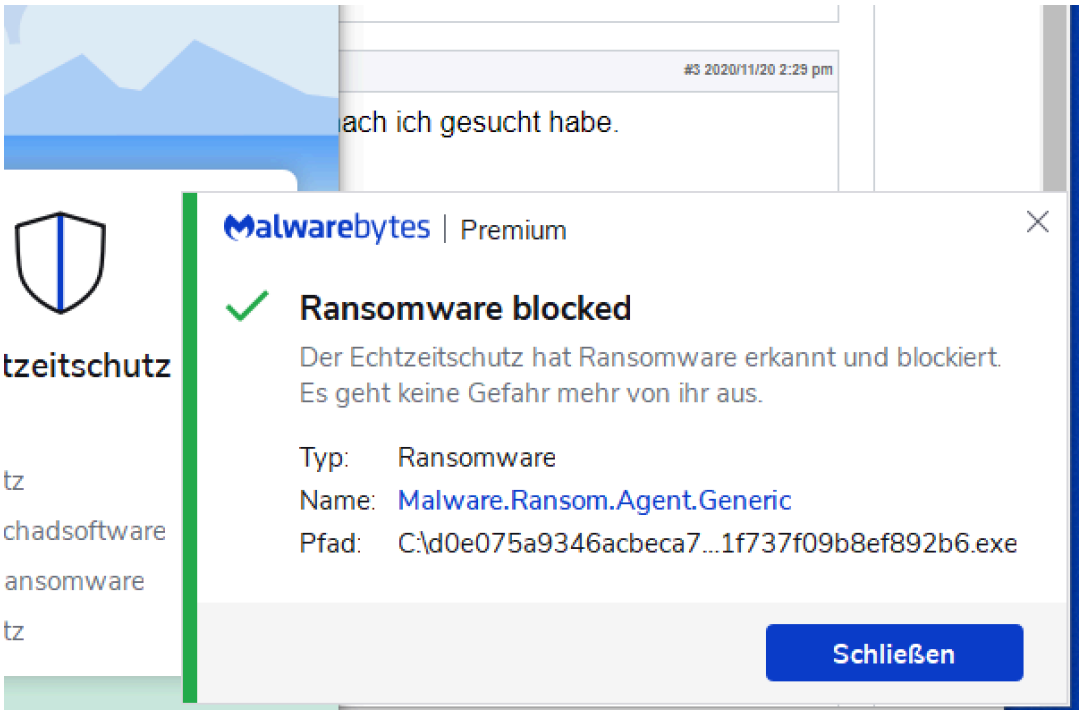
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

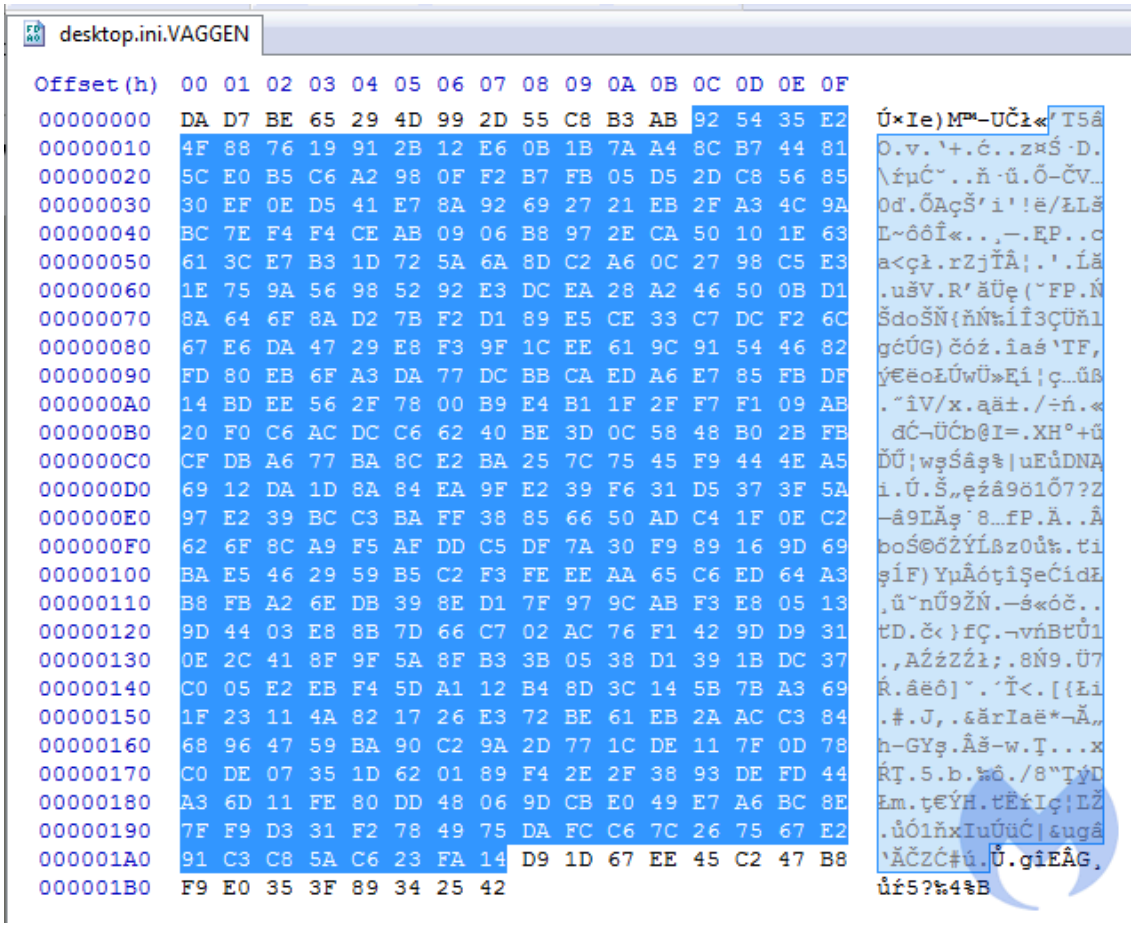
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

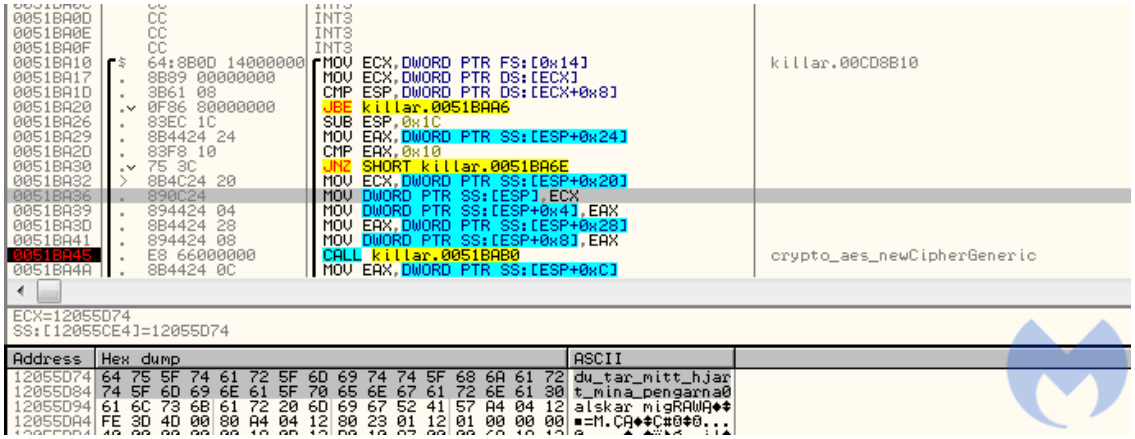


The content of the output file (with .VAGGEN extension) contains:

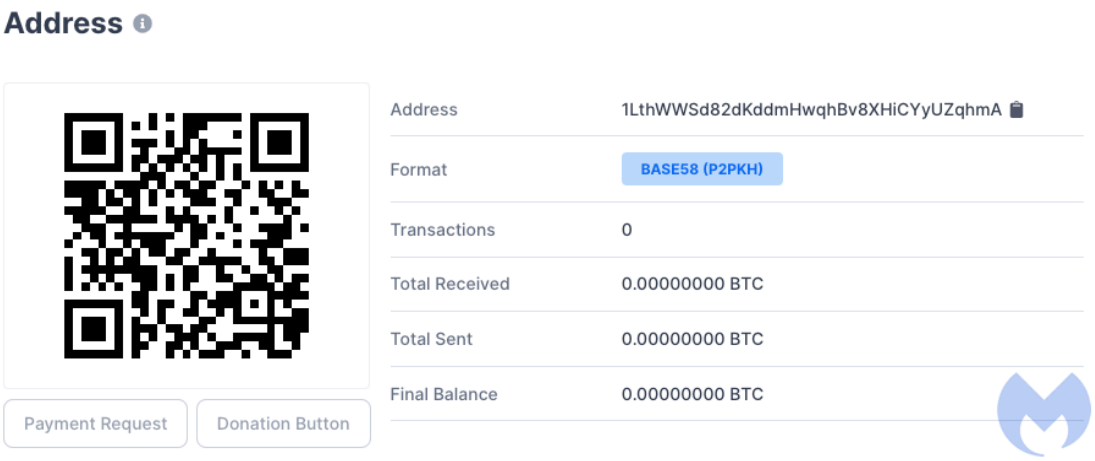
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

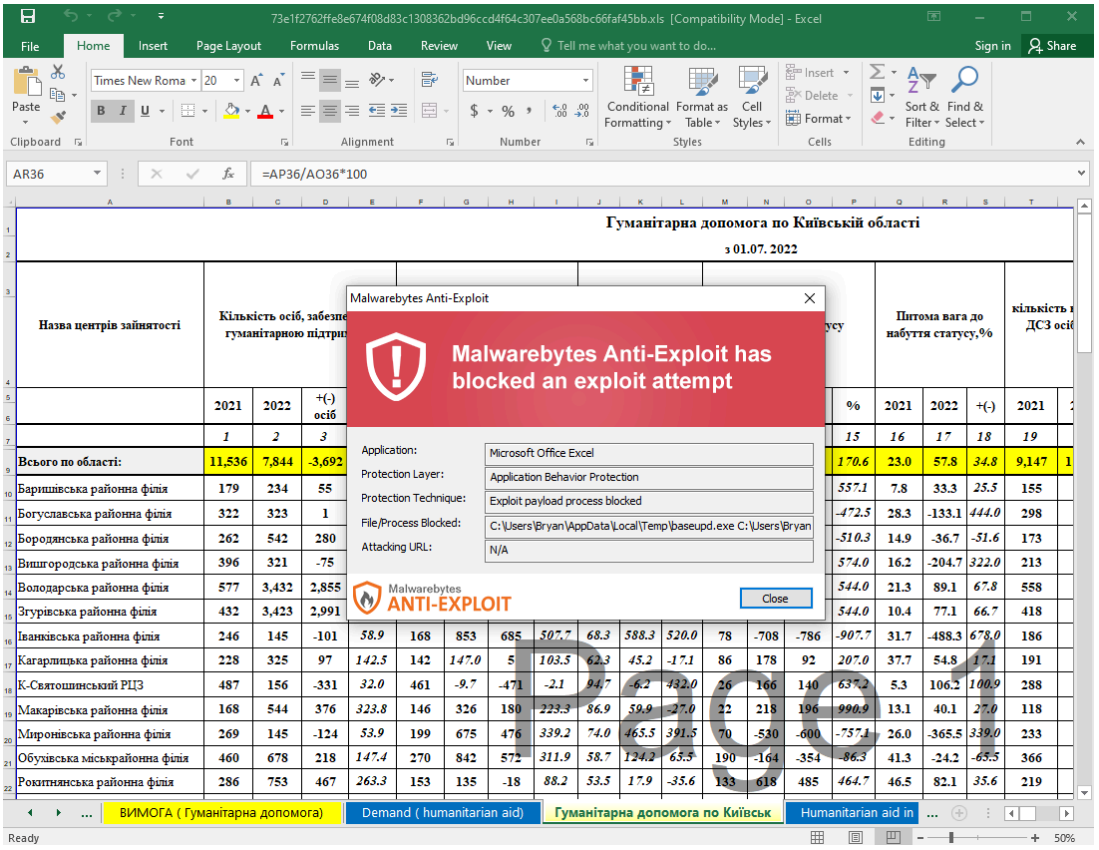
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

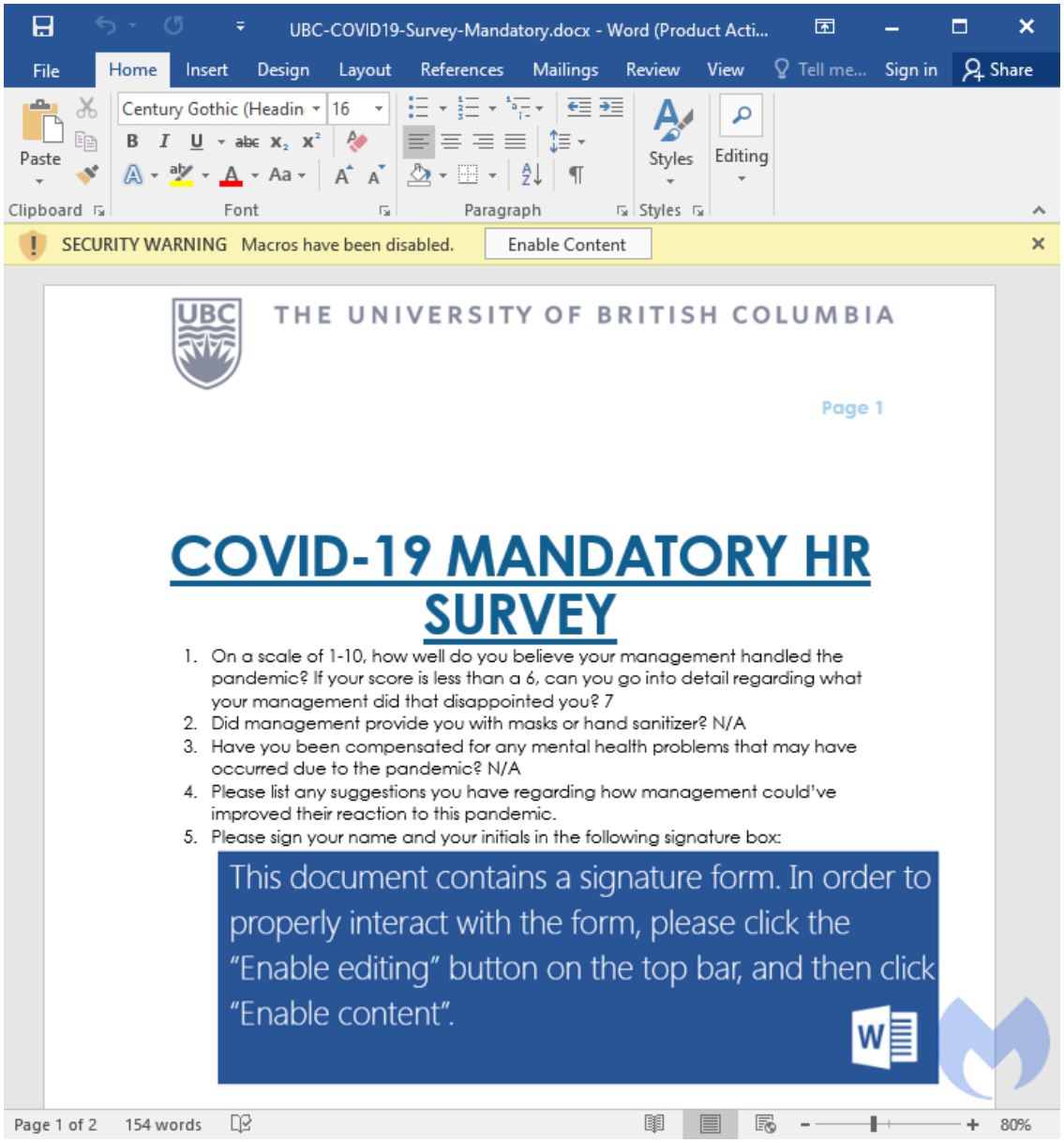
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

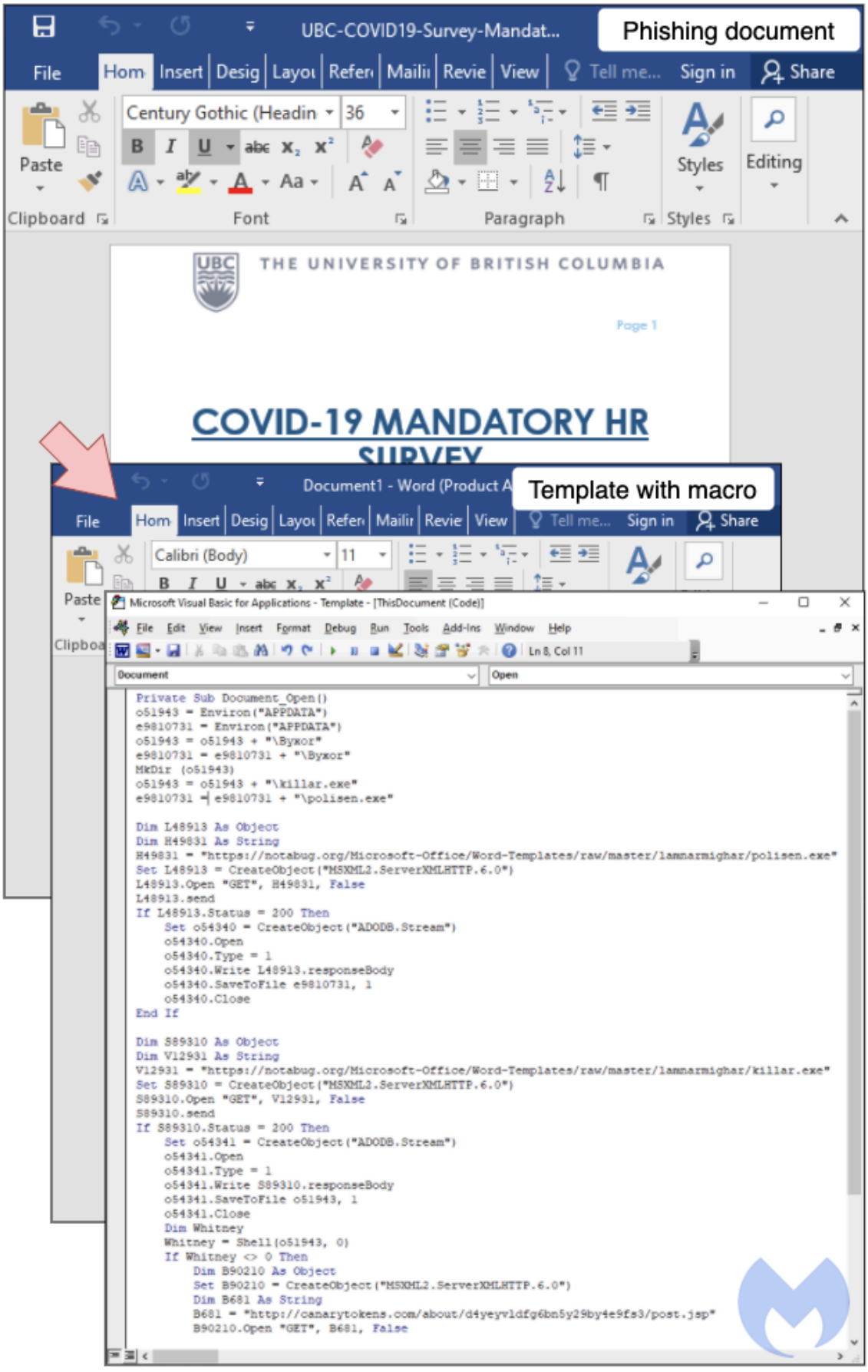
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

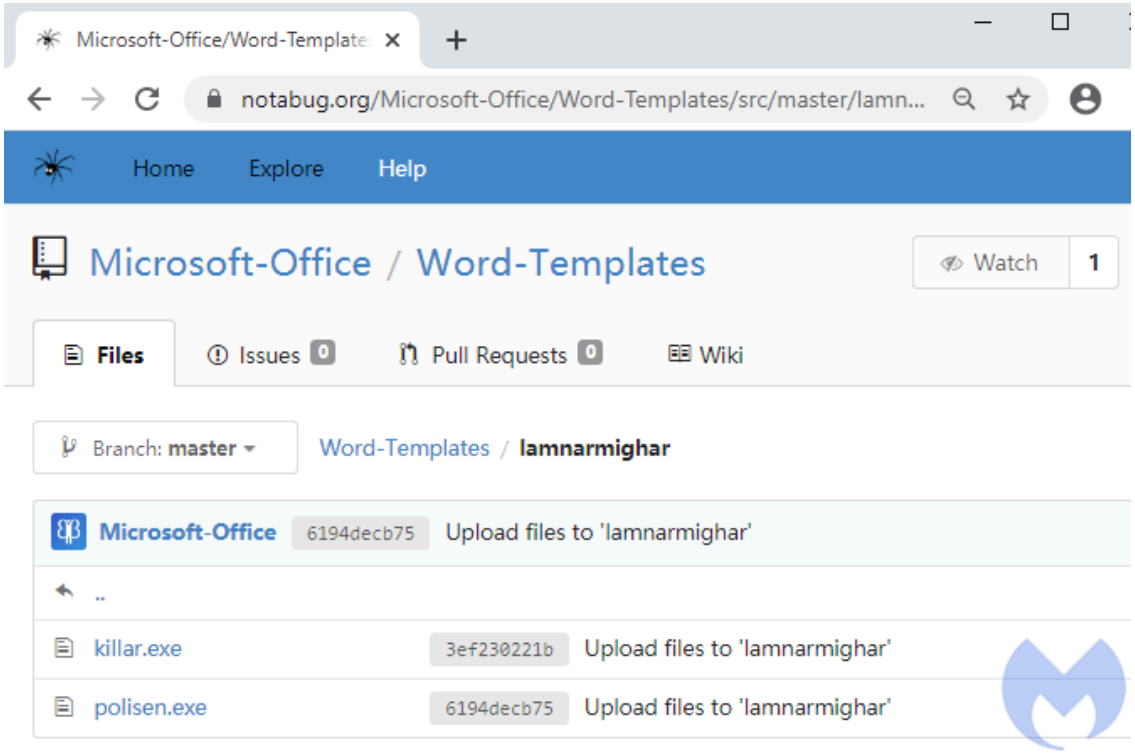
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



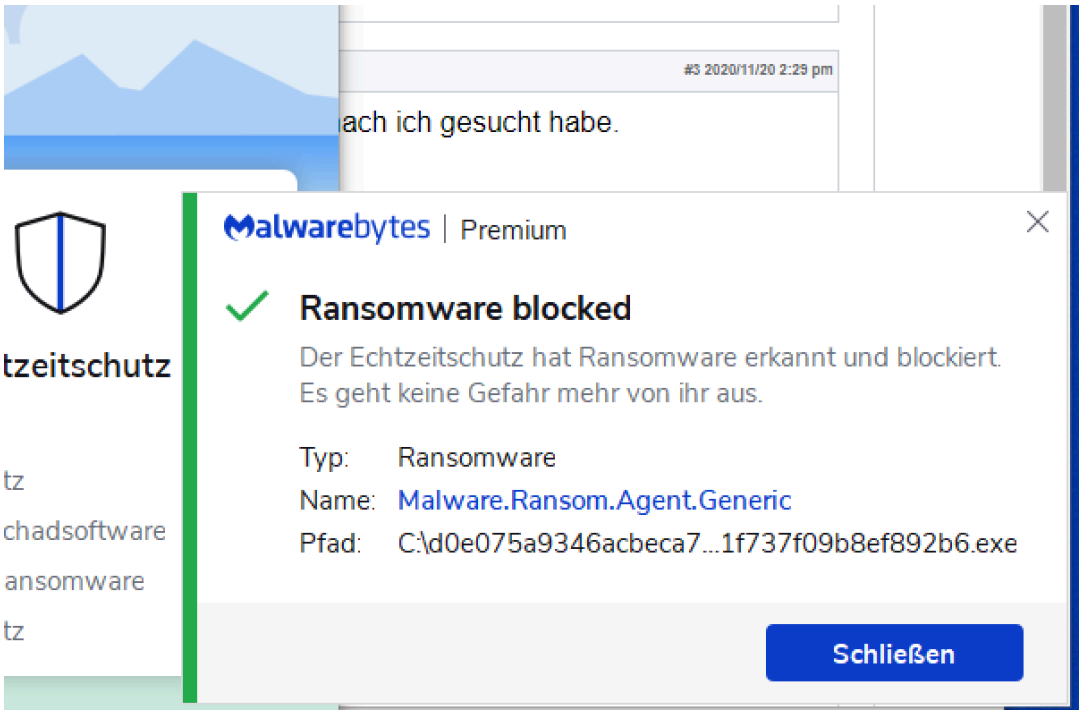
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

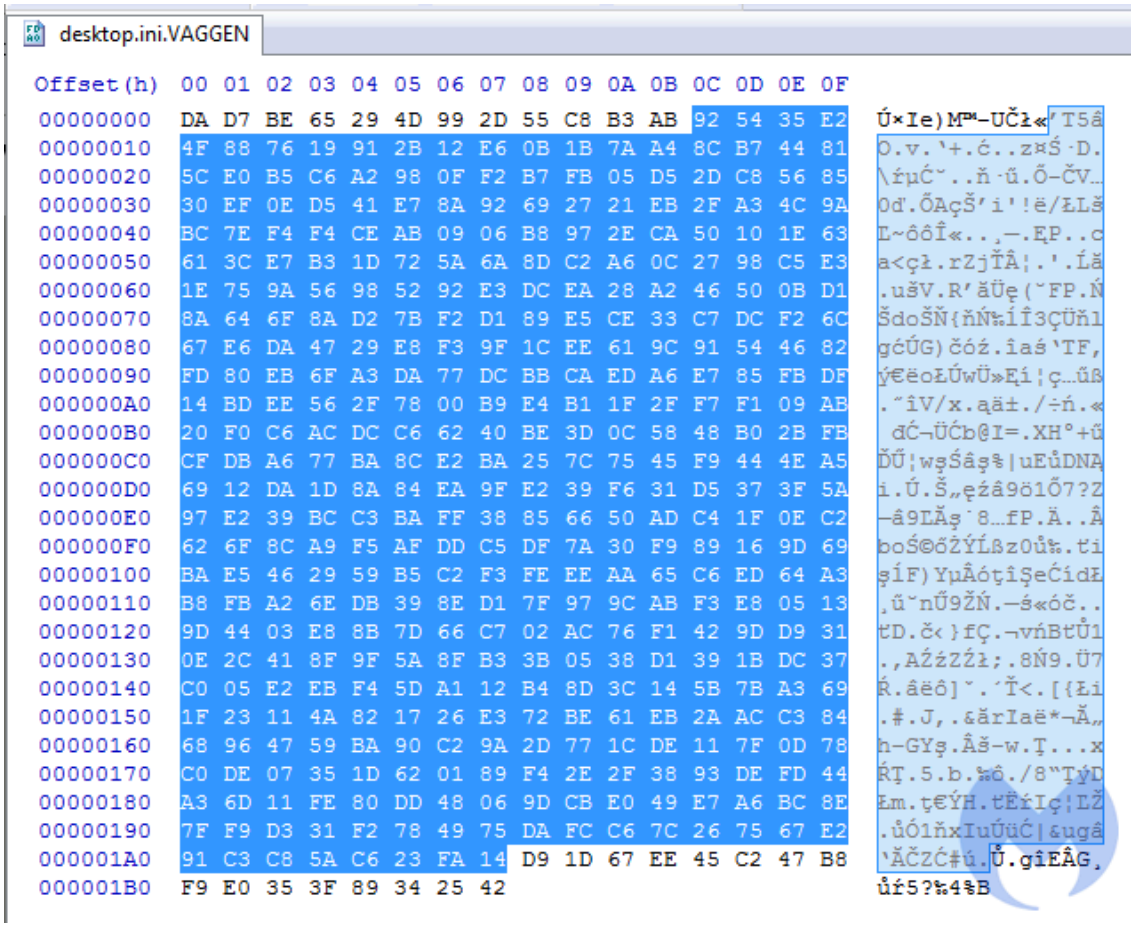
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

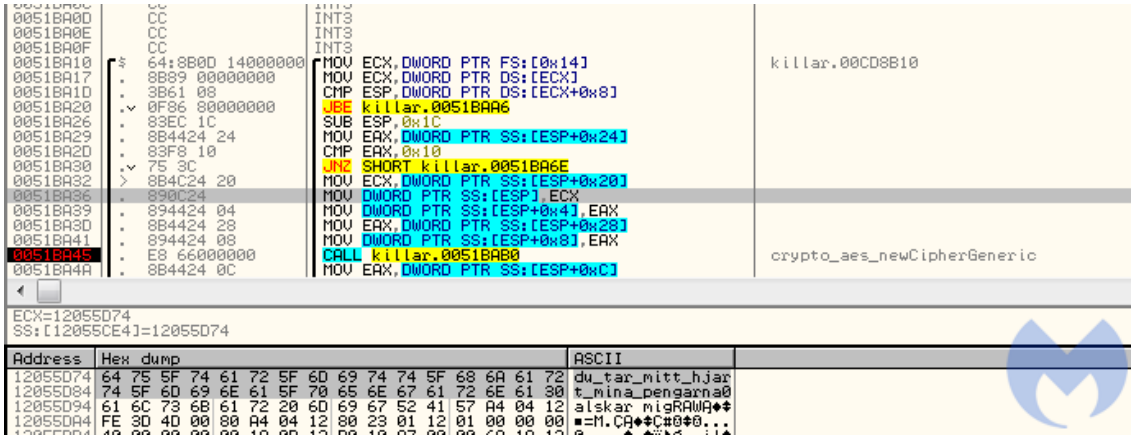


The content of the output file (with .VAGGEN extension) contains:

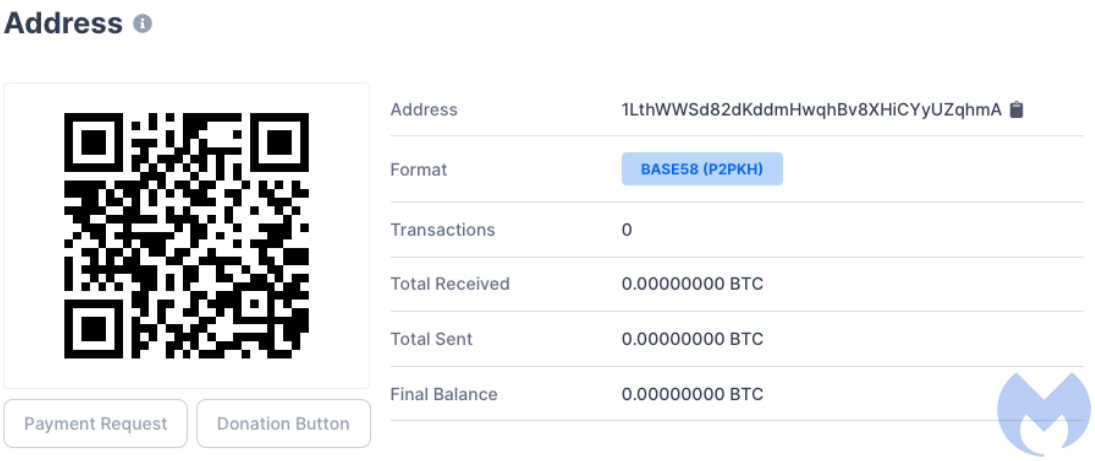
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

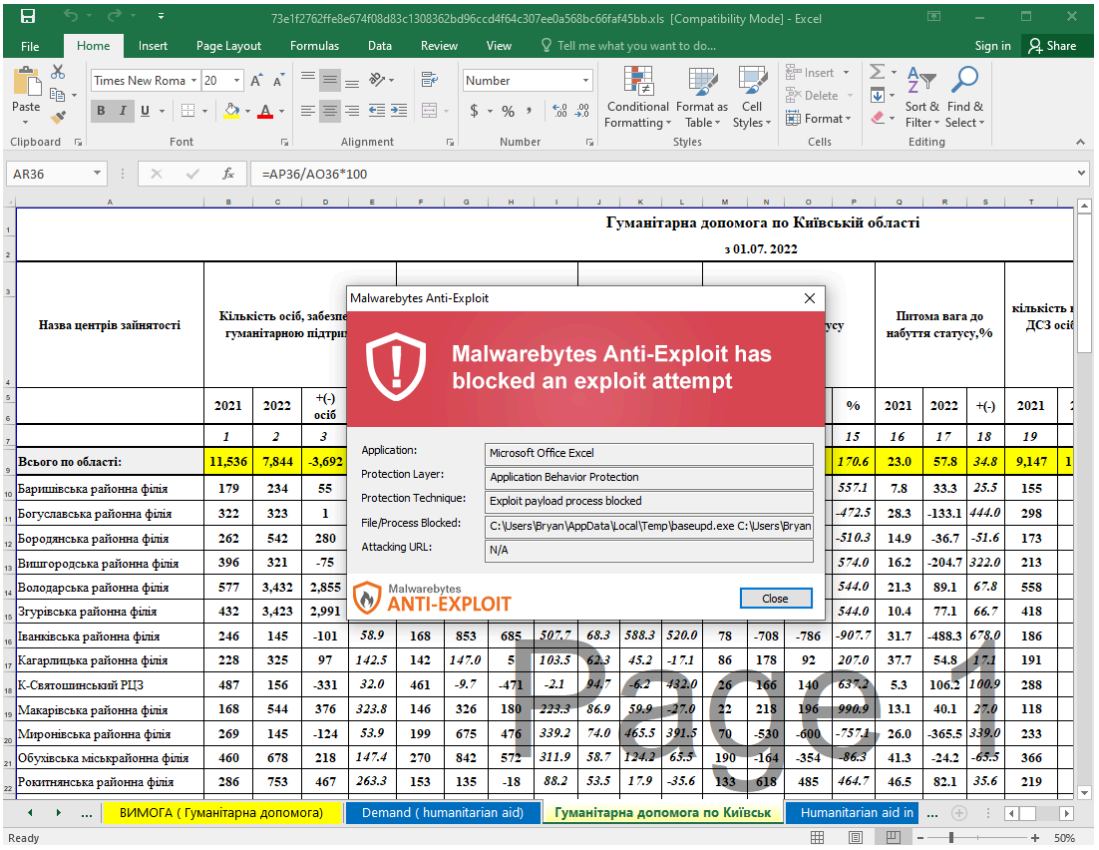
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

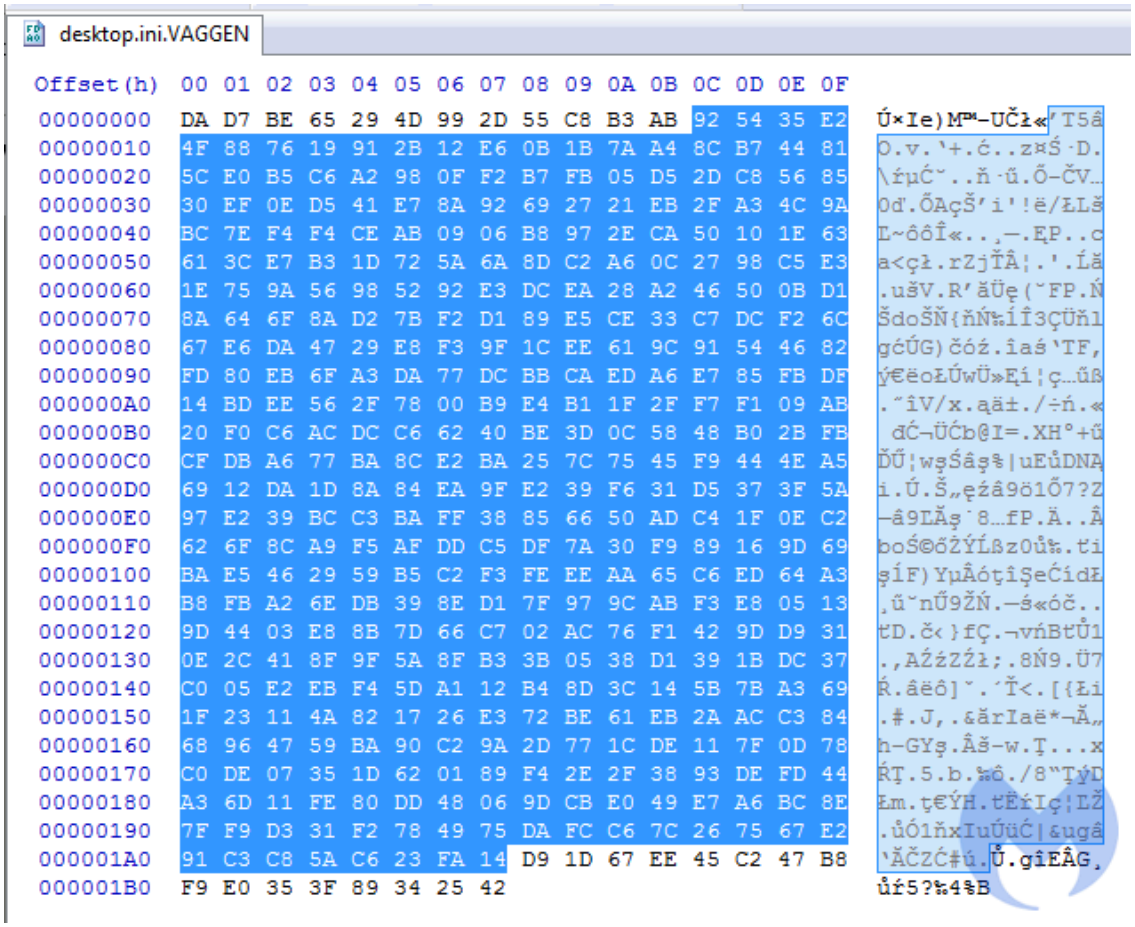
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

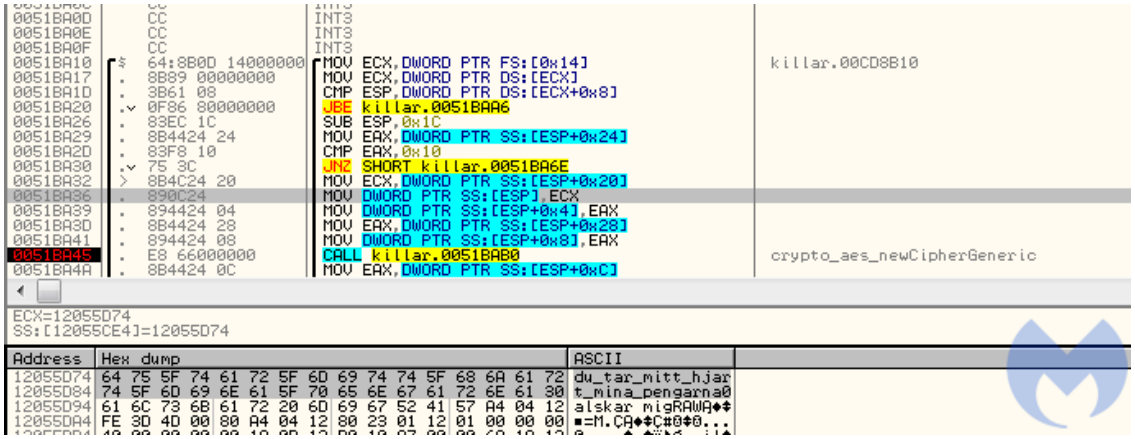


The content of the output file (with .VAGGEN extension) contains:

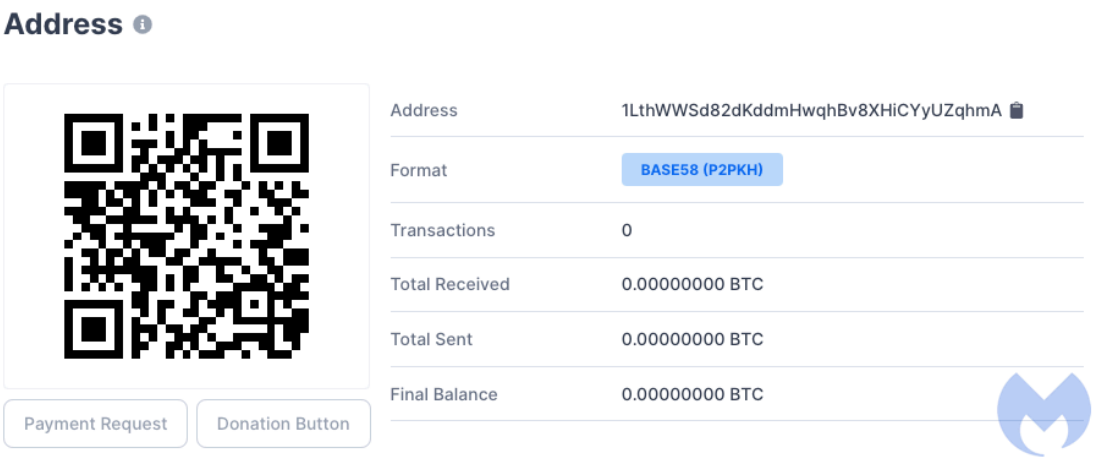
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

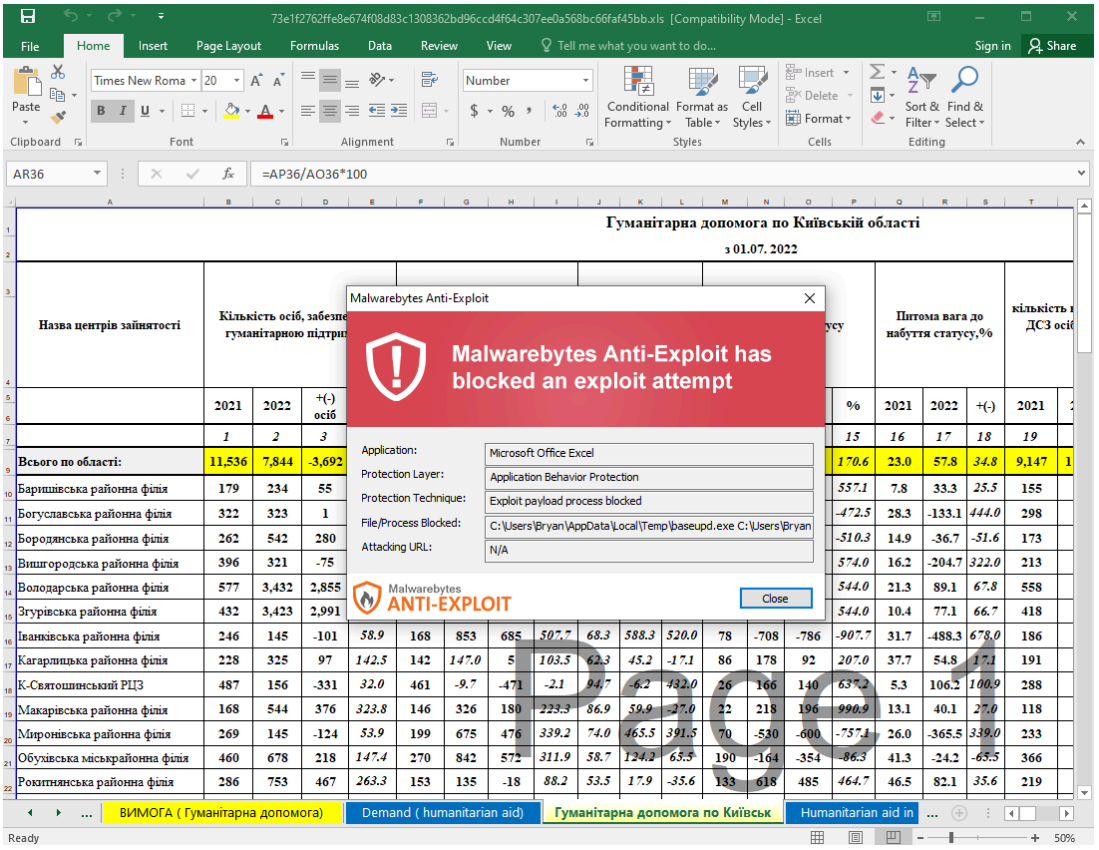
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

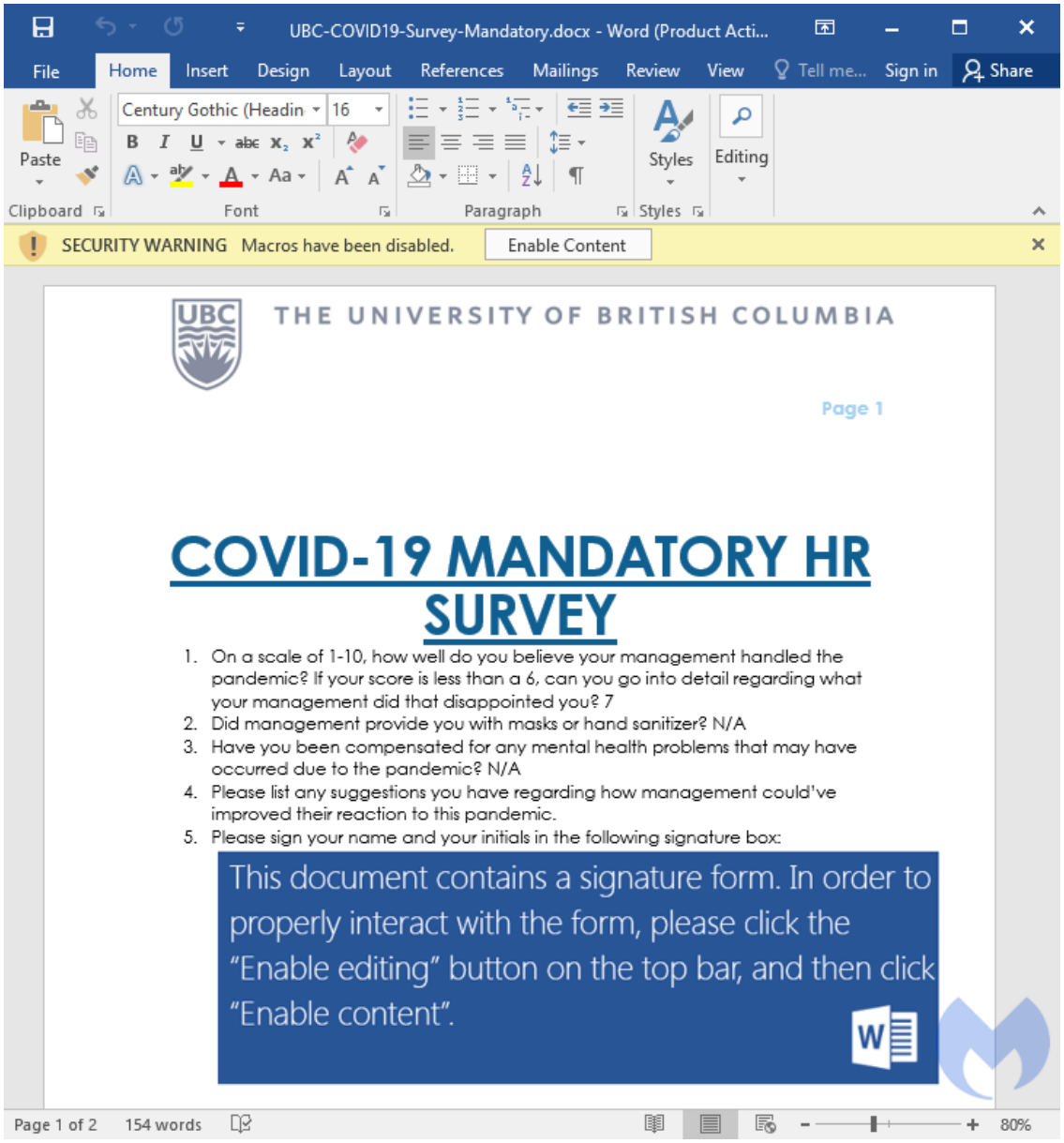
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

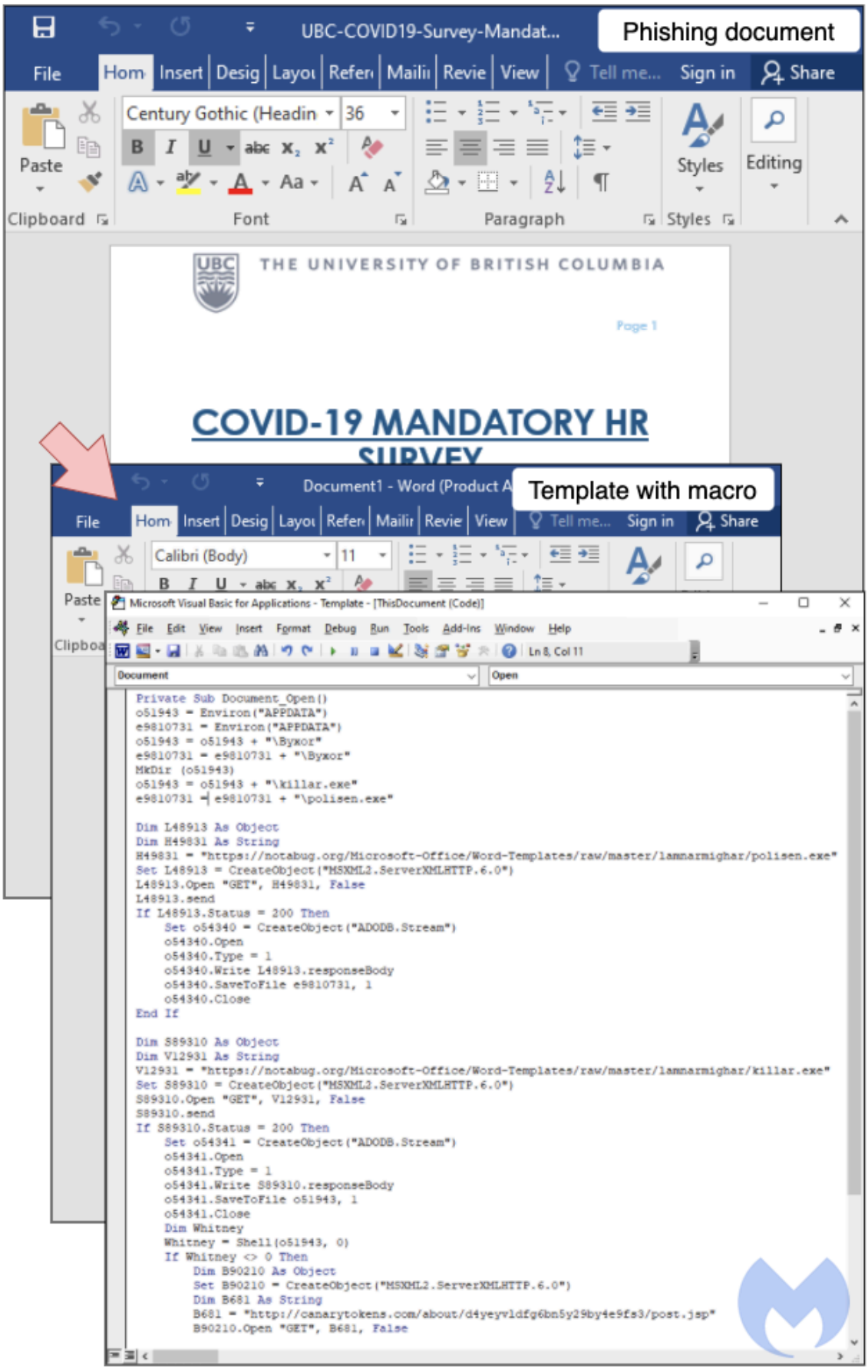
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

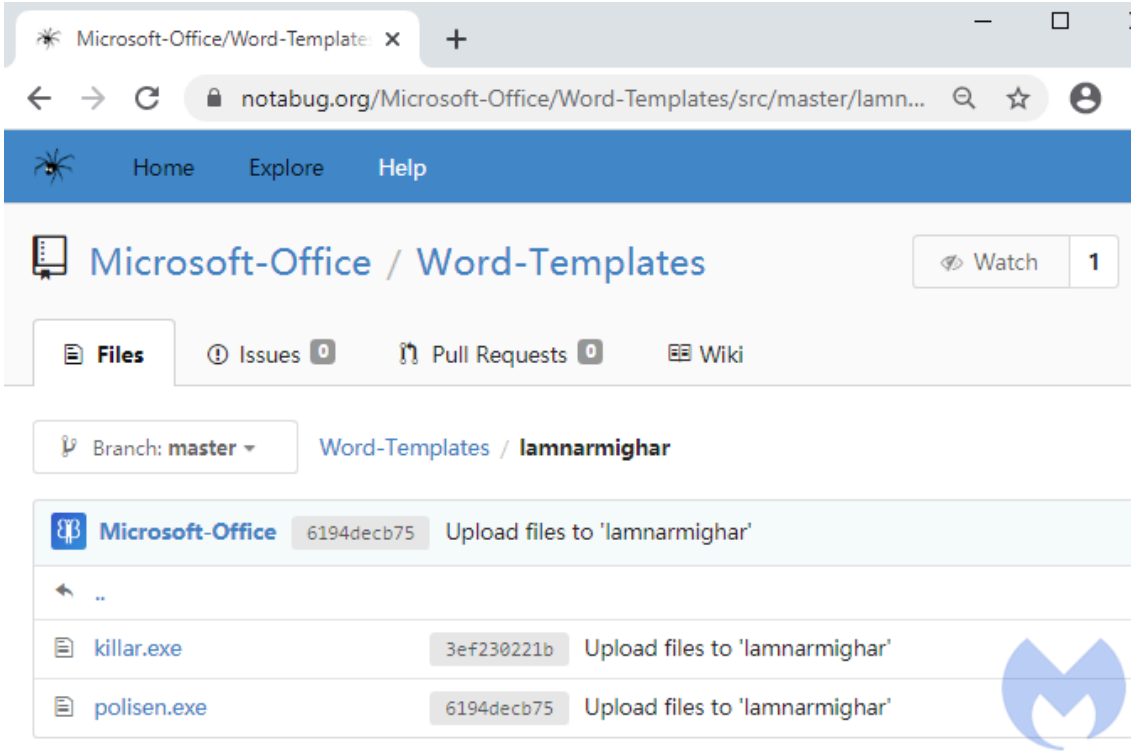
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



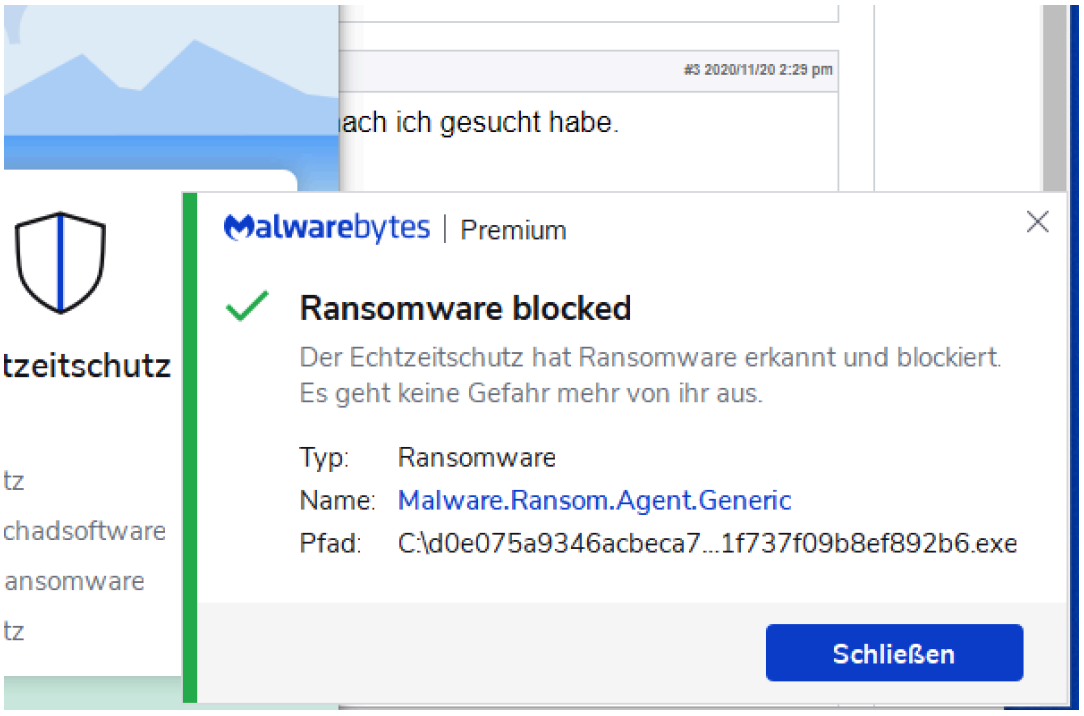
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

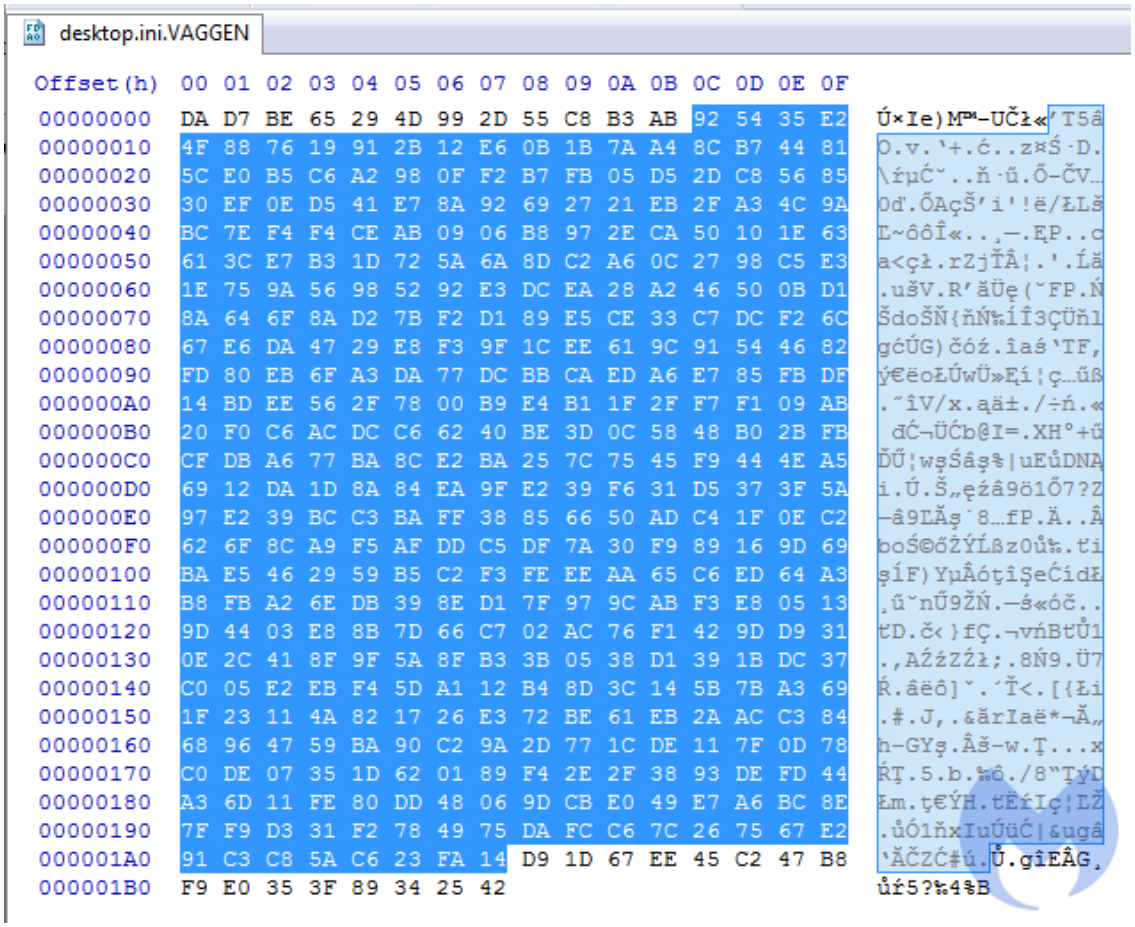
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



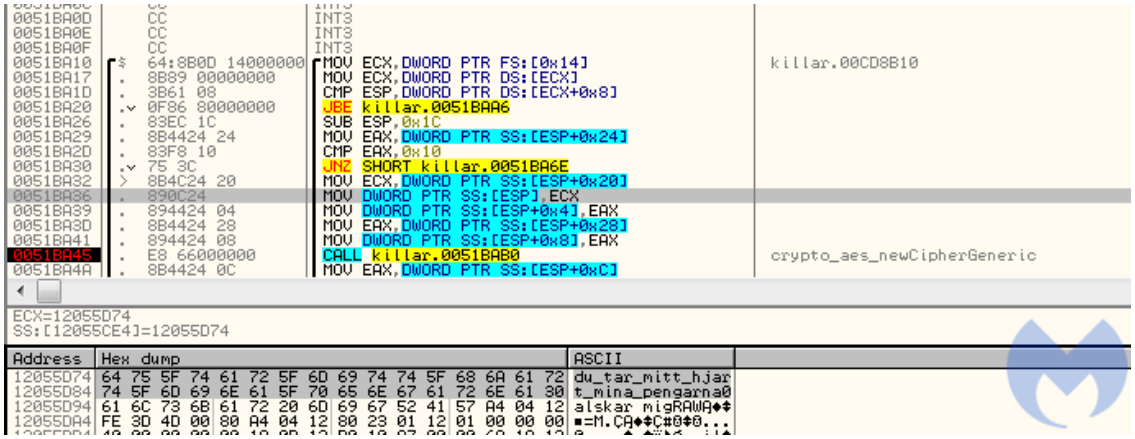
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

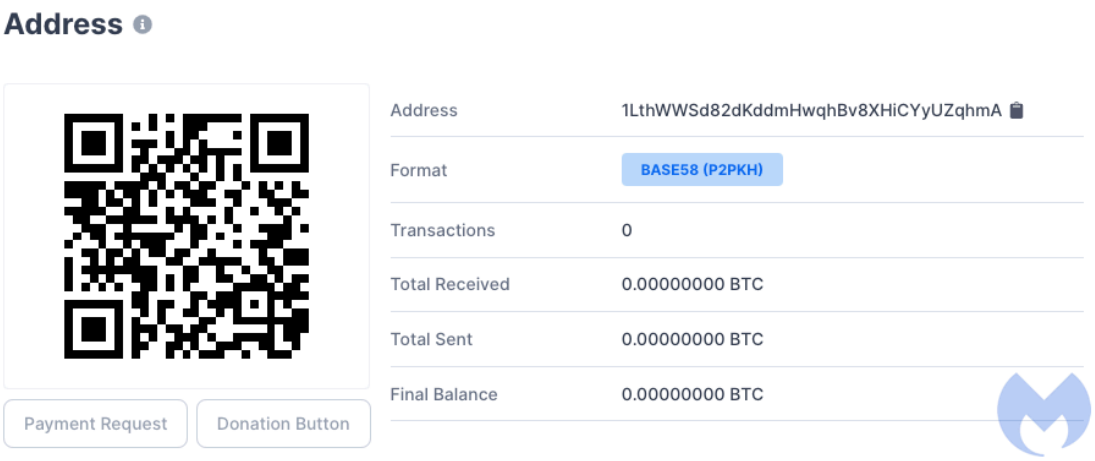
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

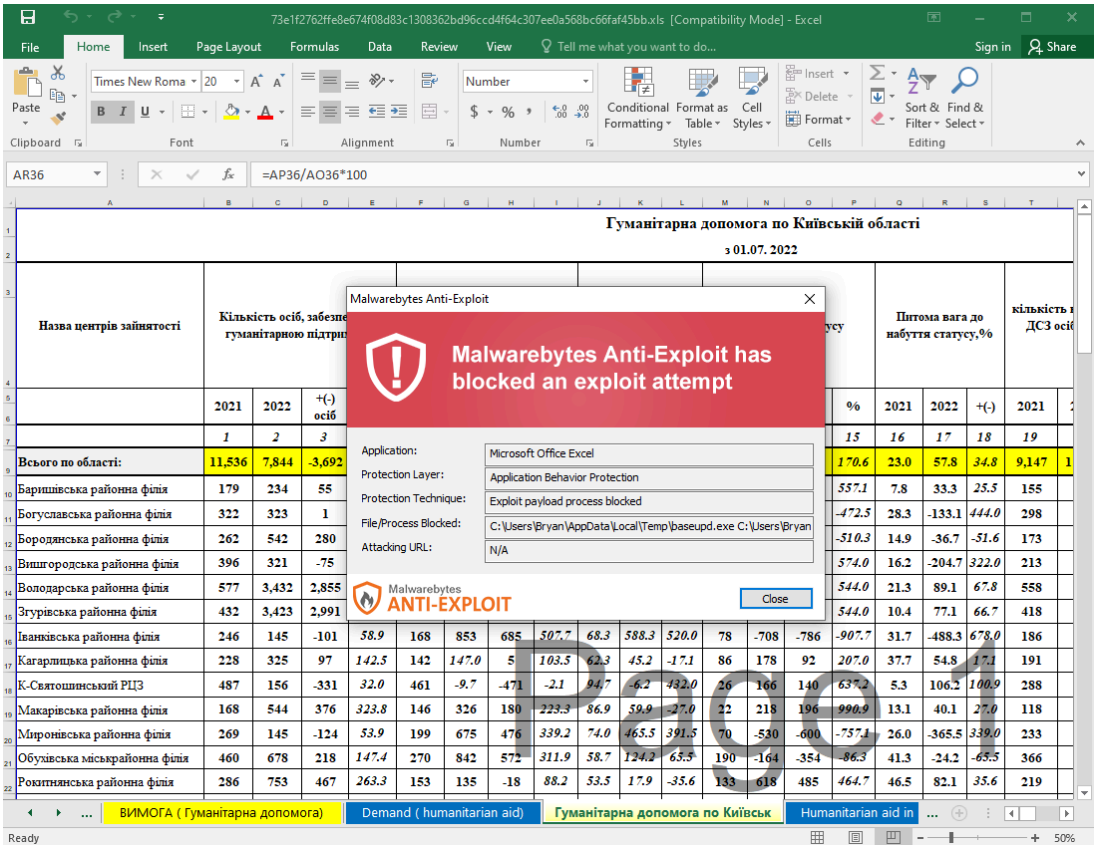
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

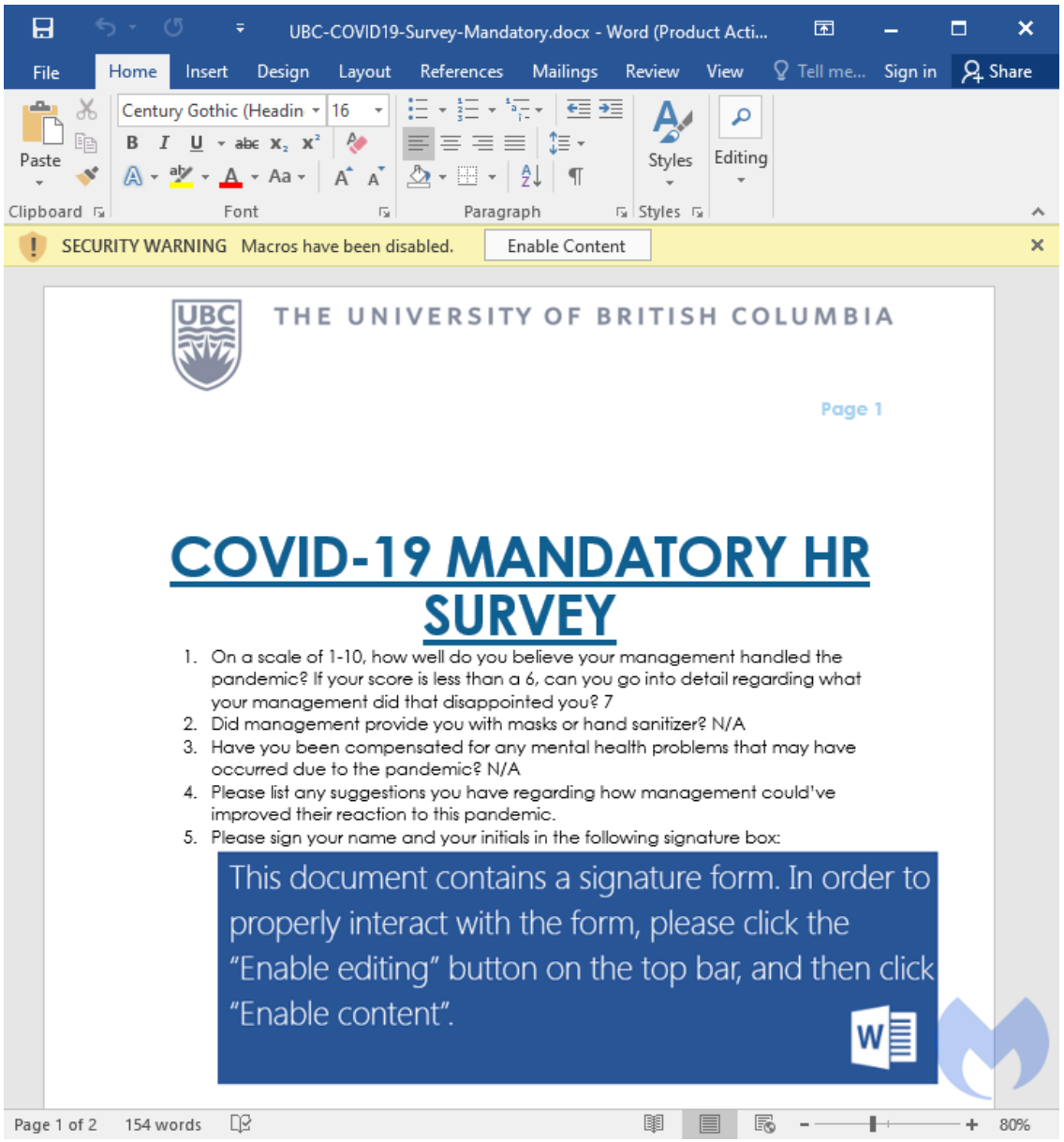
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

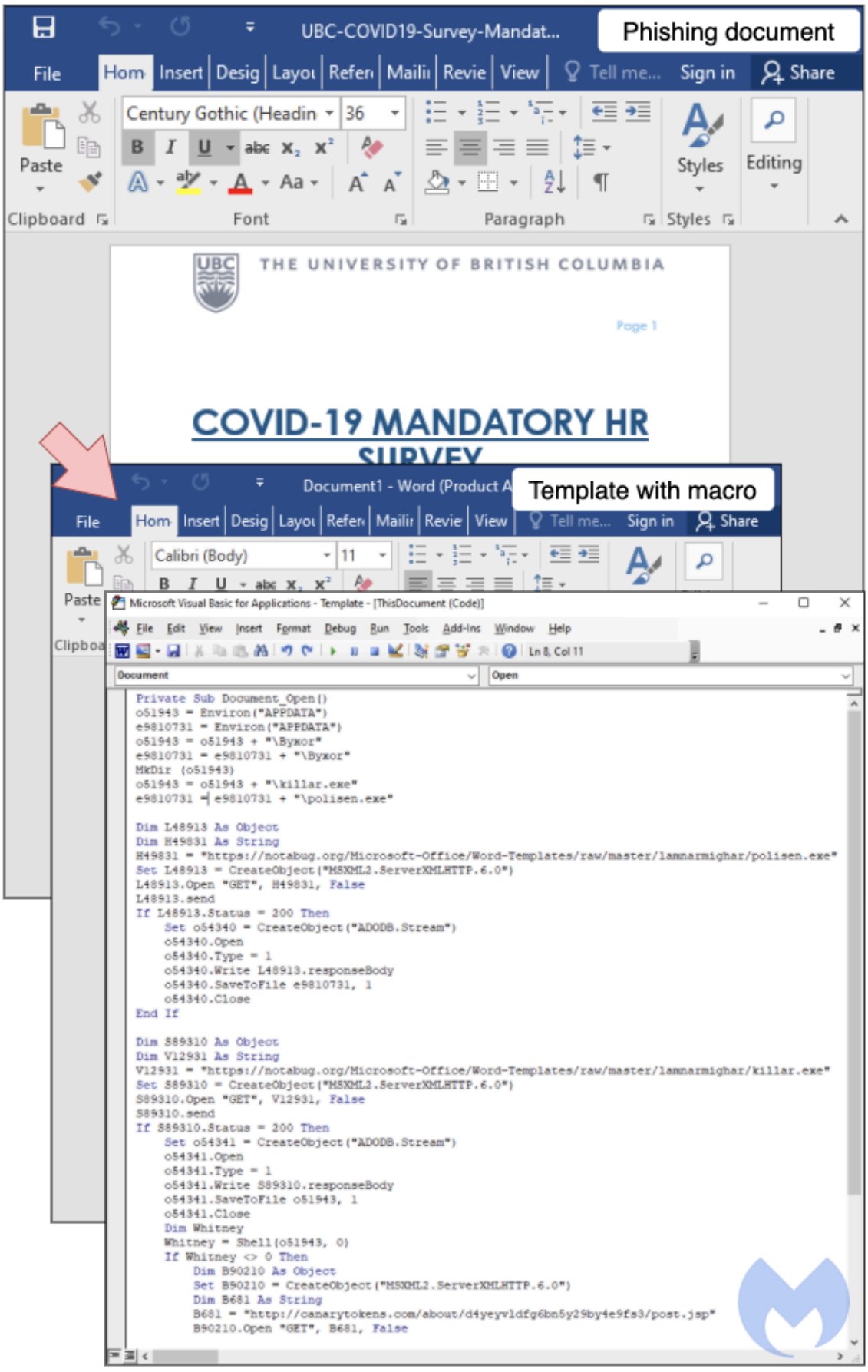
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

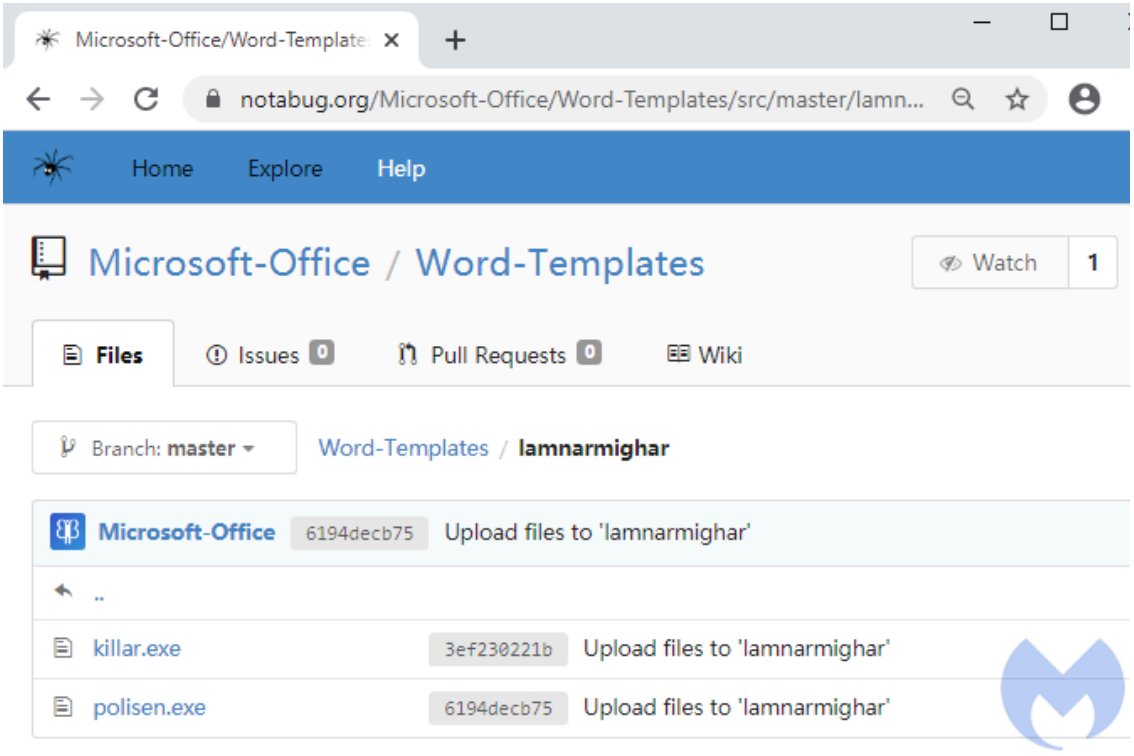
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



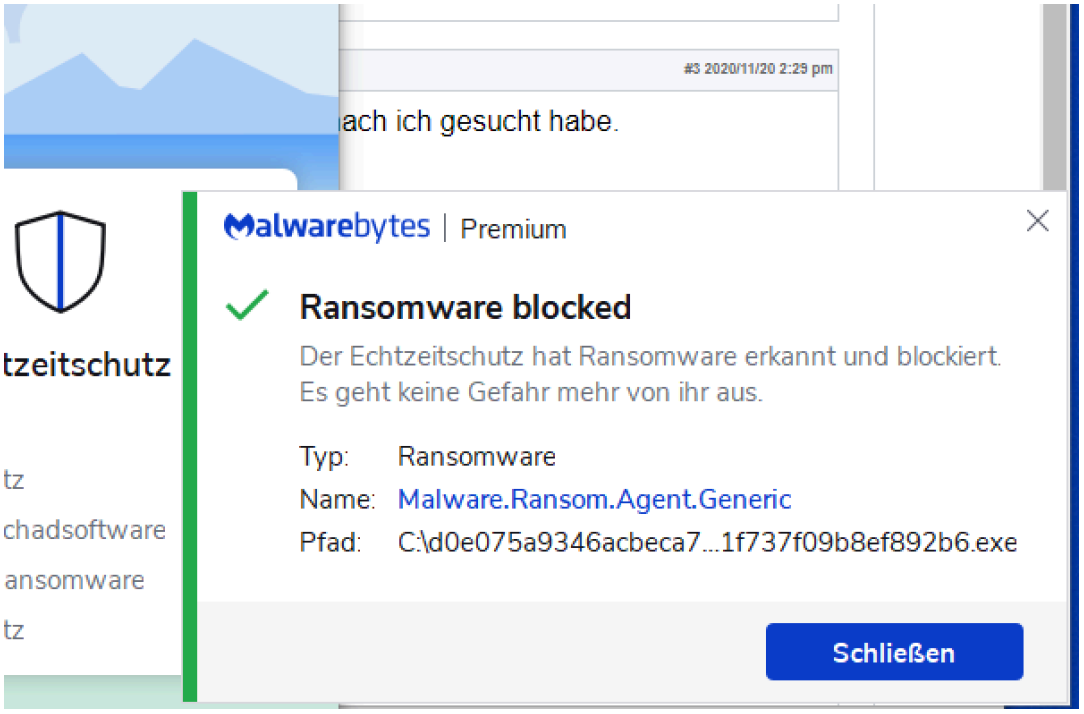
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

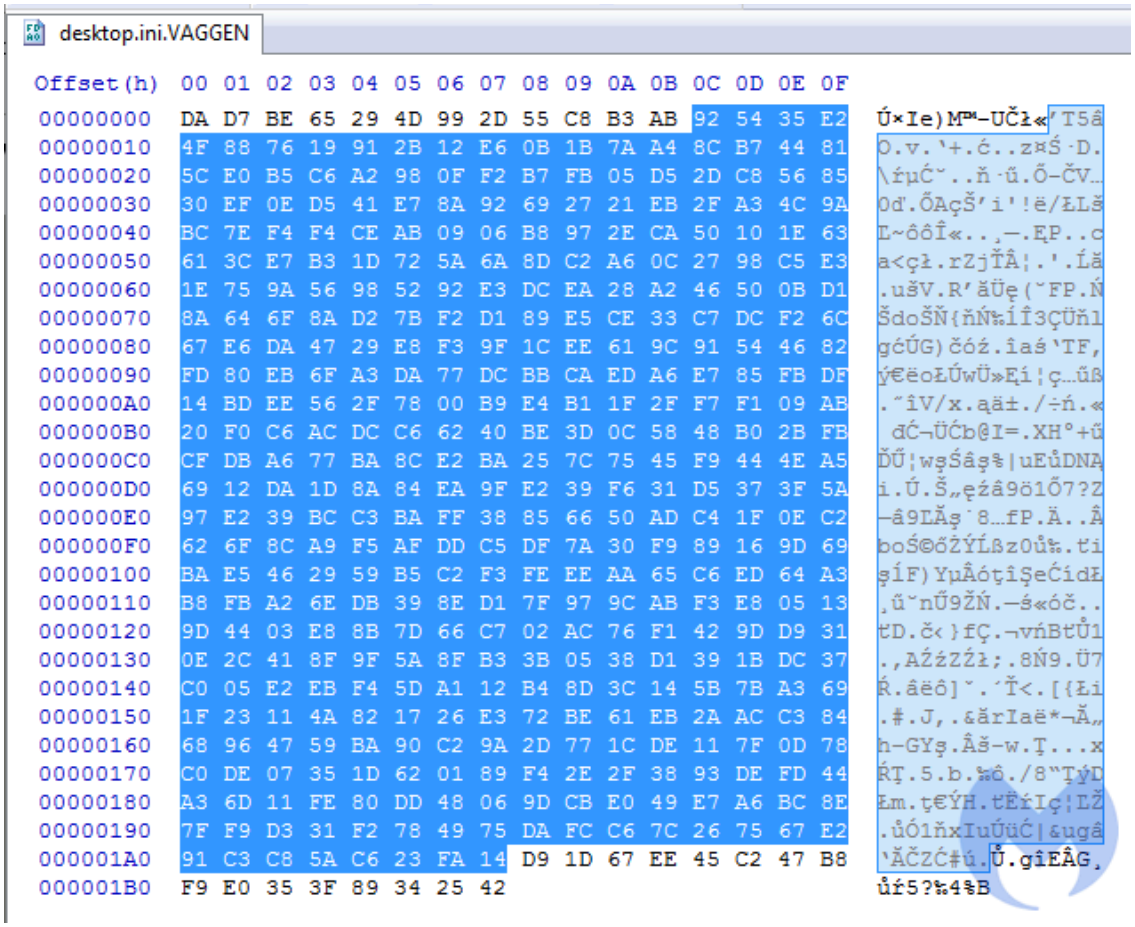
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

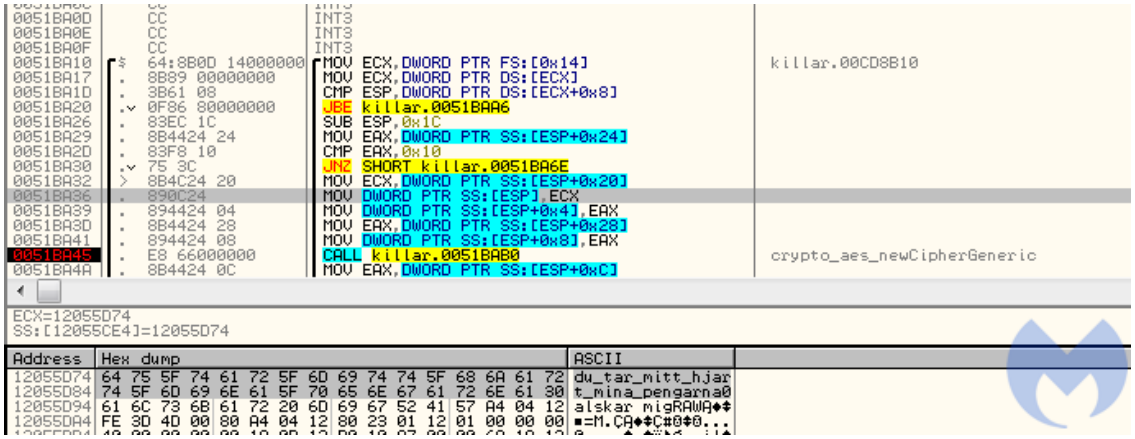


The content of the output file (with .VAGGEN extension) contains:

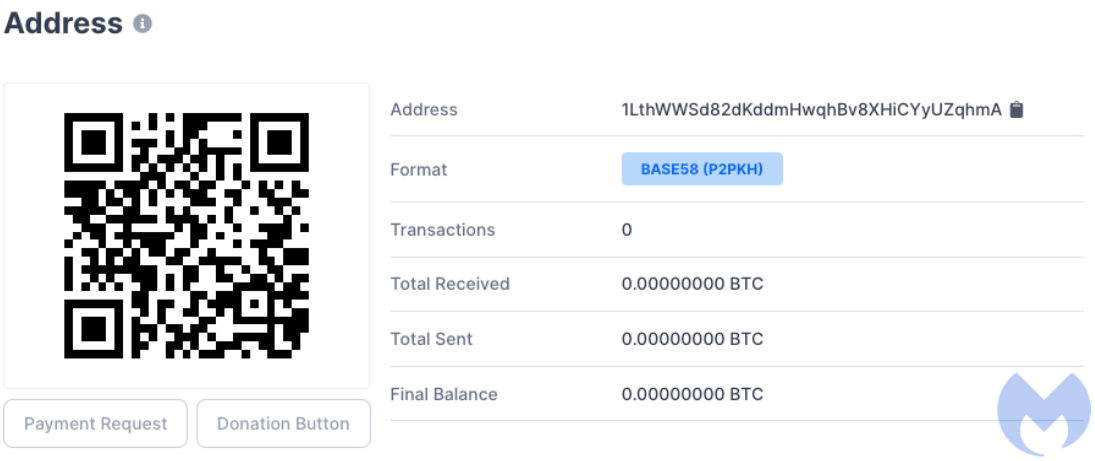
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

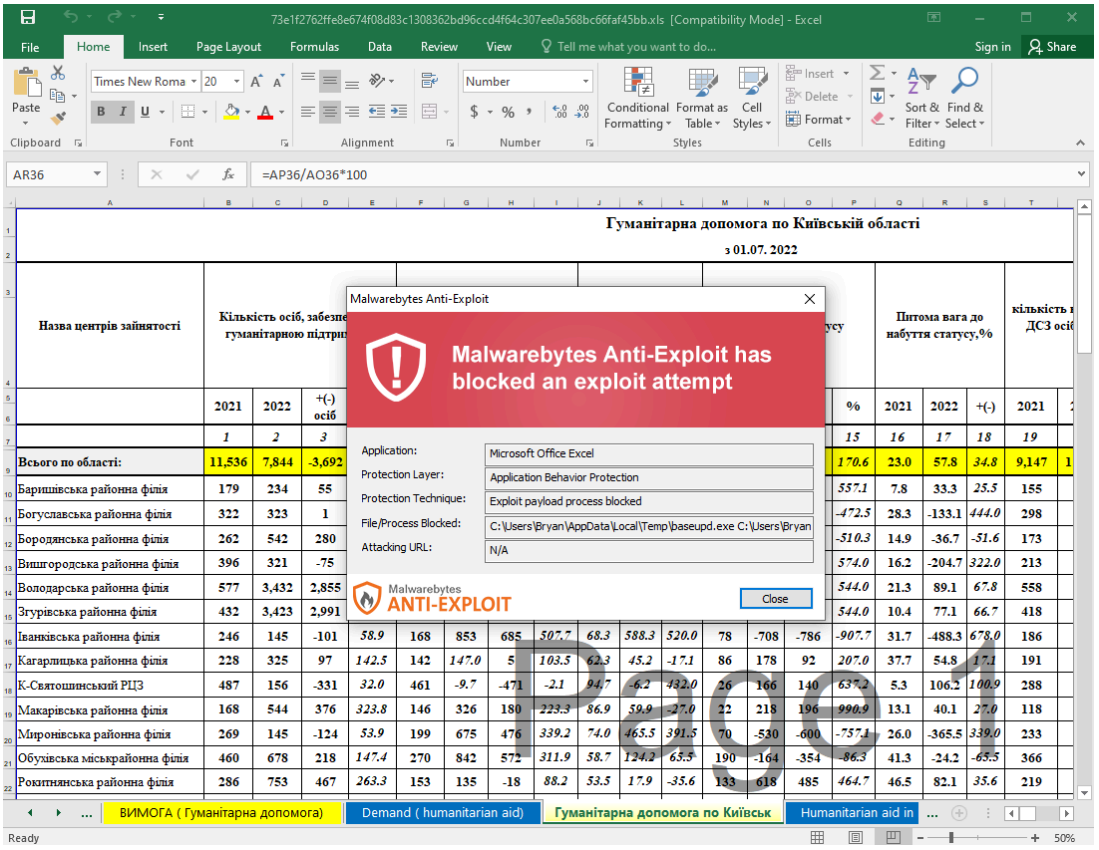
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

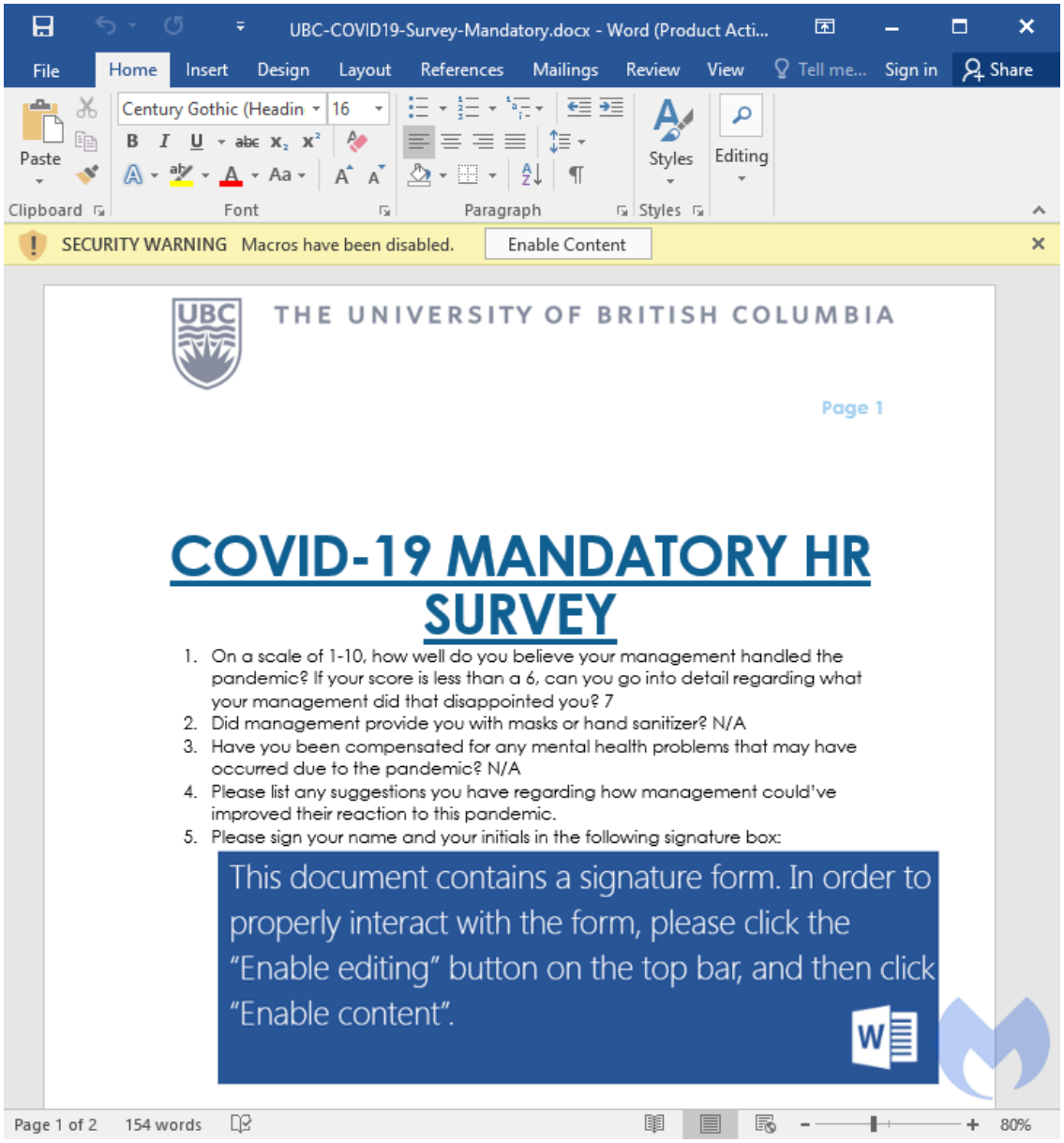
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

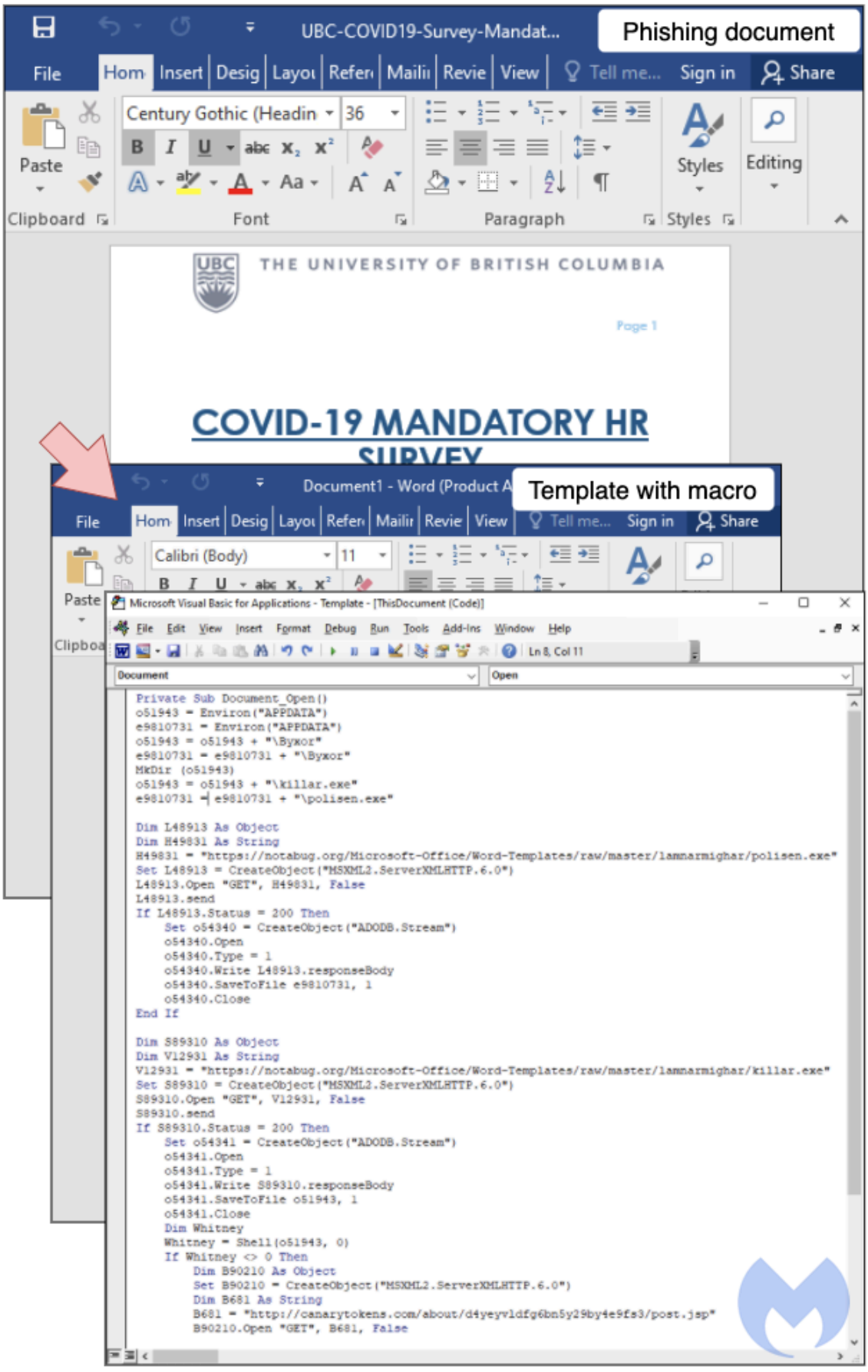
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

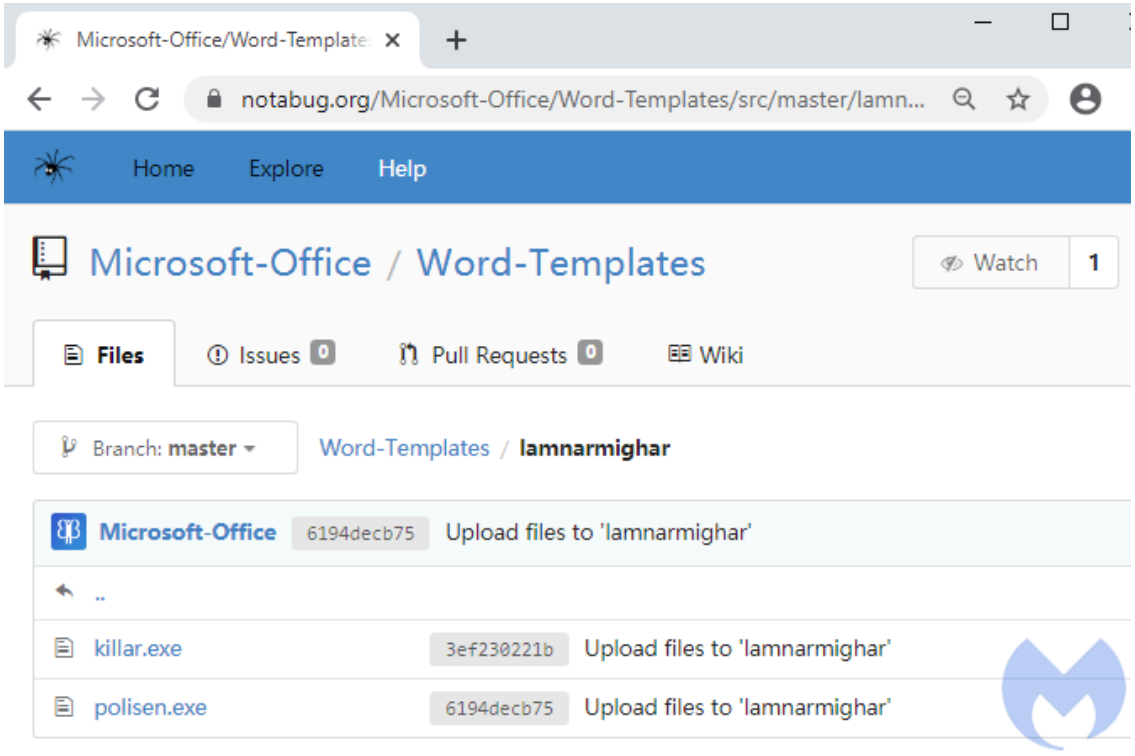
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



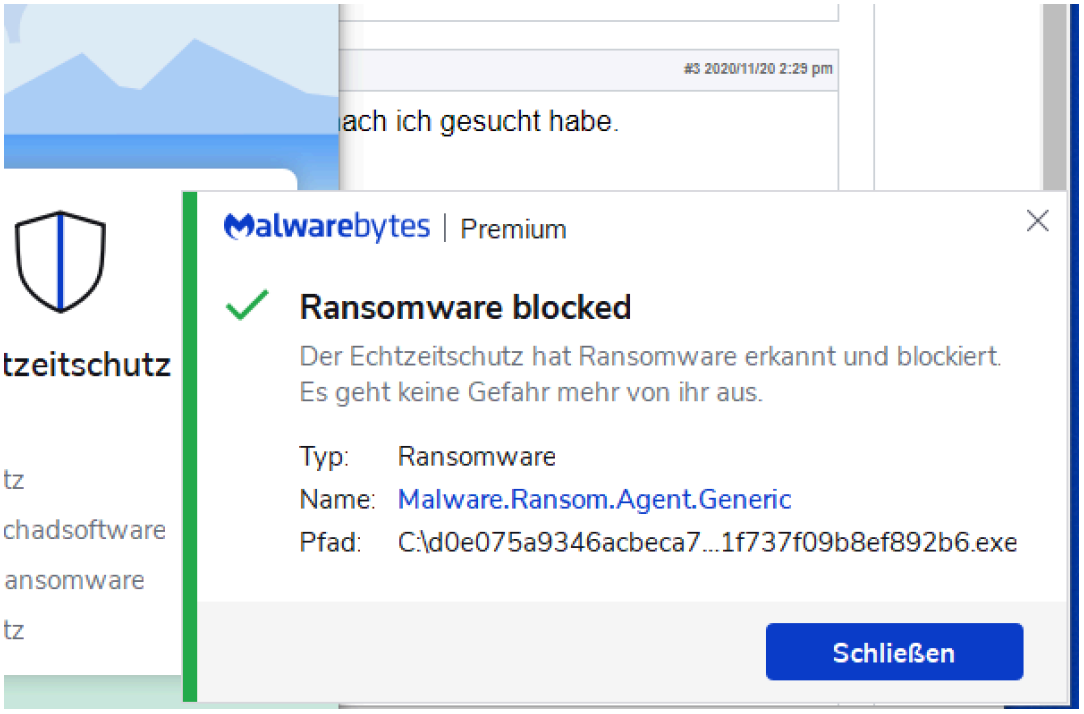
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

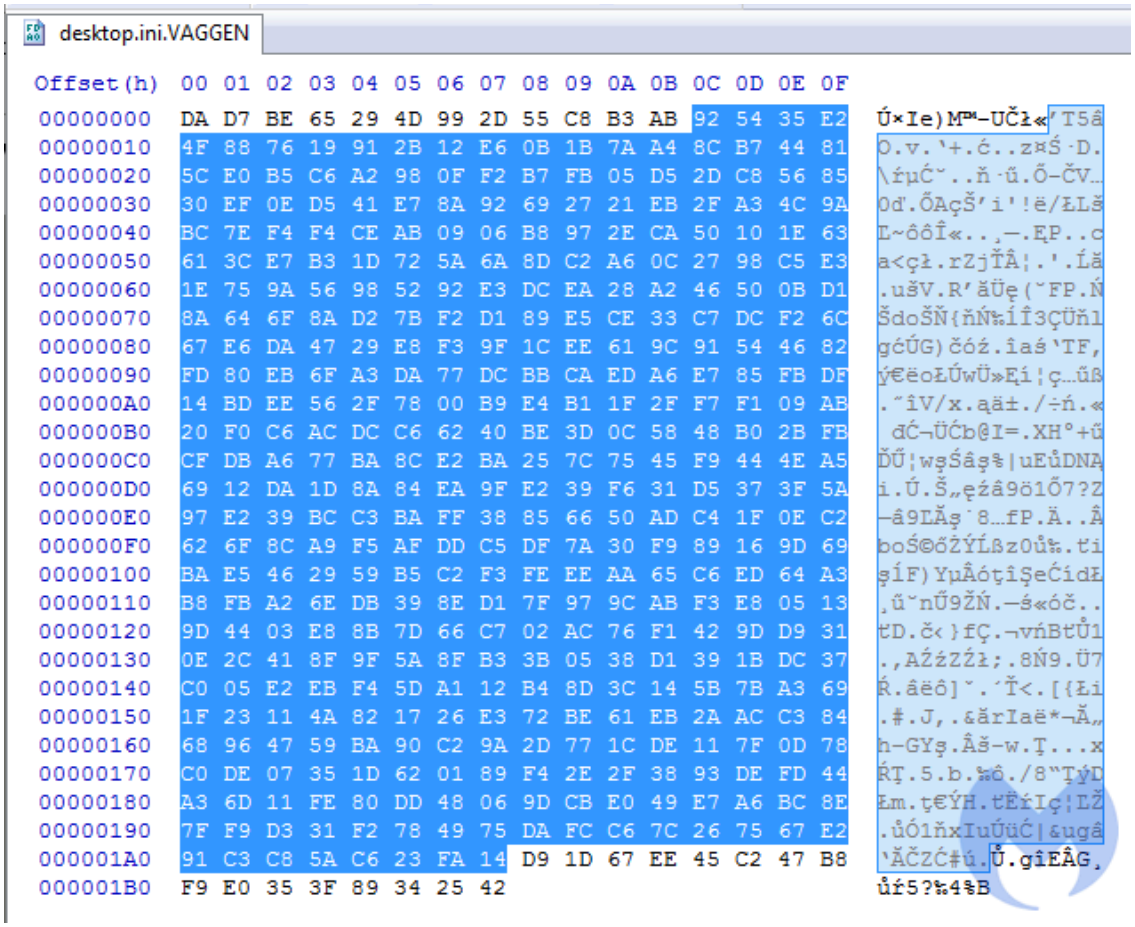
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

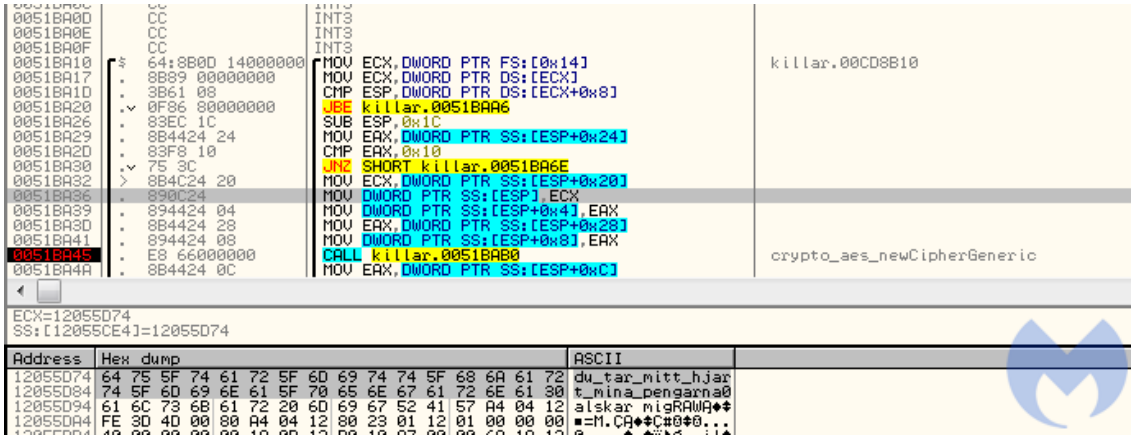


The content of the output file (with .VAGGEN extension) contains:

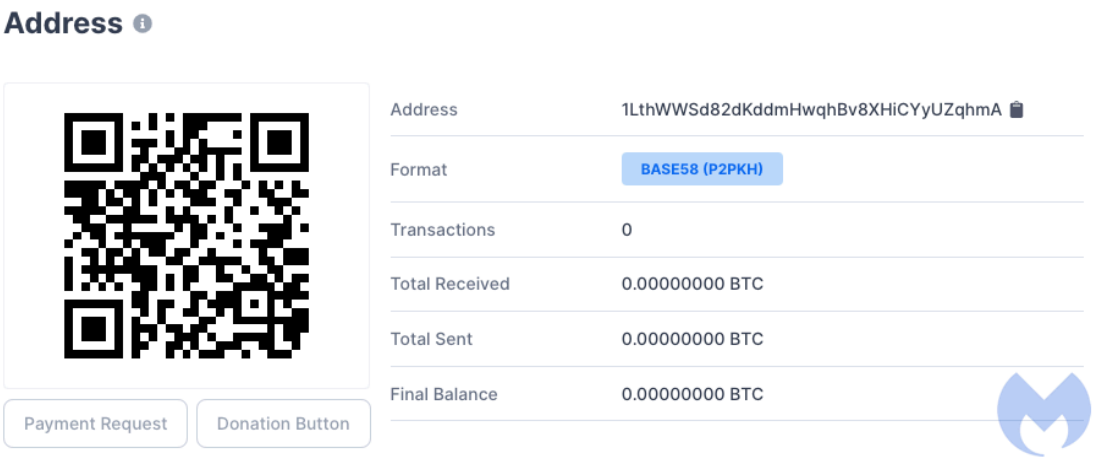
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

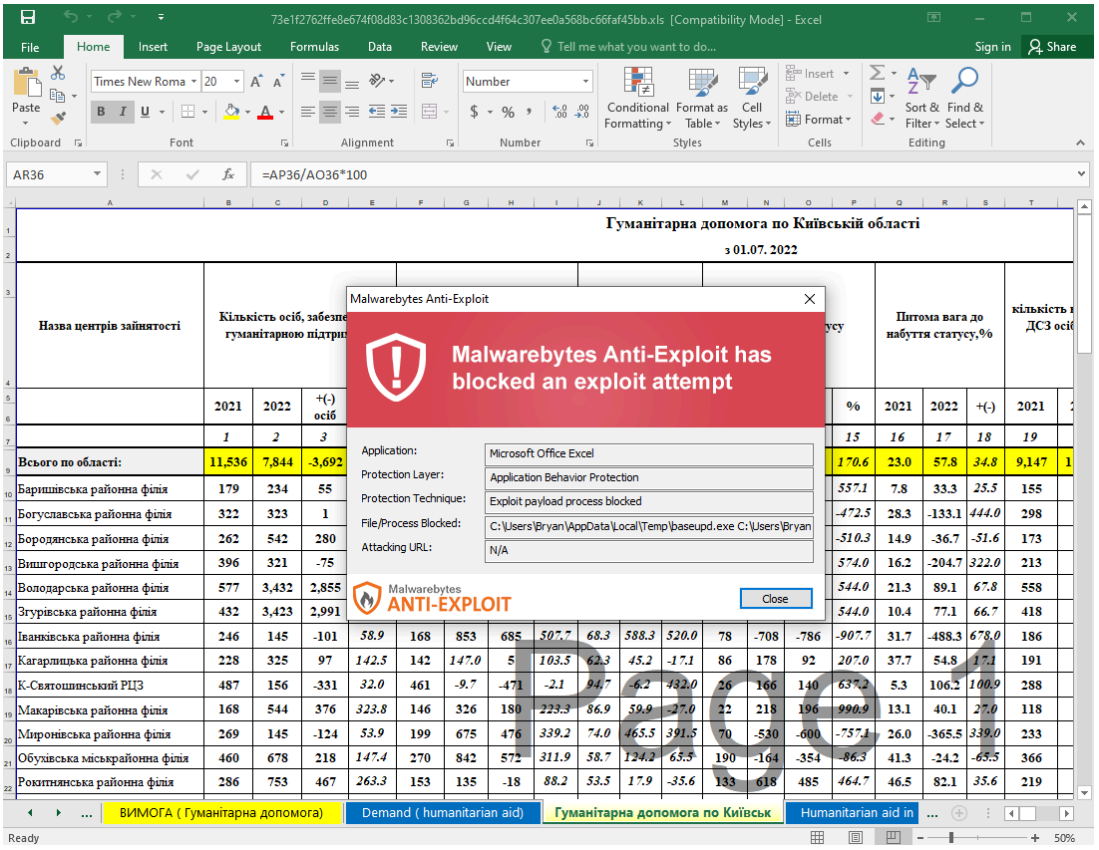
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

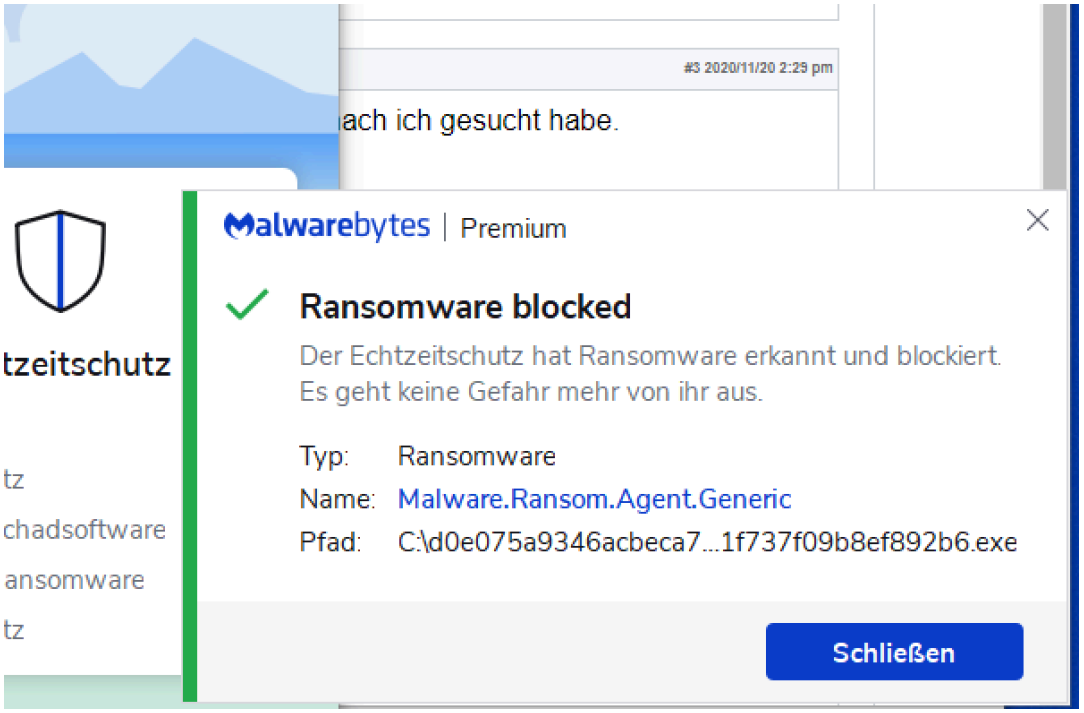
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

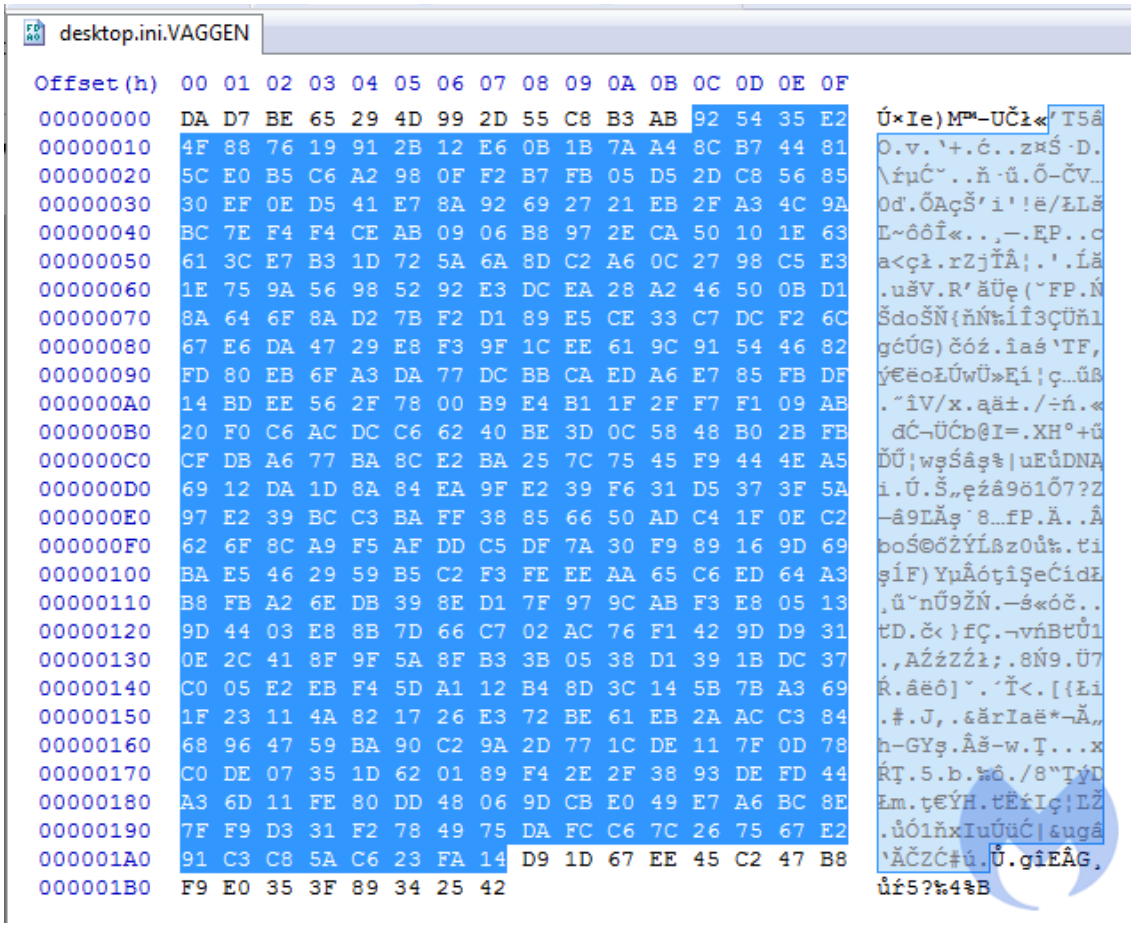
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

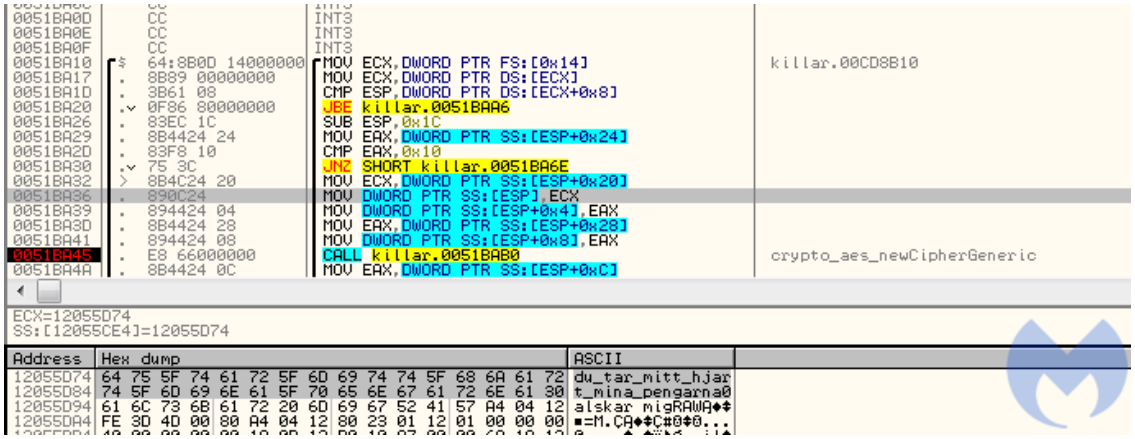


The content of the output file (with .VAGGEN extension) contains:

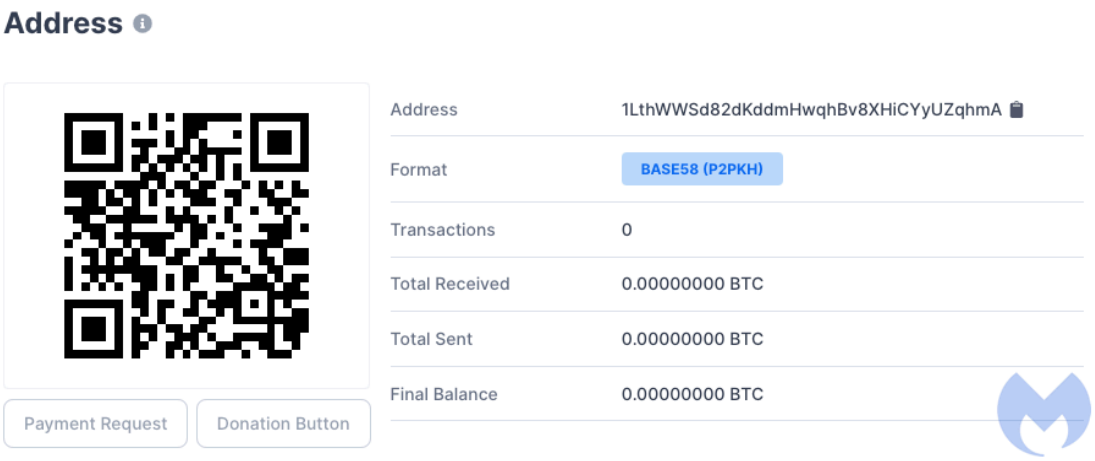
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

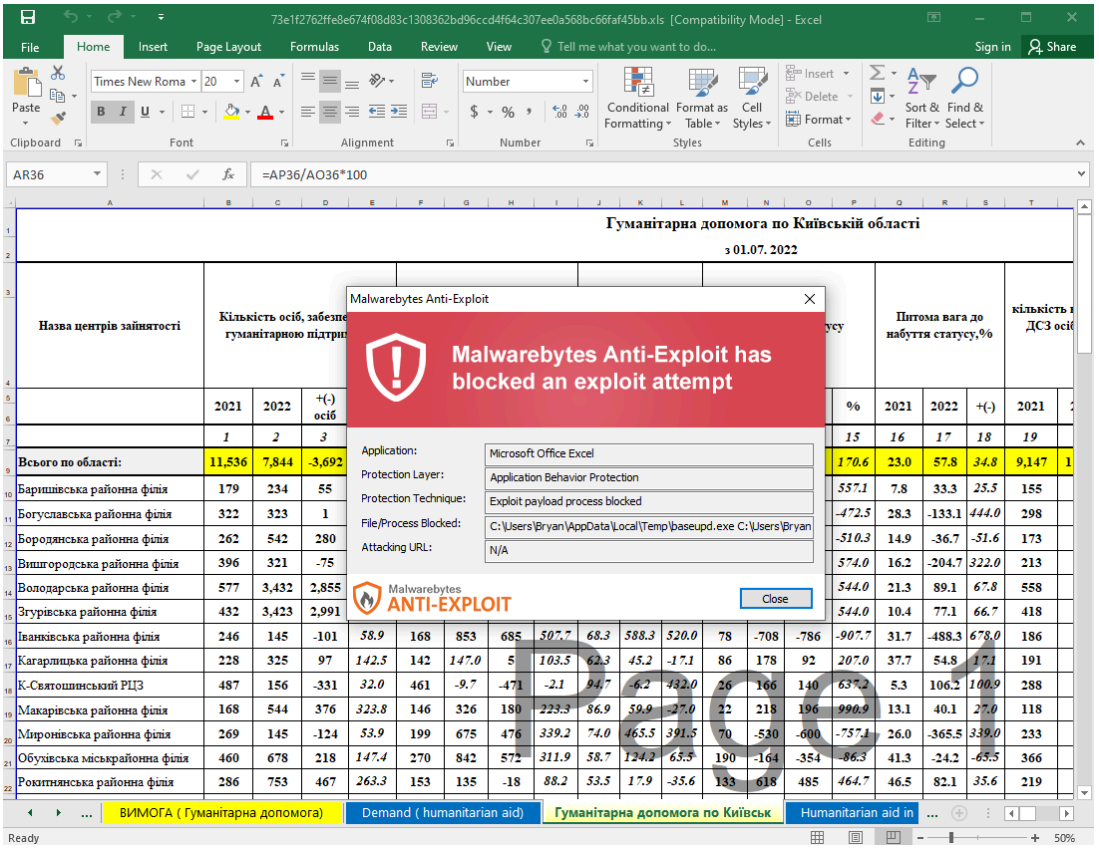
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

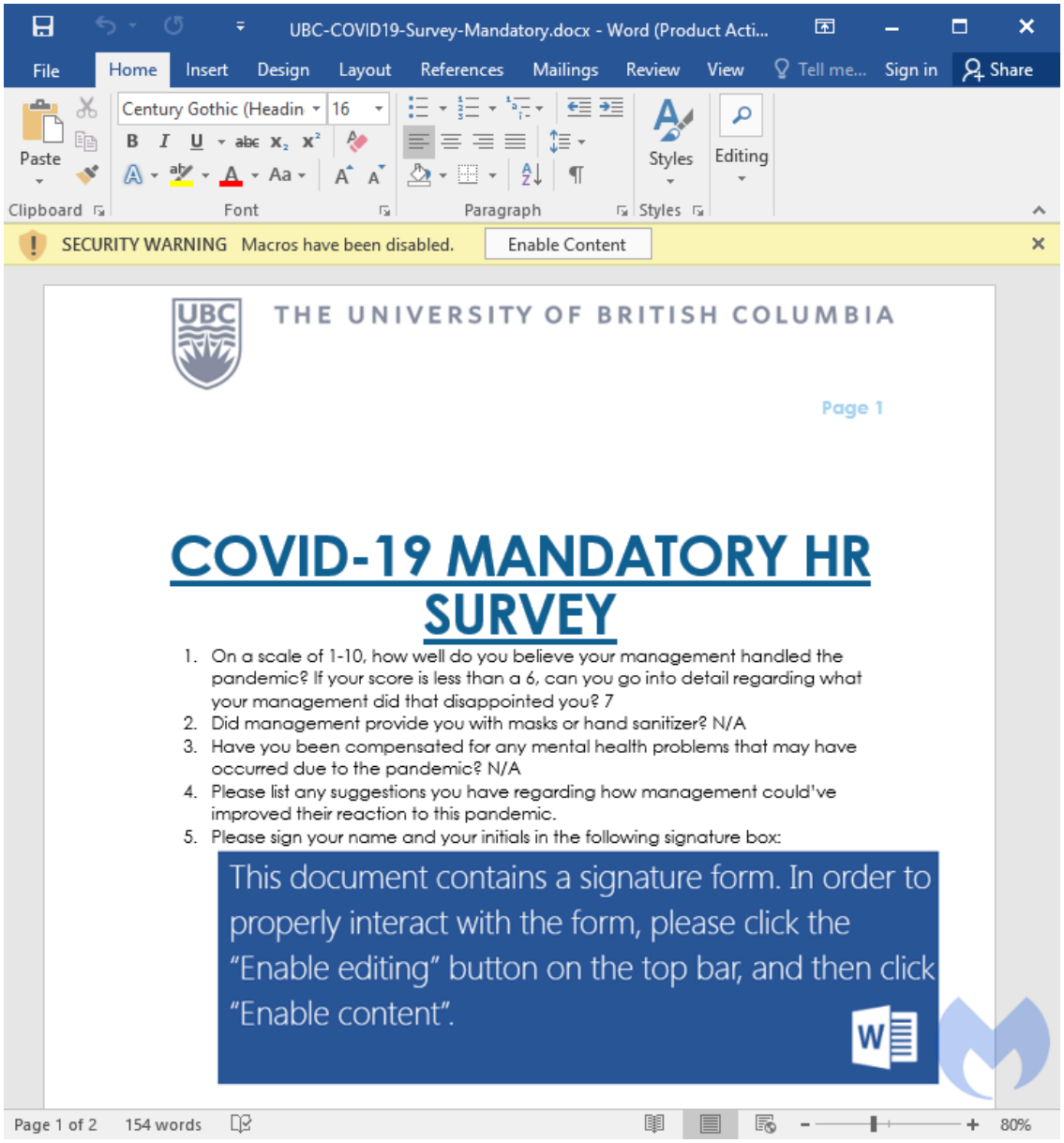
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

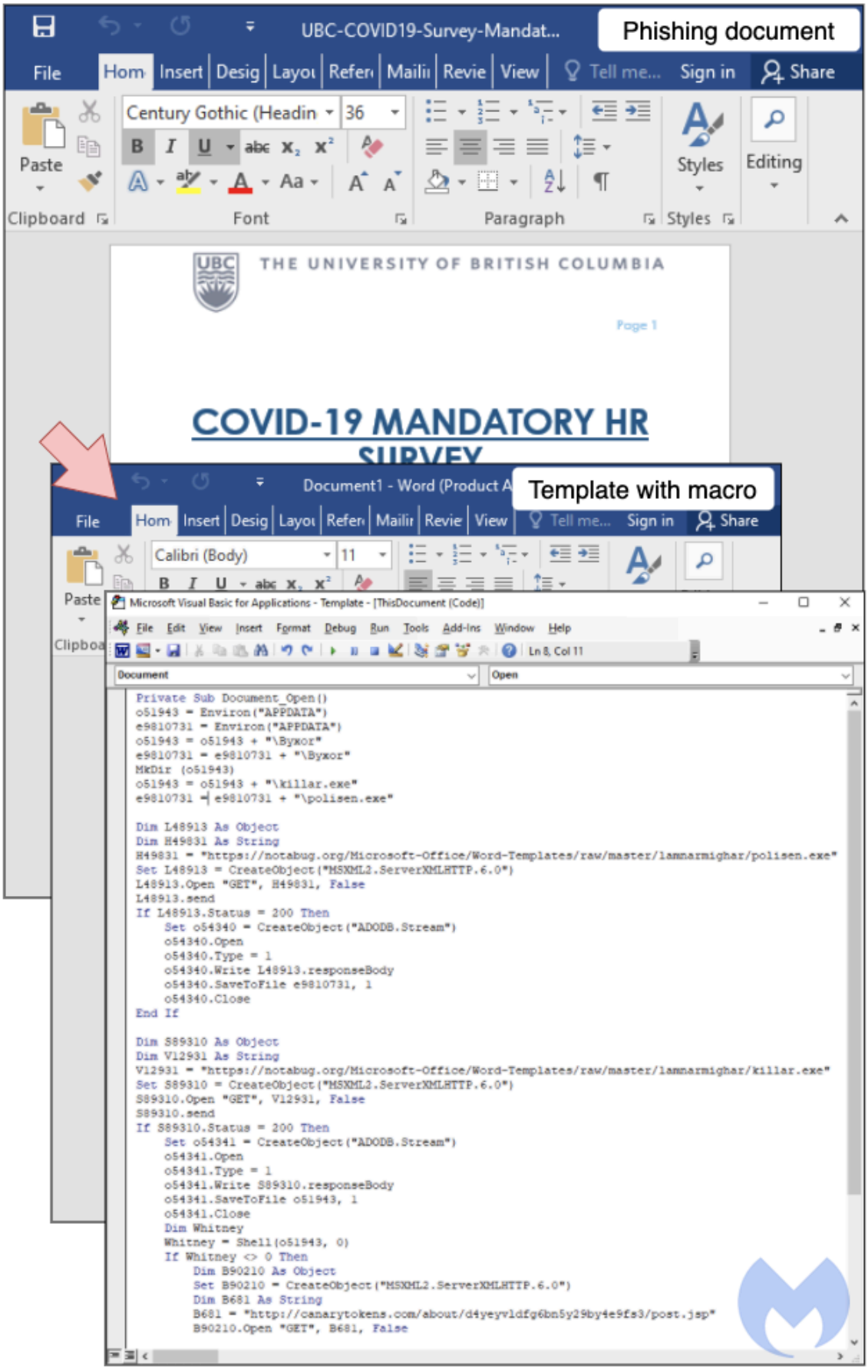
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

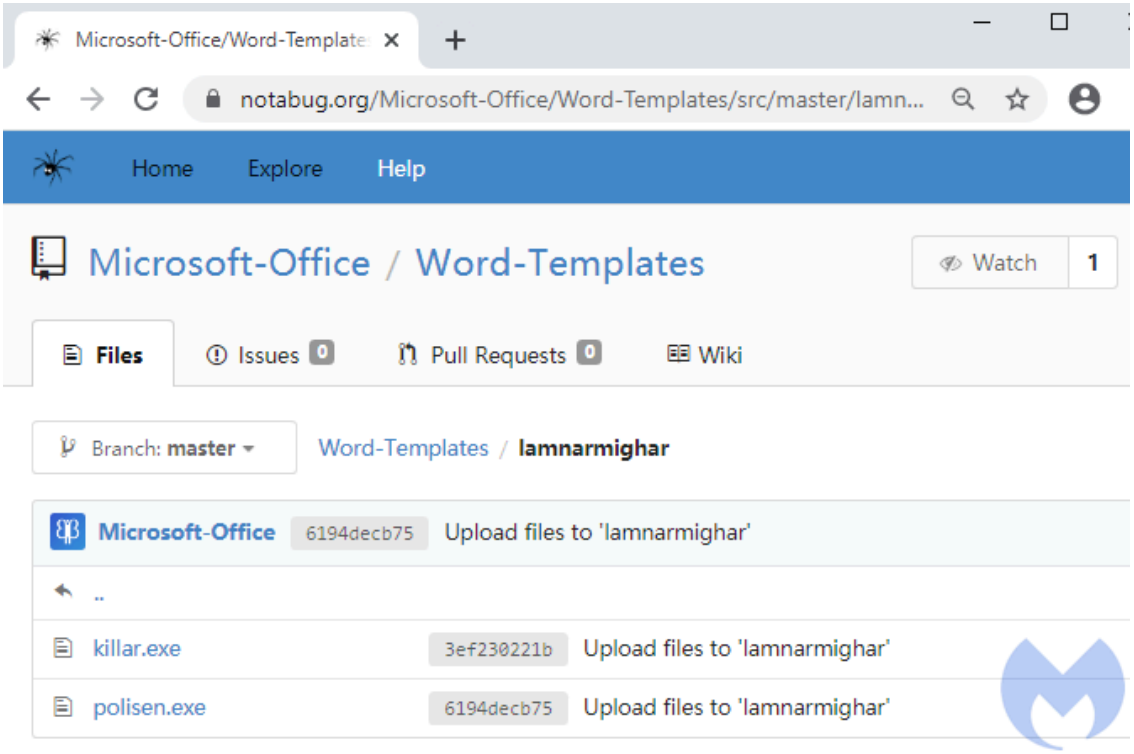
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



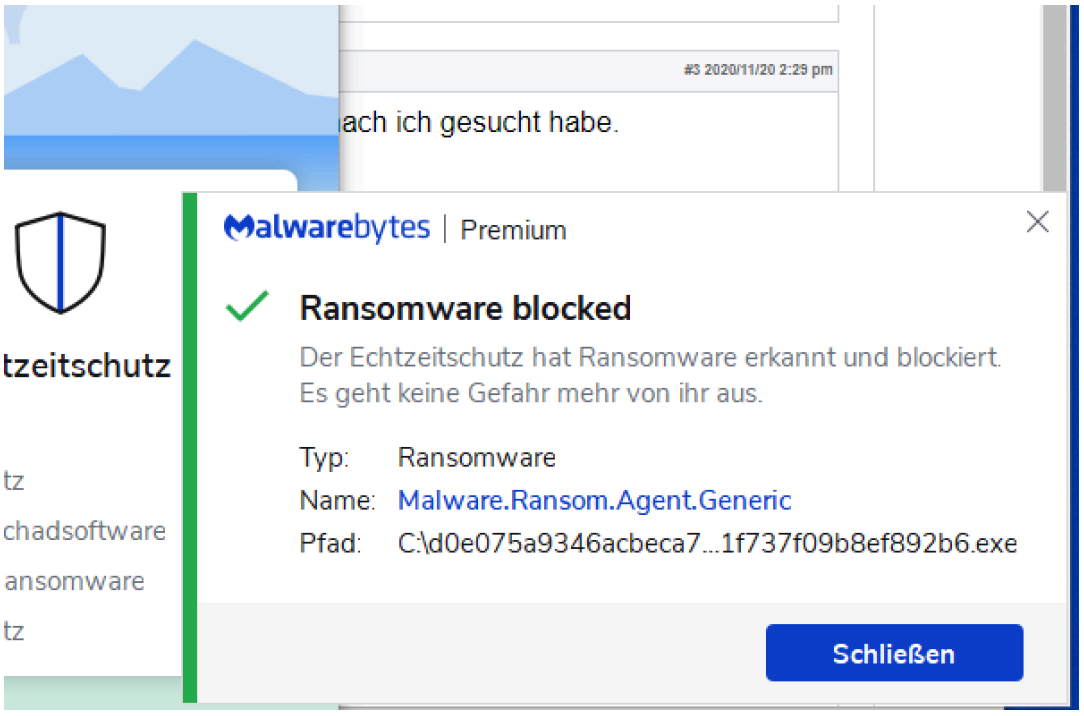
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

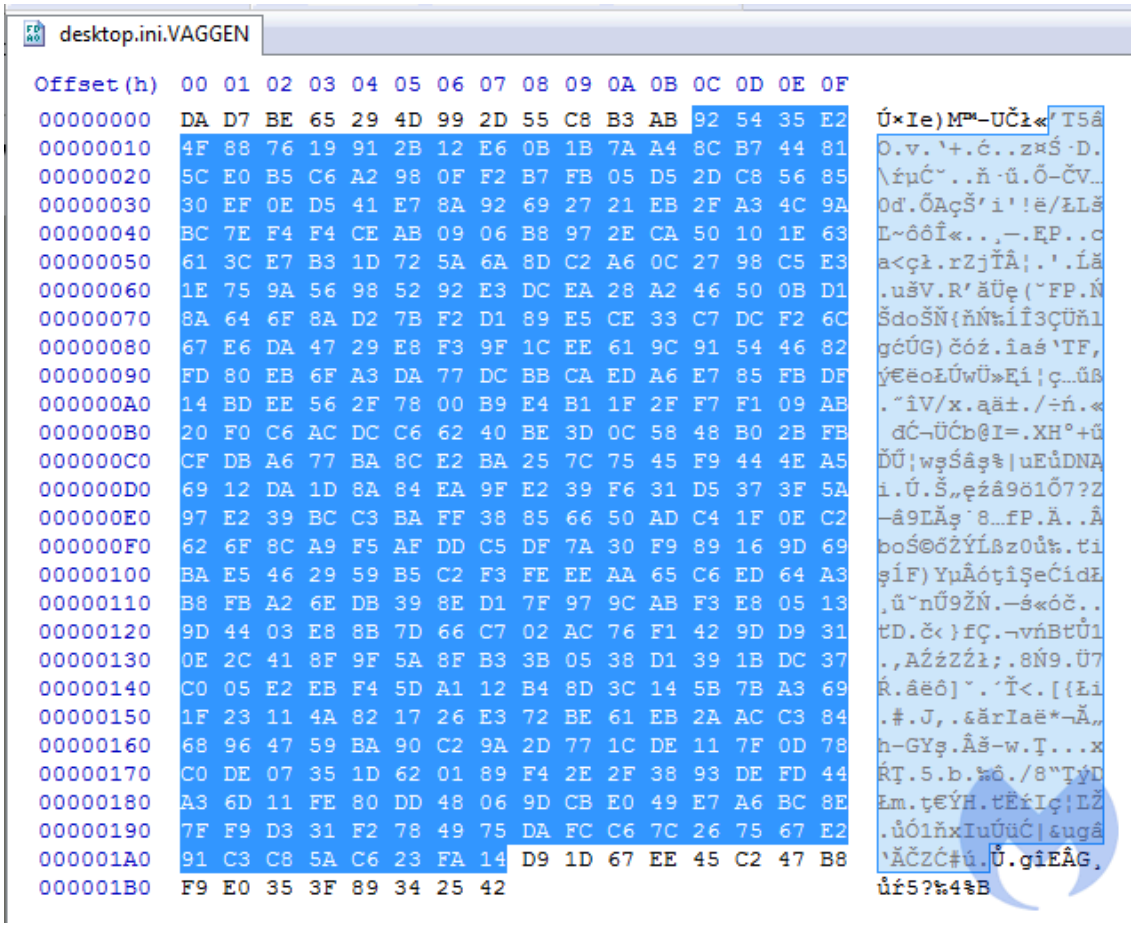
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

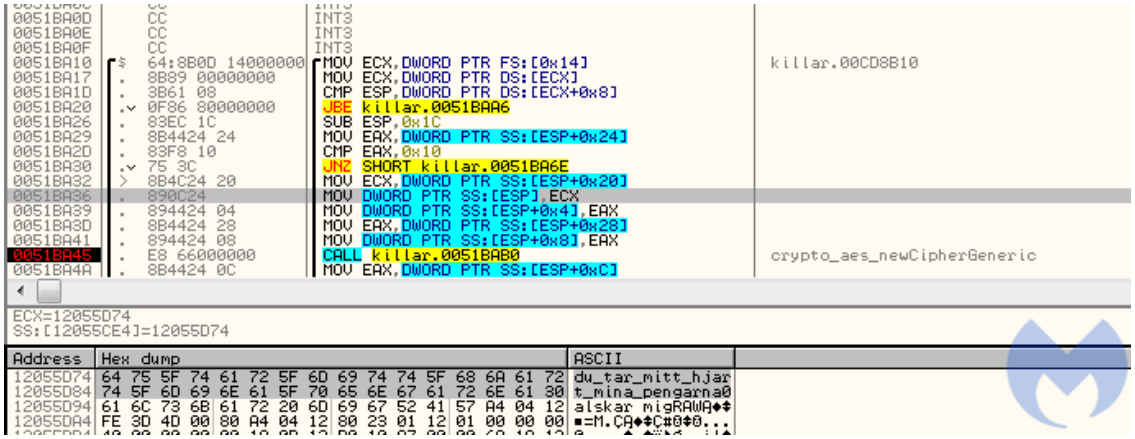


The content of the output file (with .VAGGEN extension) contains:

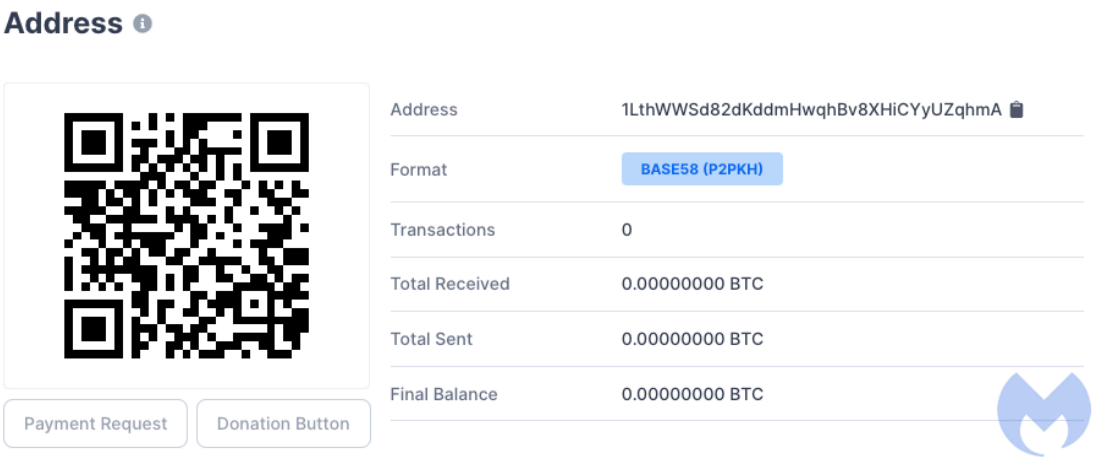
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

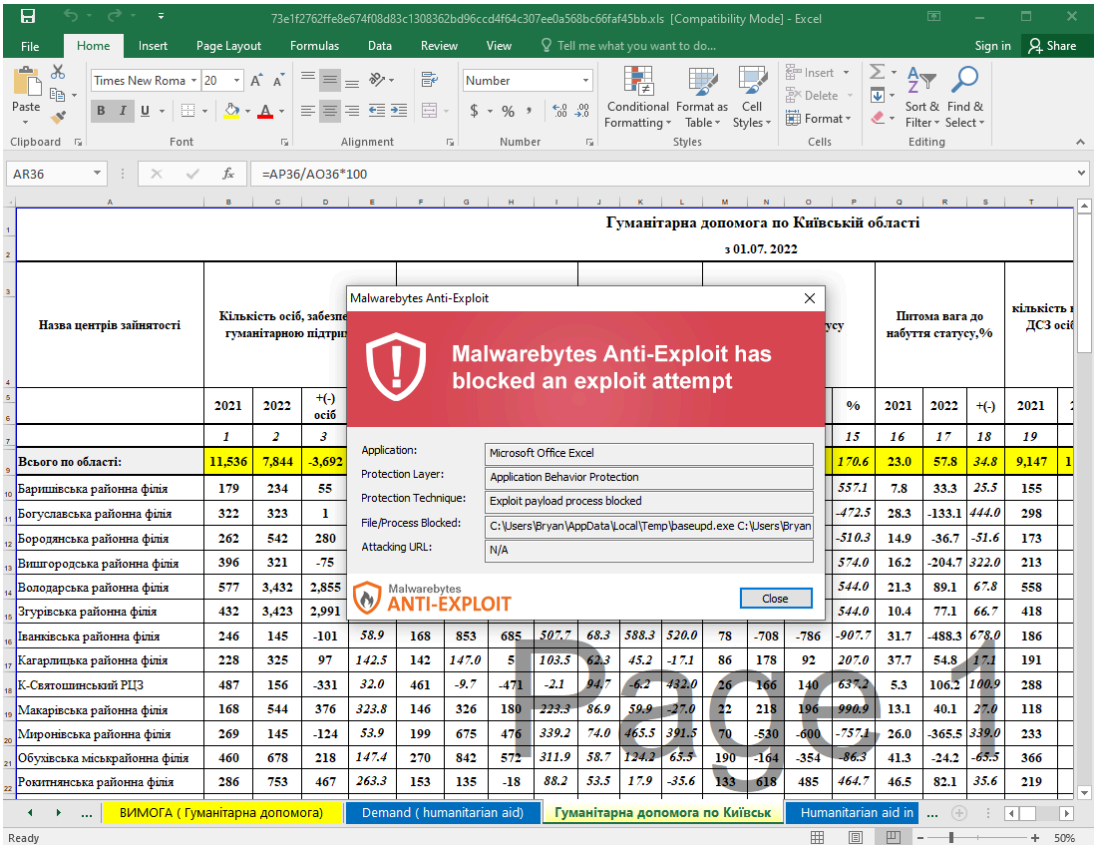
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

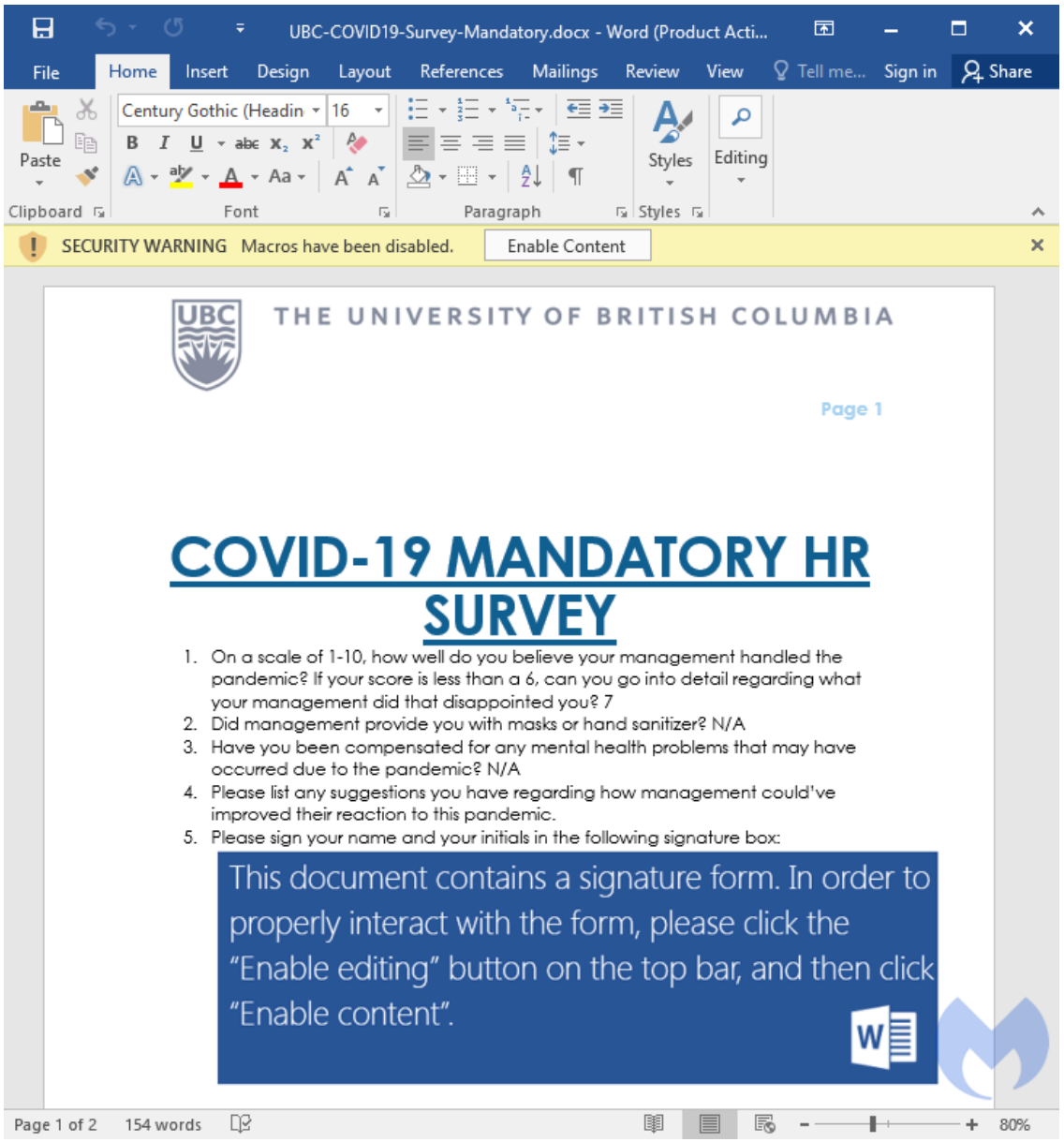
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

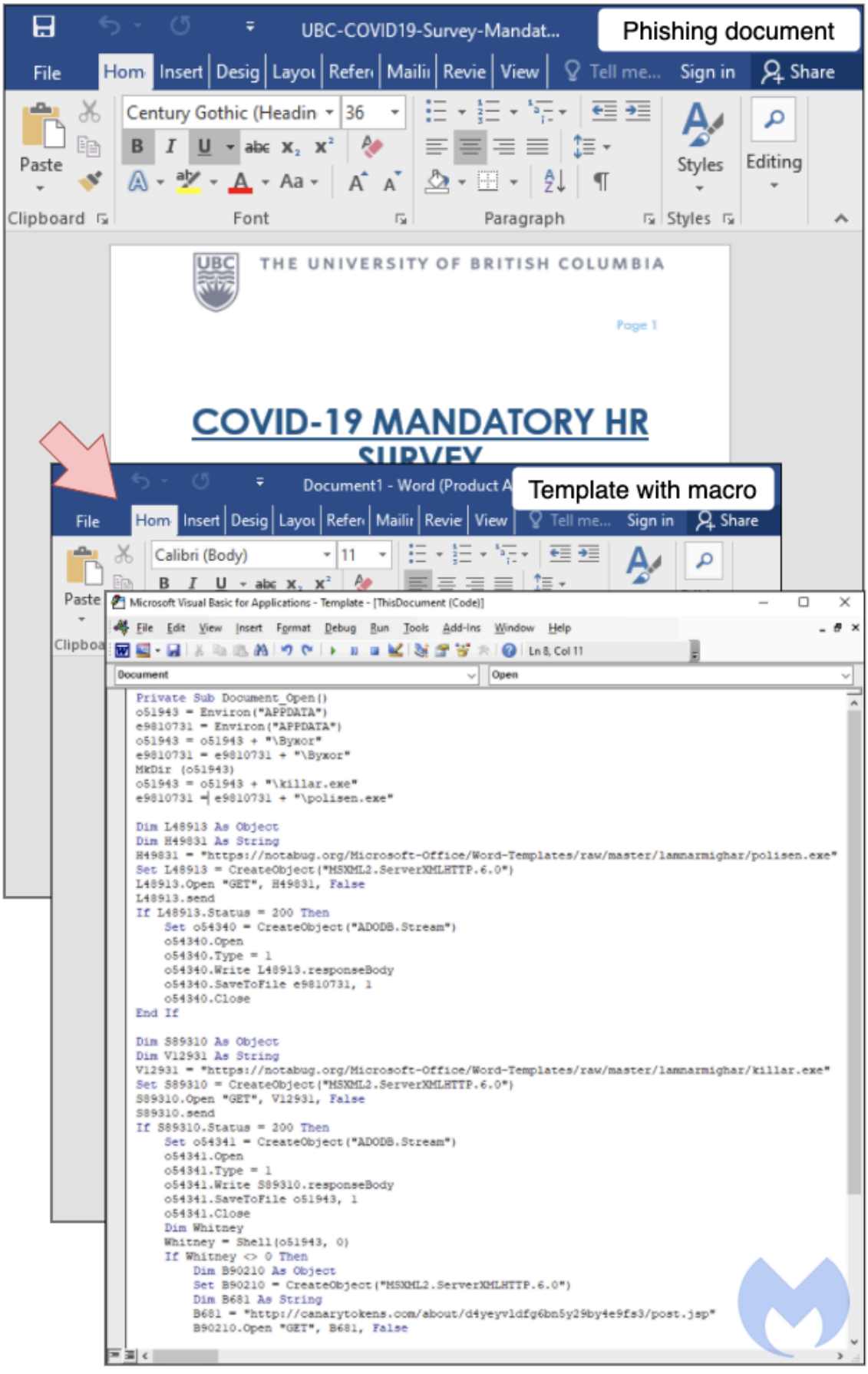
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

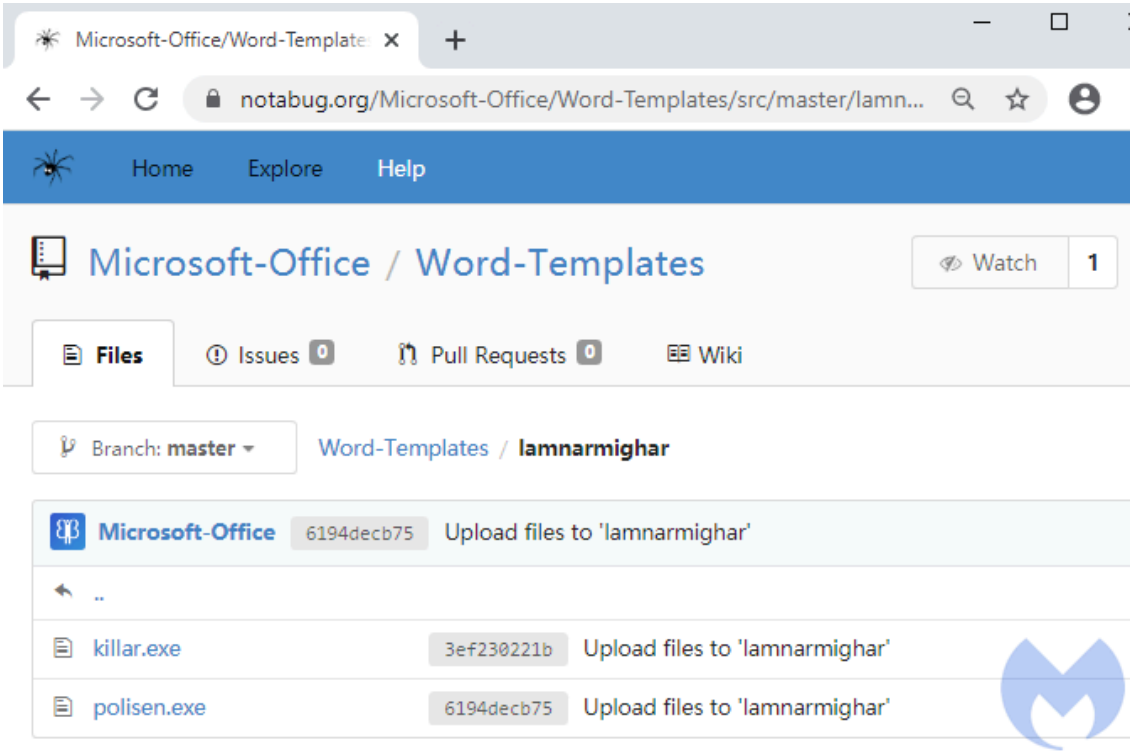
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



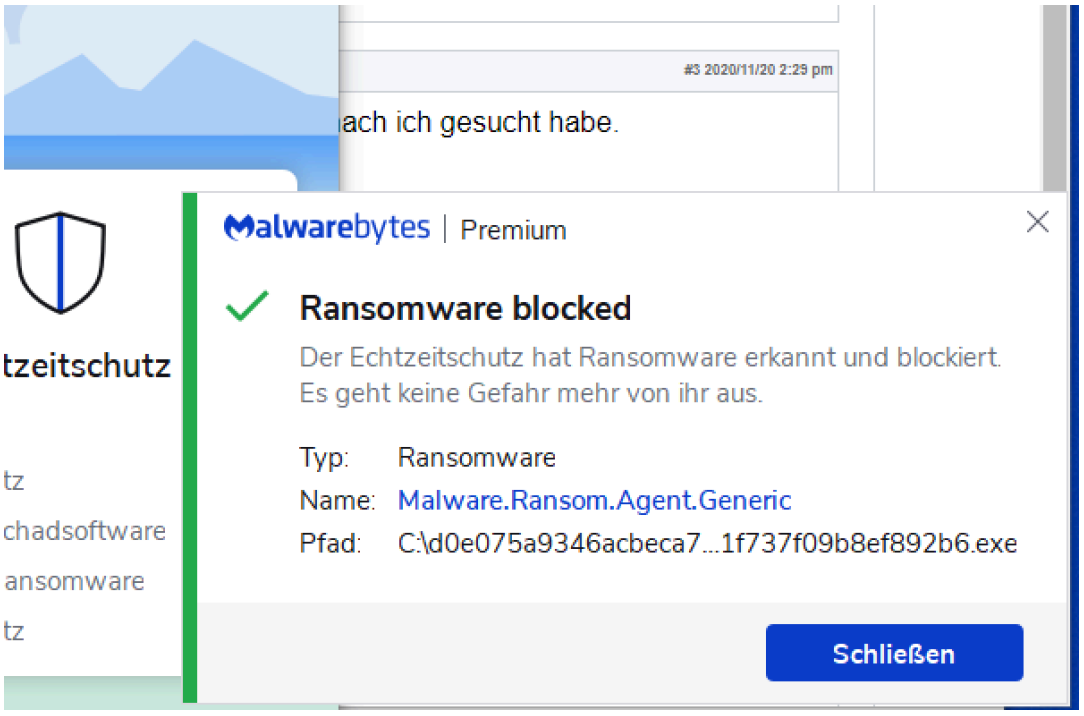
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

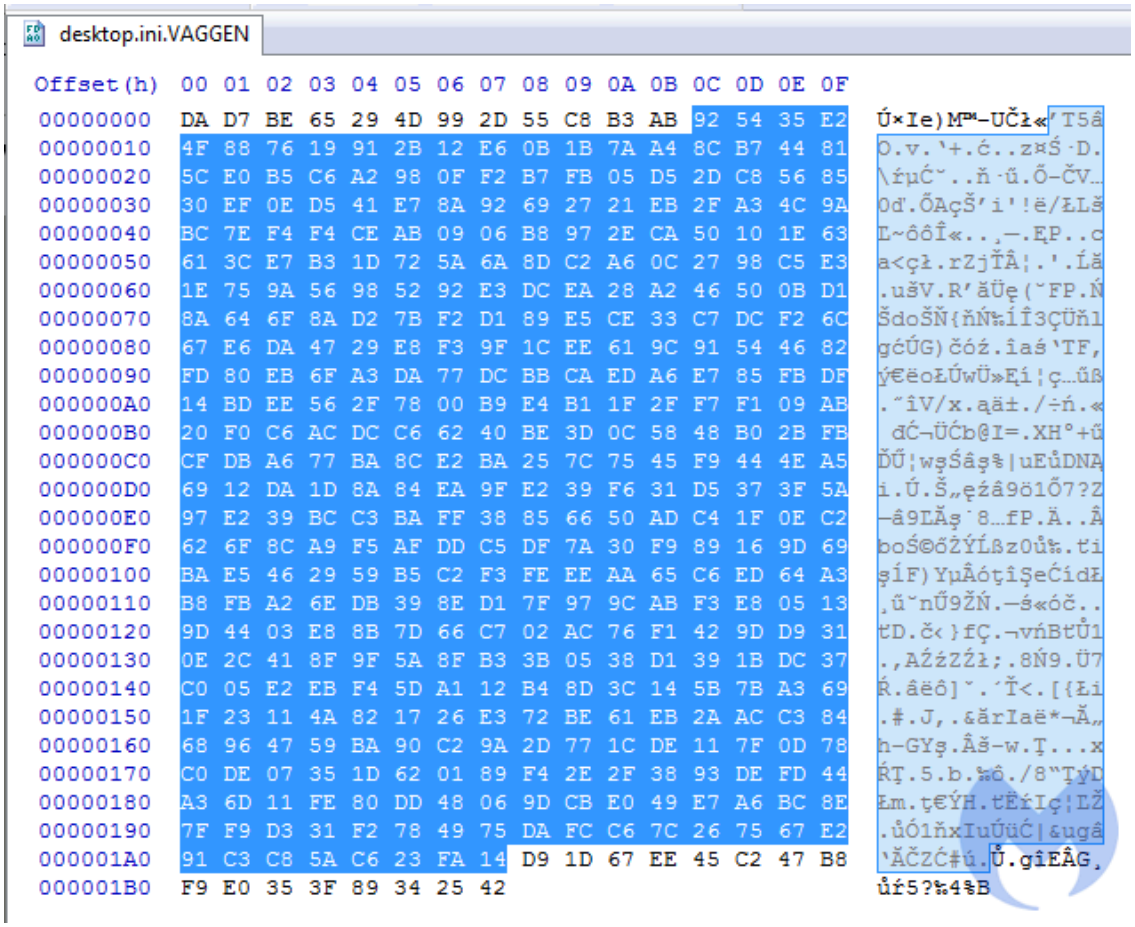
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



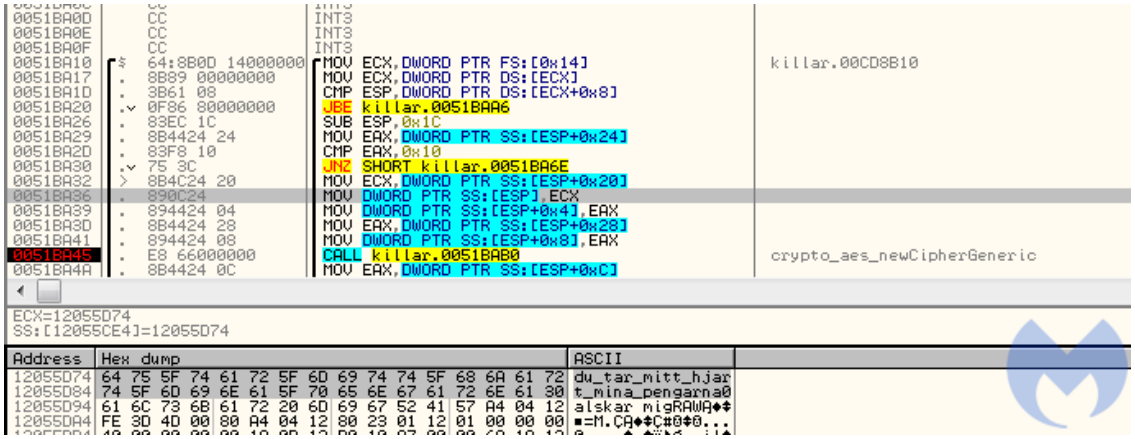
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

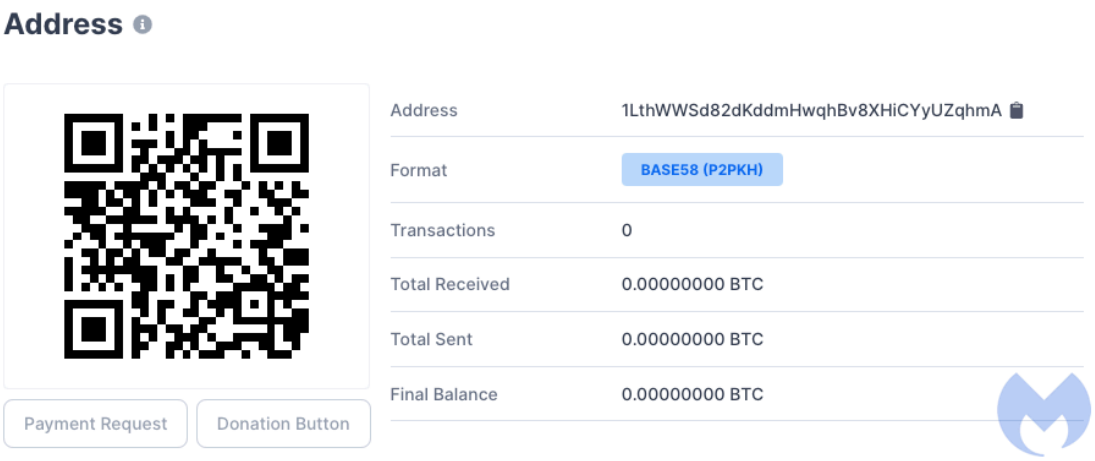
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

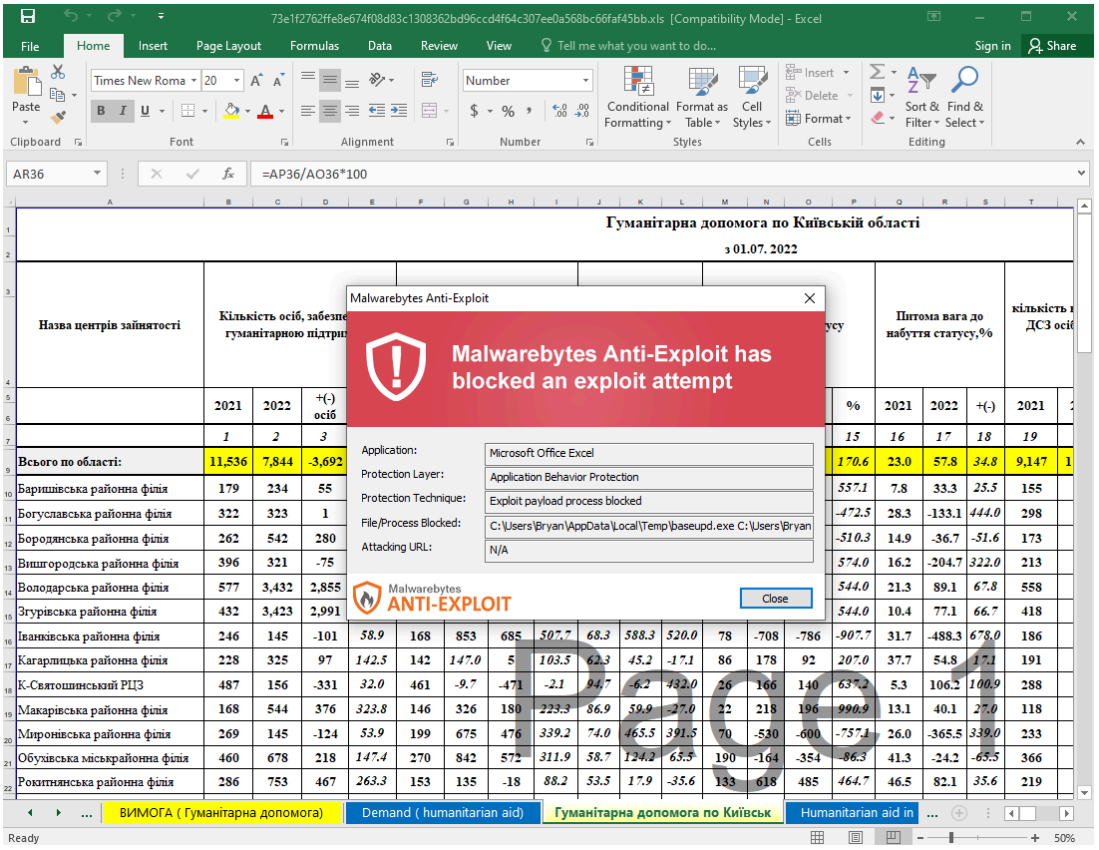
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

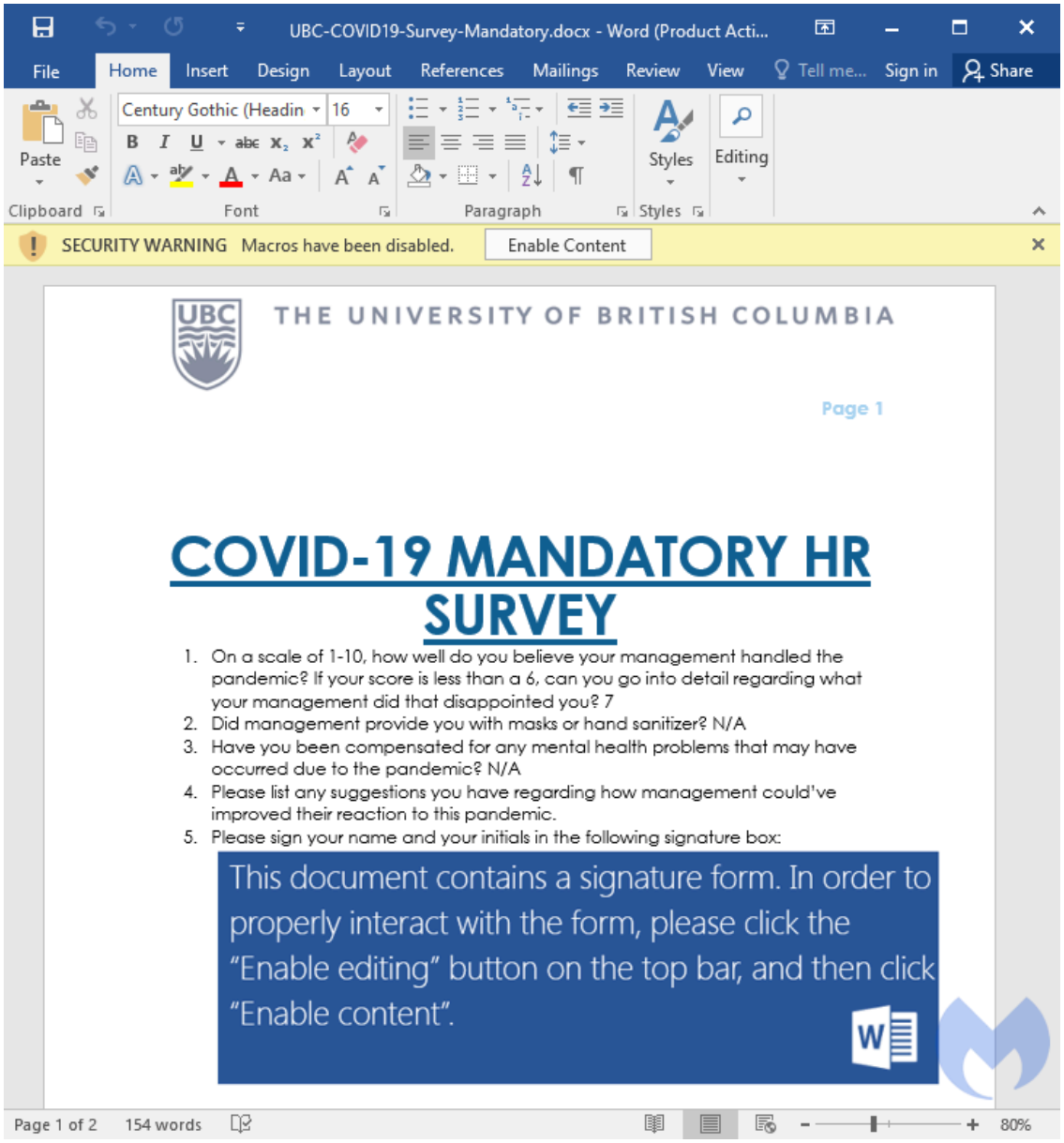
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

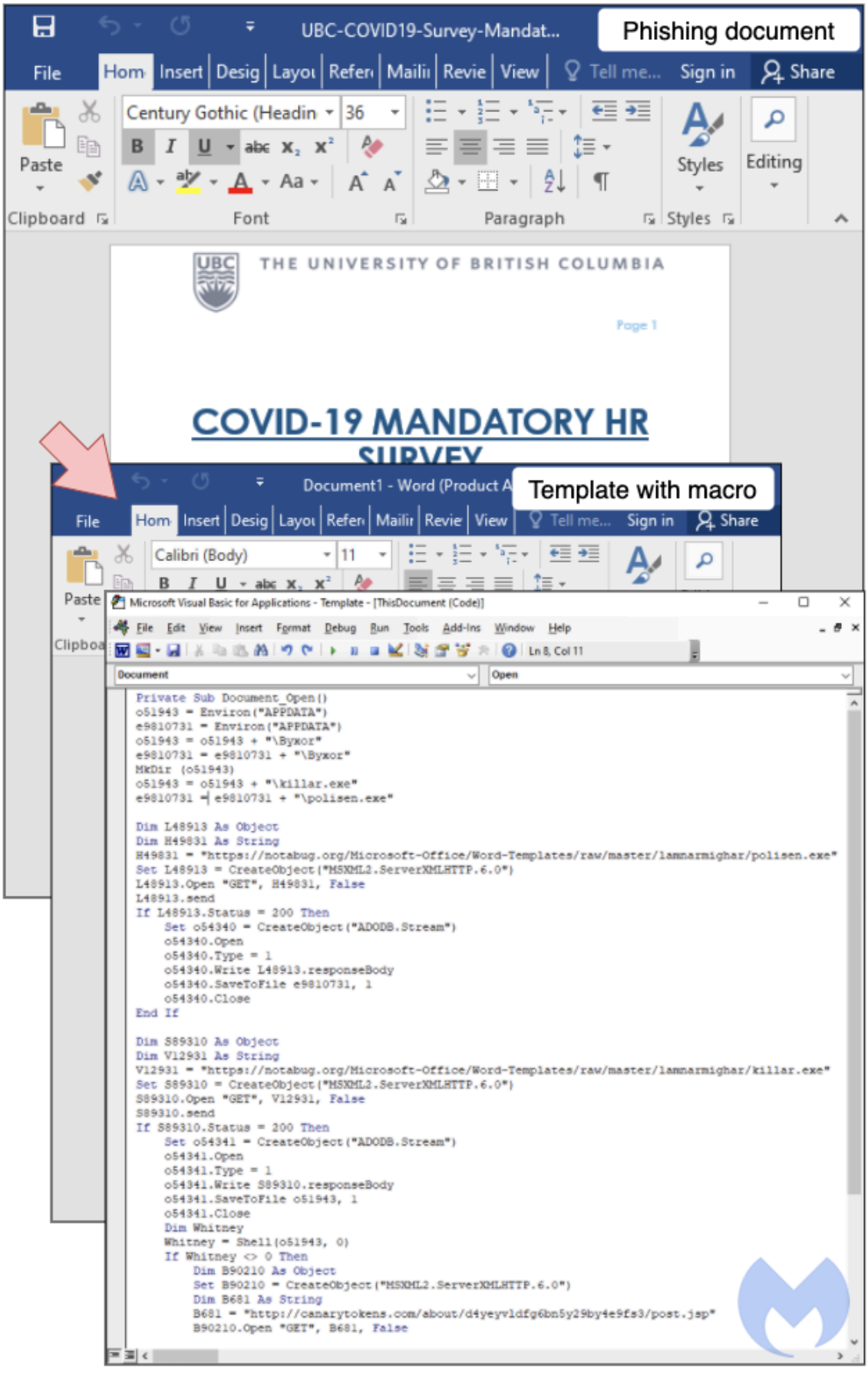
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

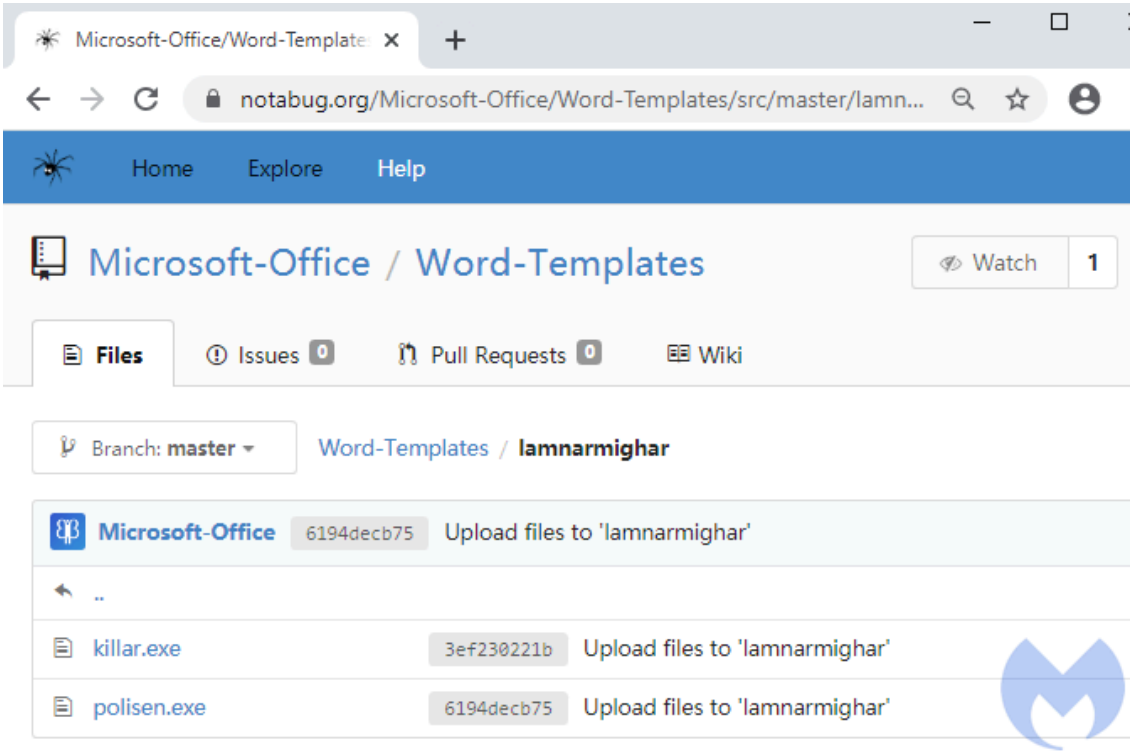
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



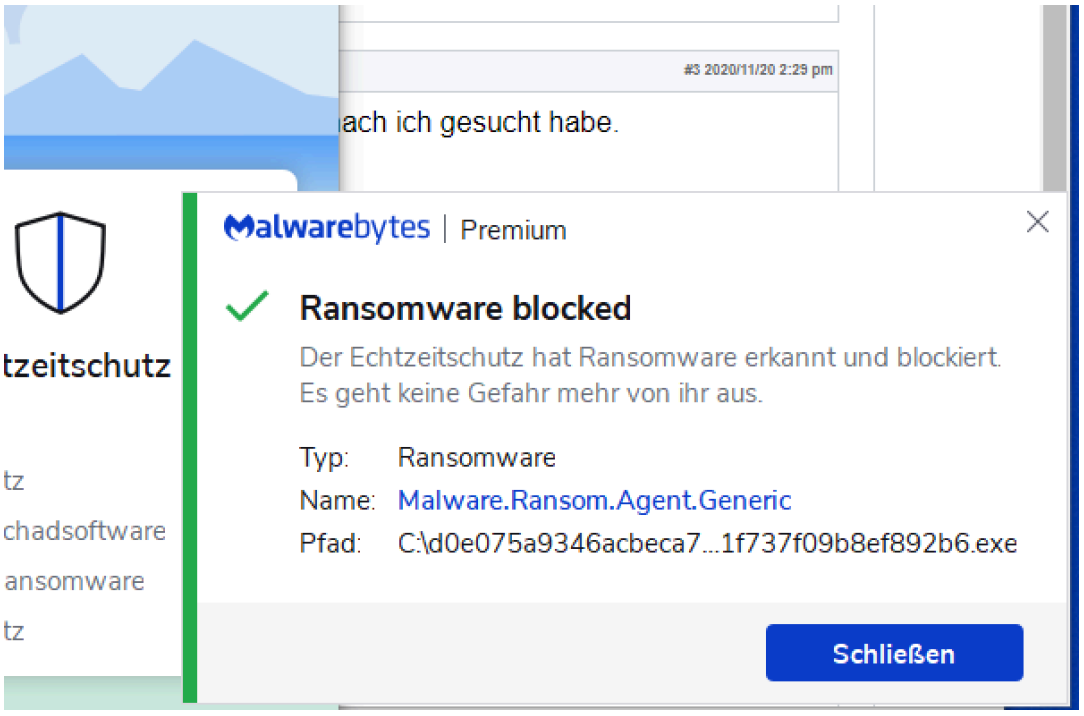
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

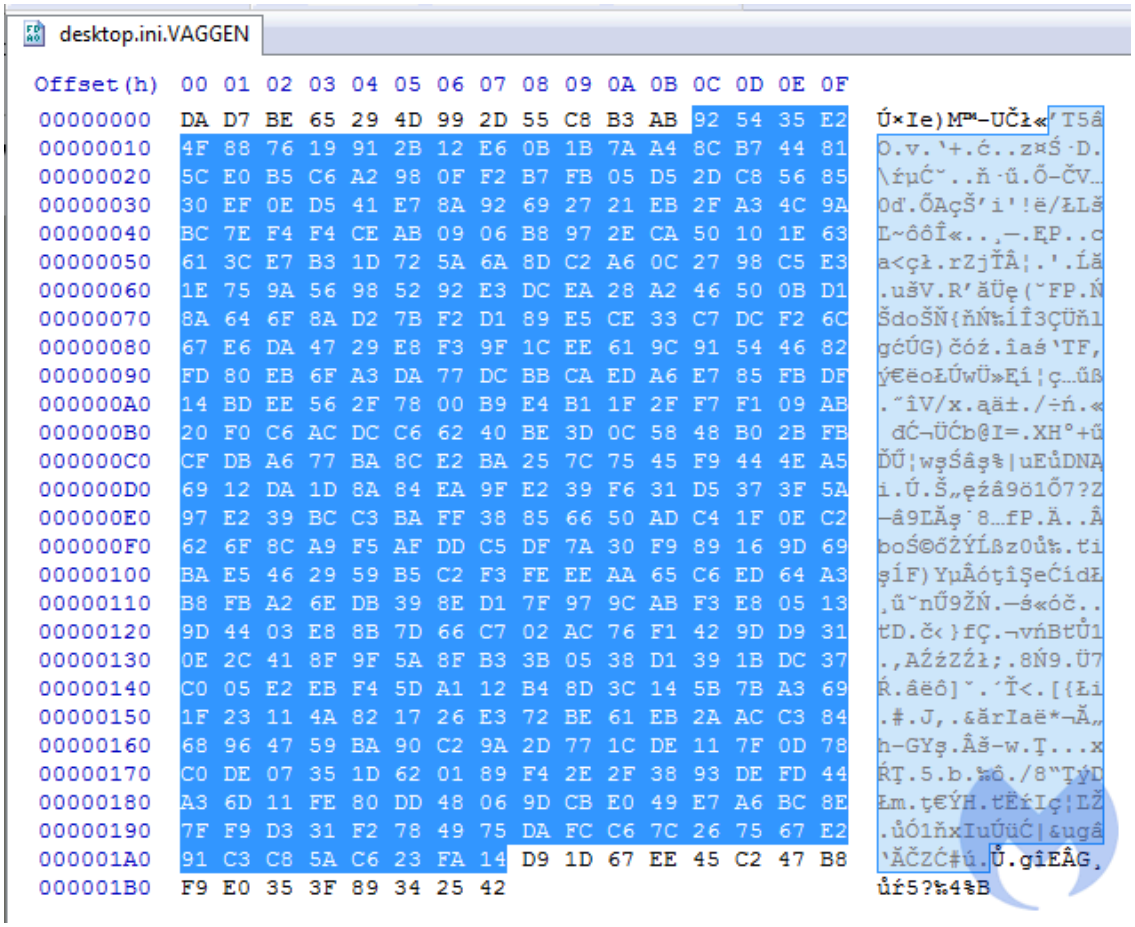
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



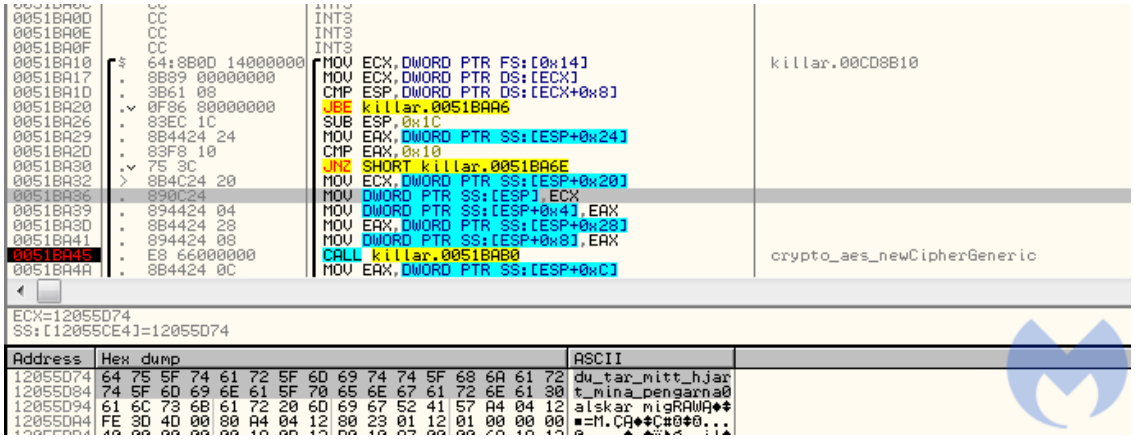
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

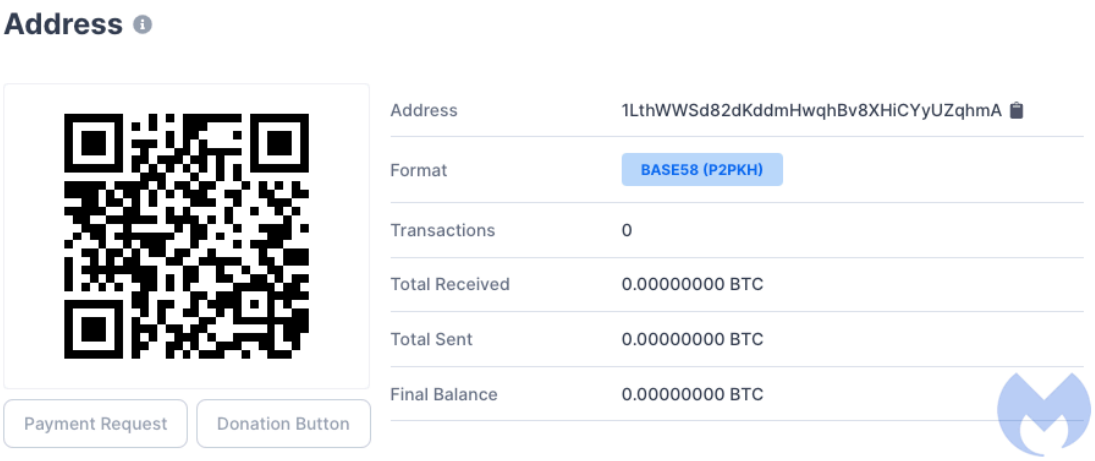
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

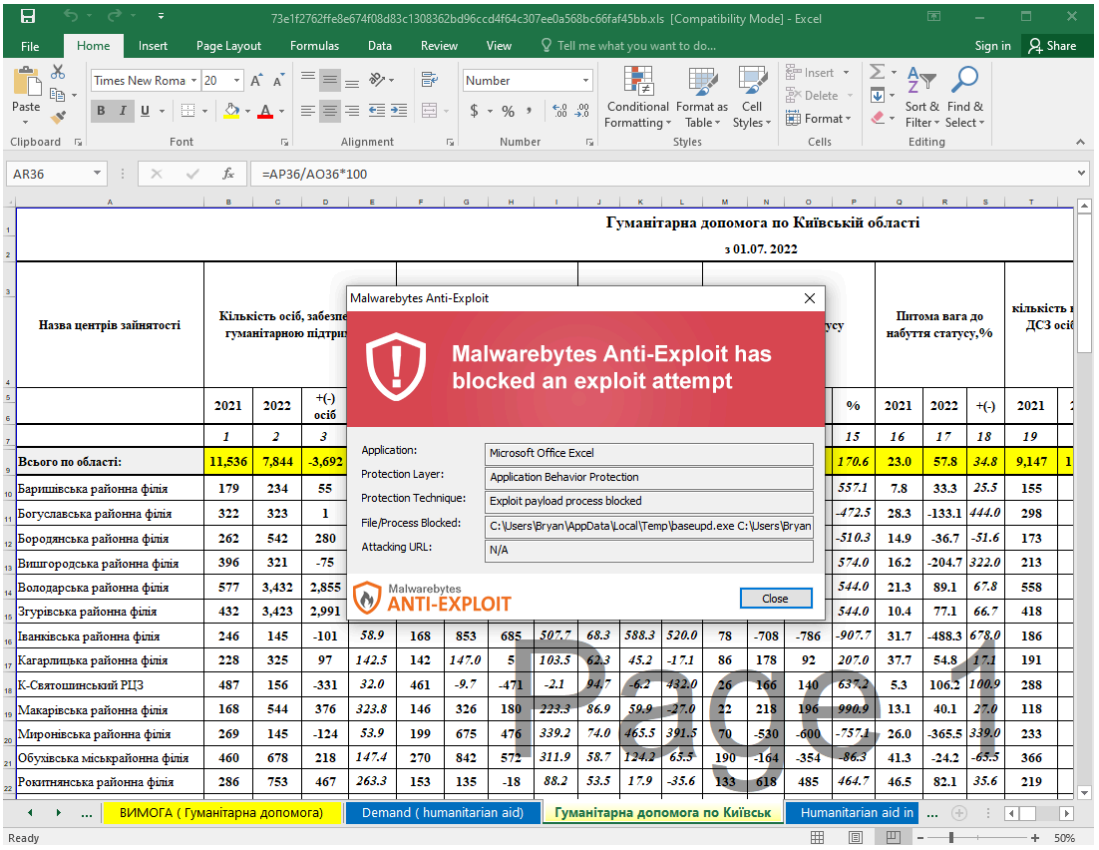
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

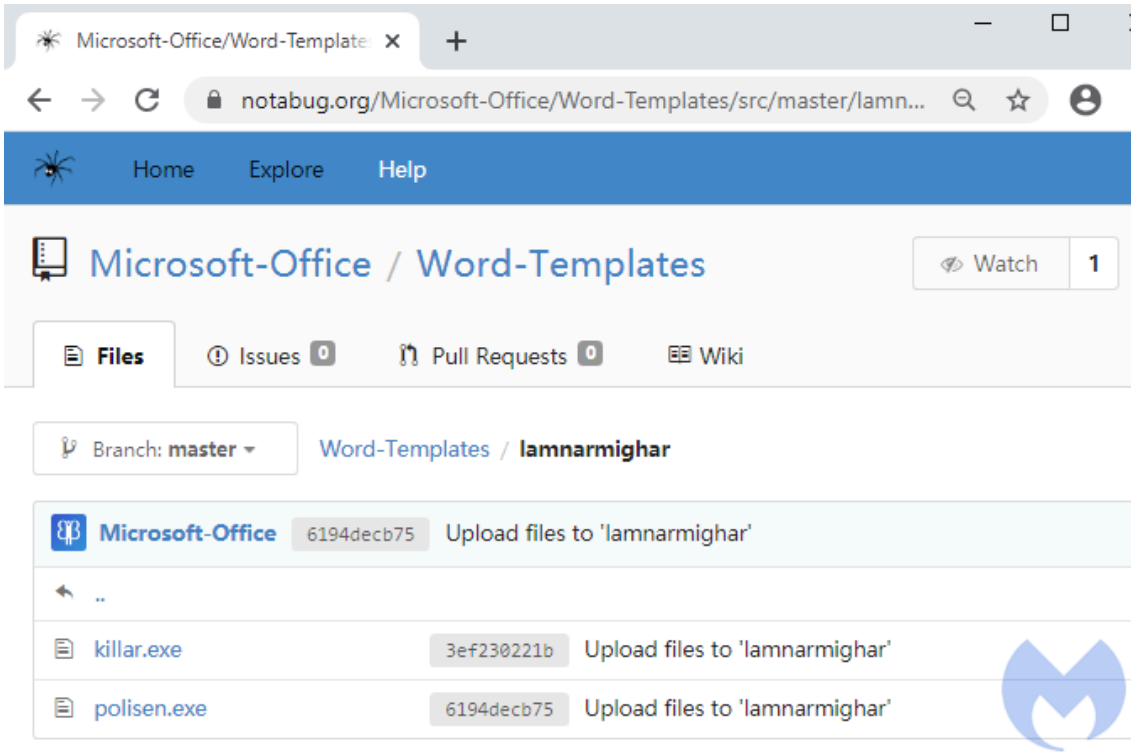
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



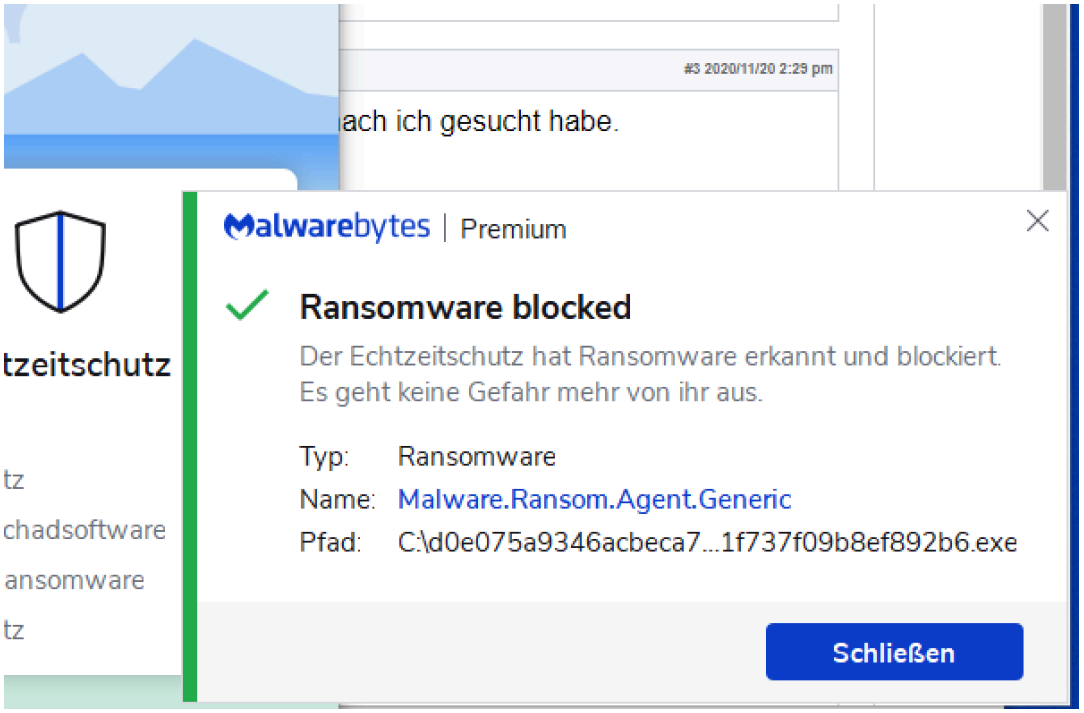
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

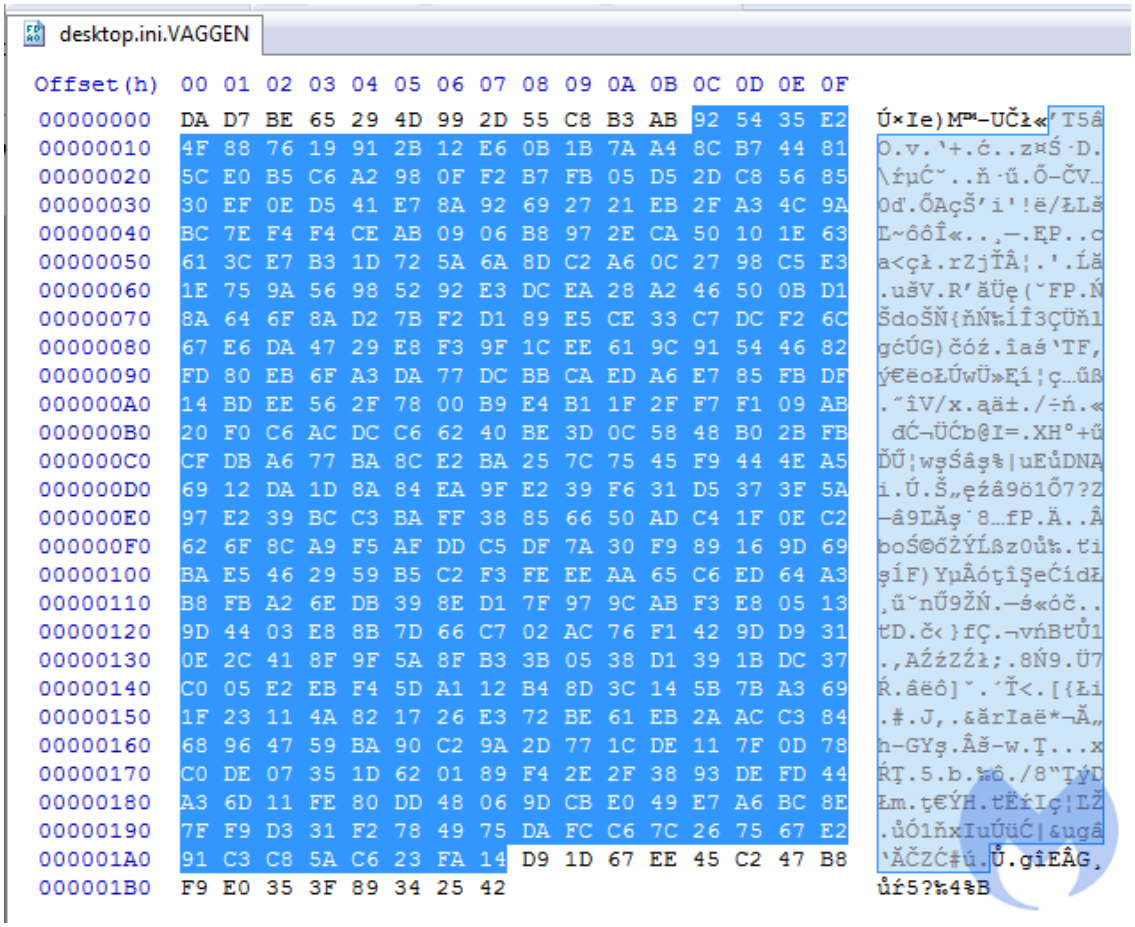
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

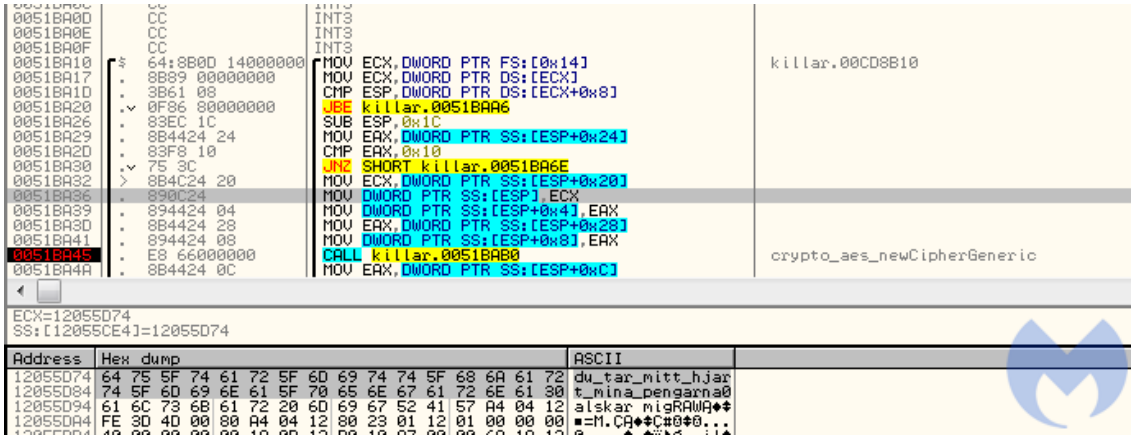


The content of the output file (with .VAGGEN extension) contains:

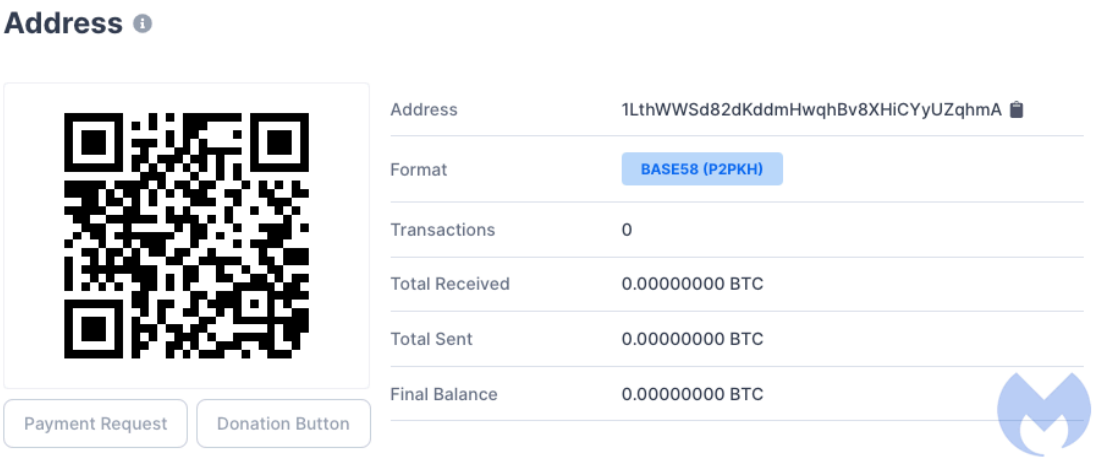
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

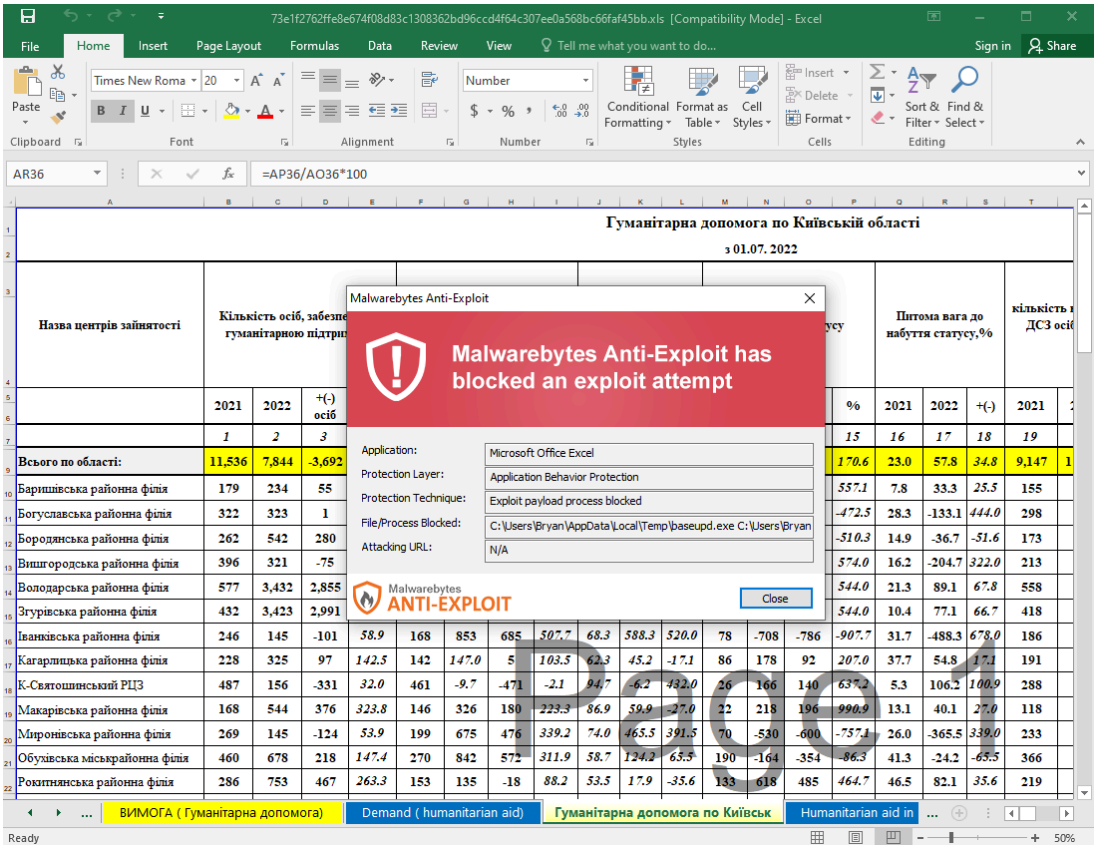
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

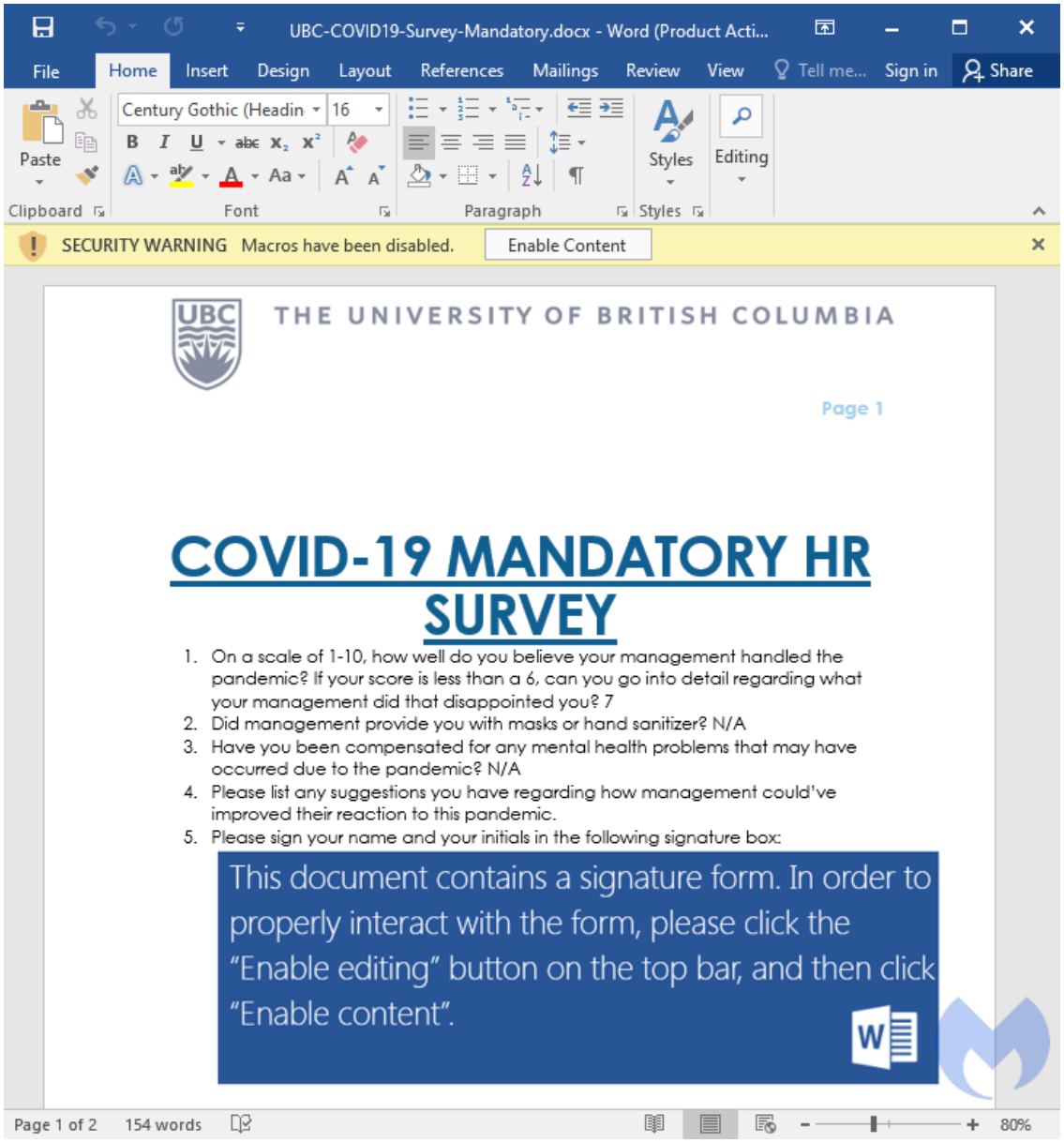
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

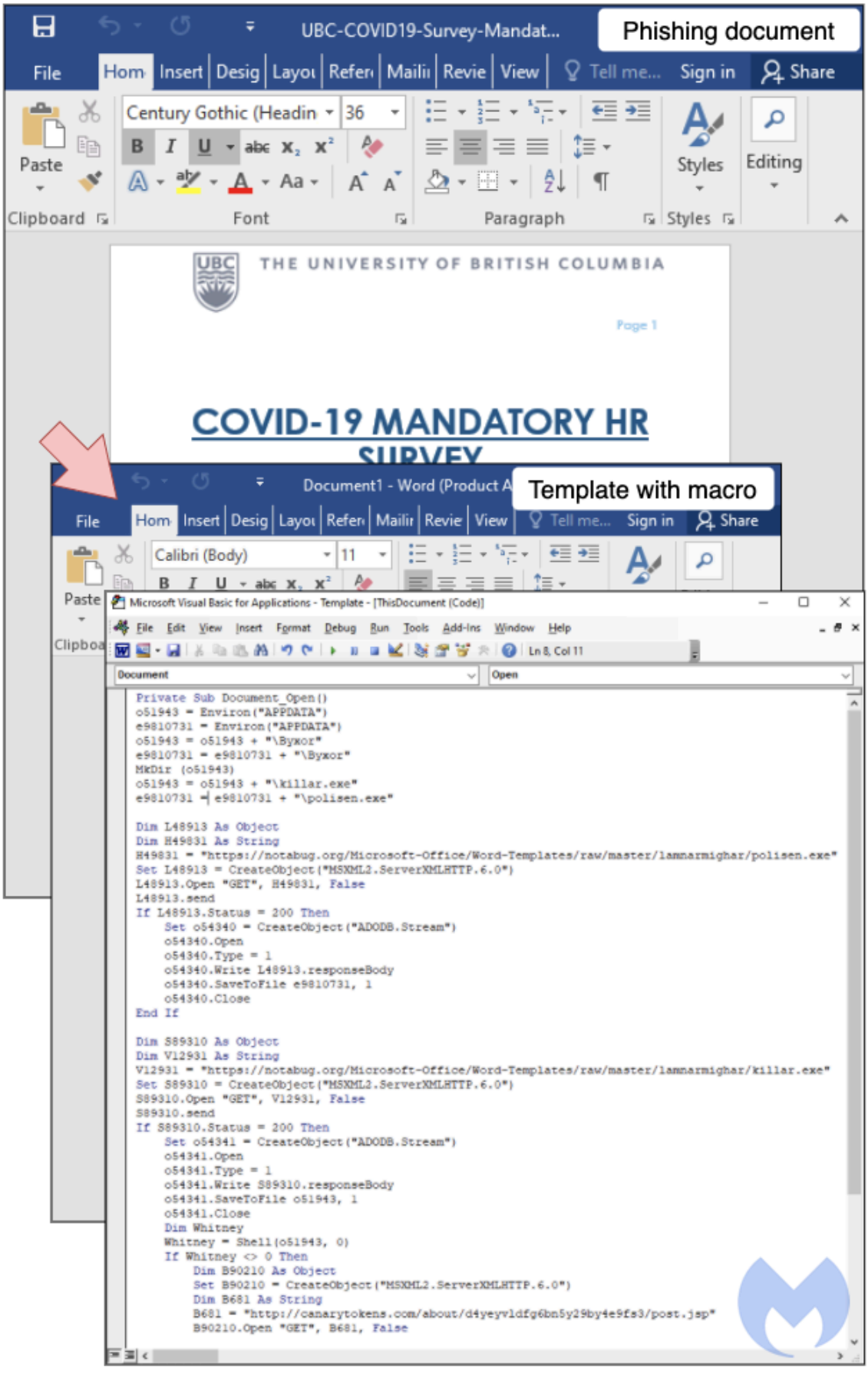
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

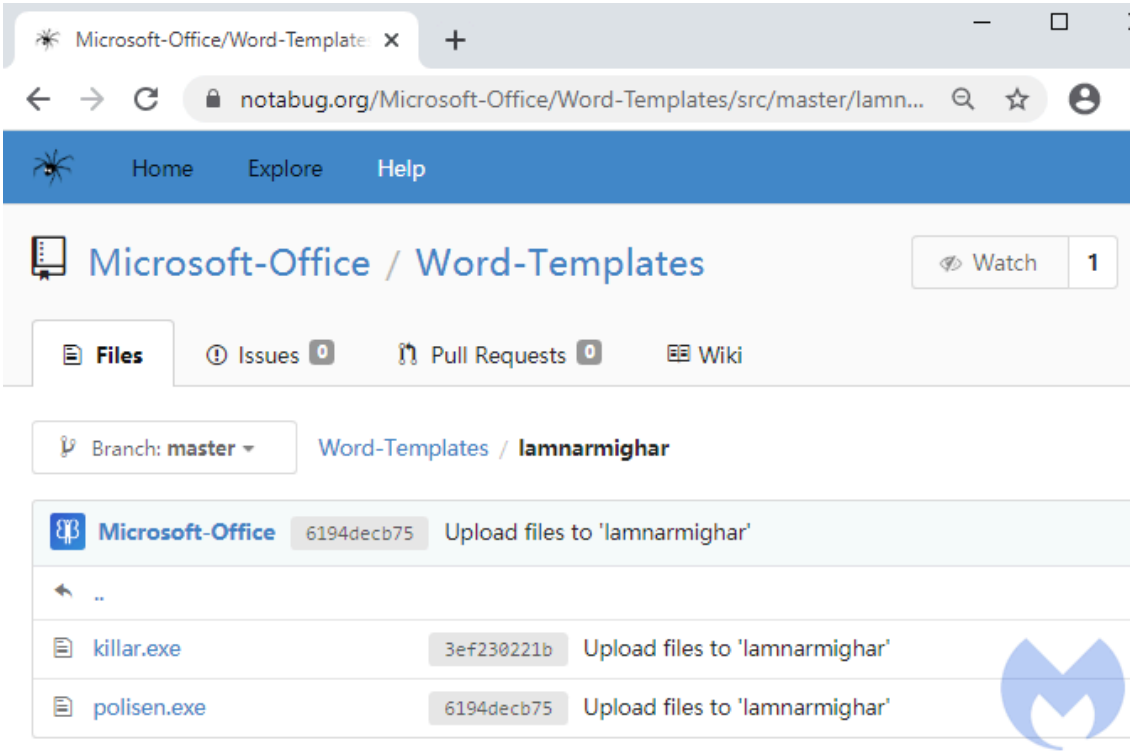
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



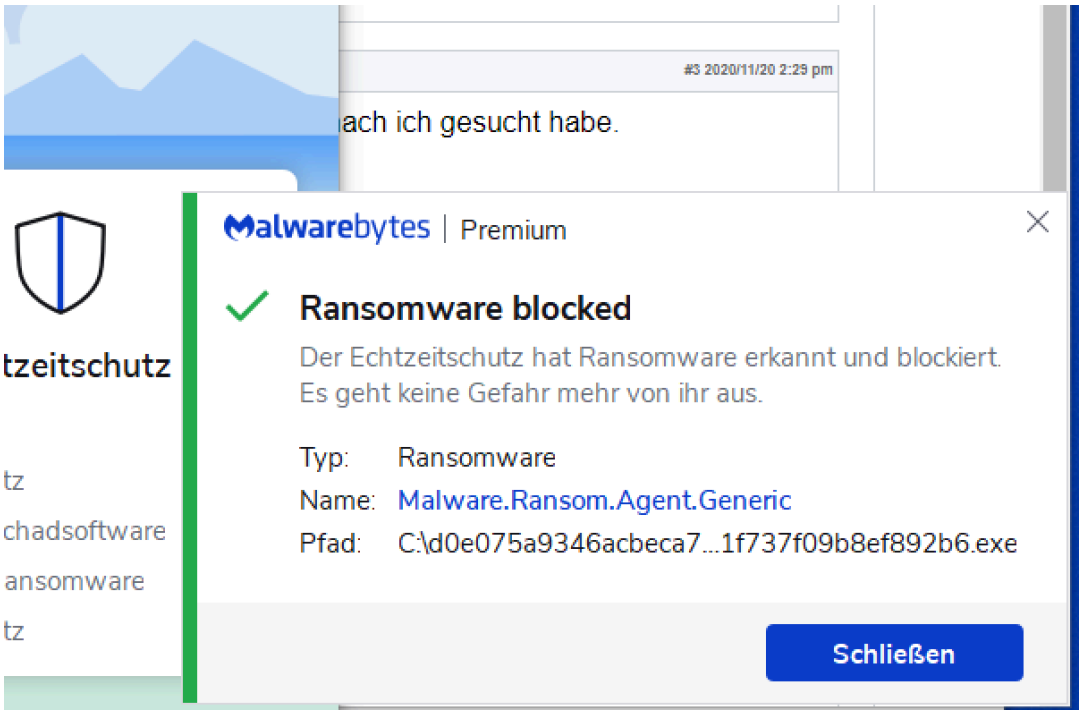
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

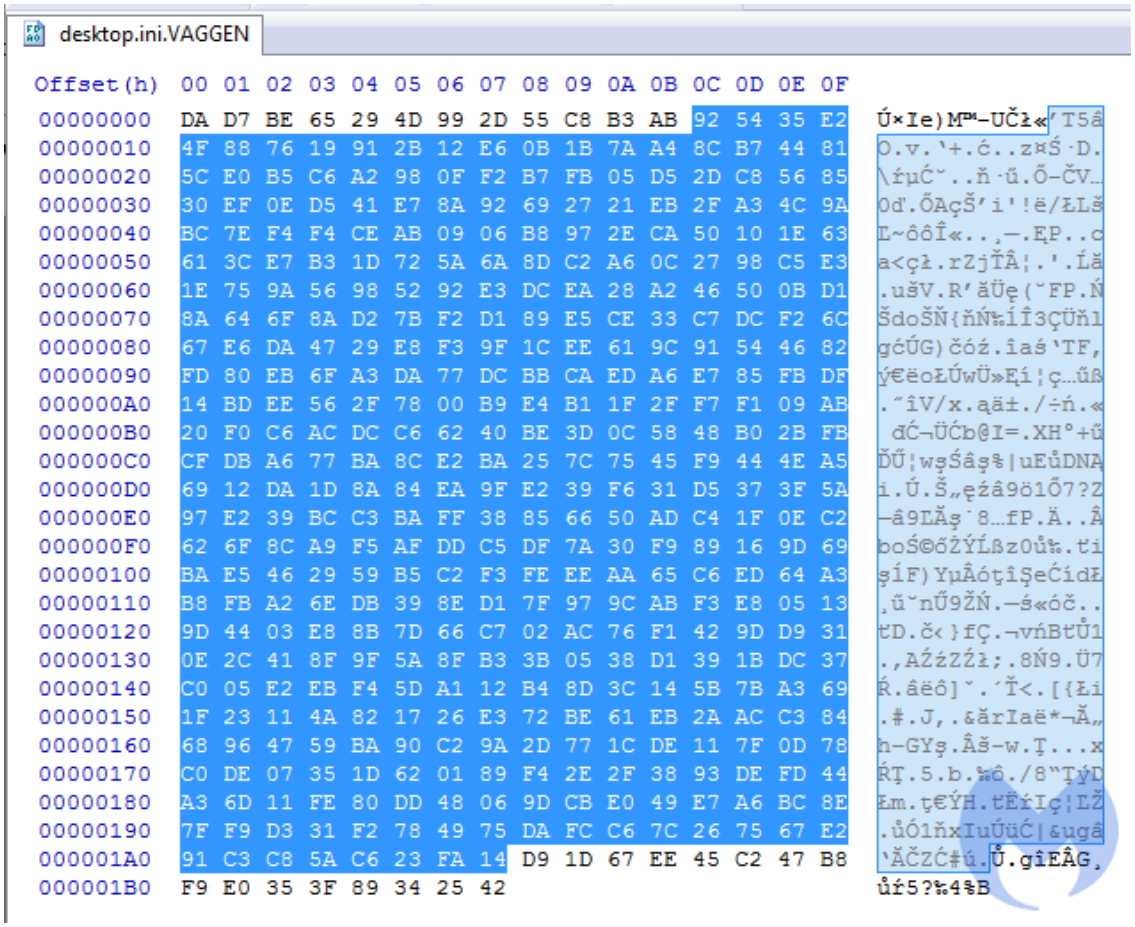
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



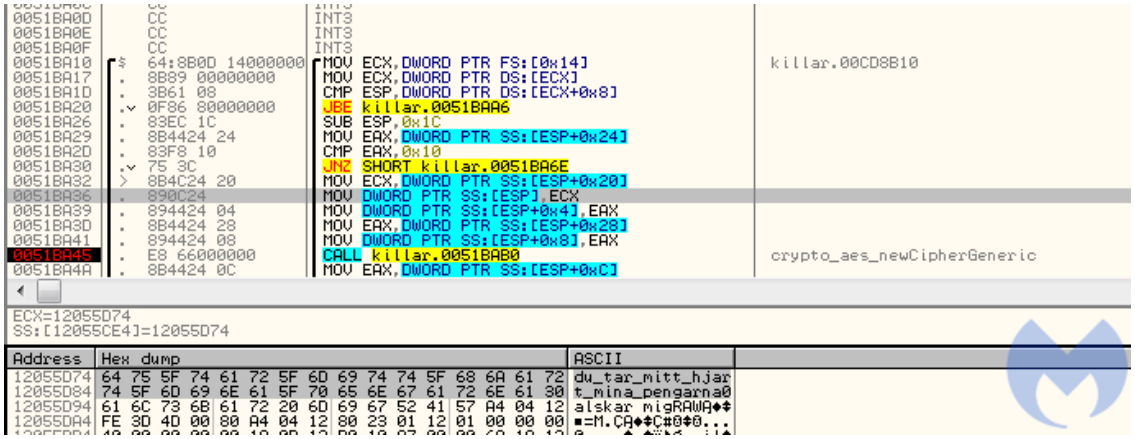
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

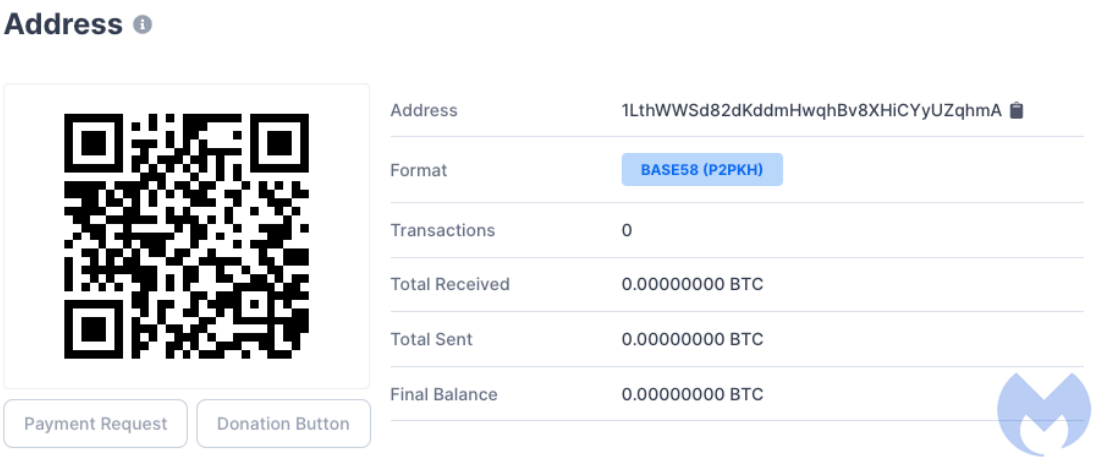
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

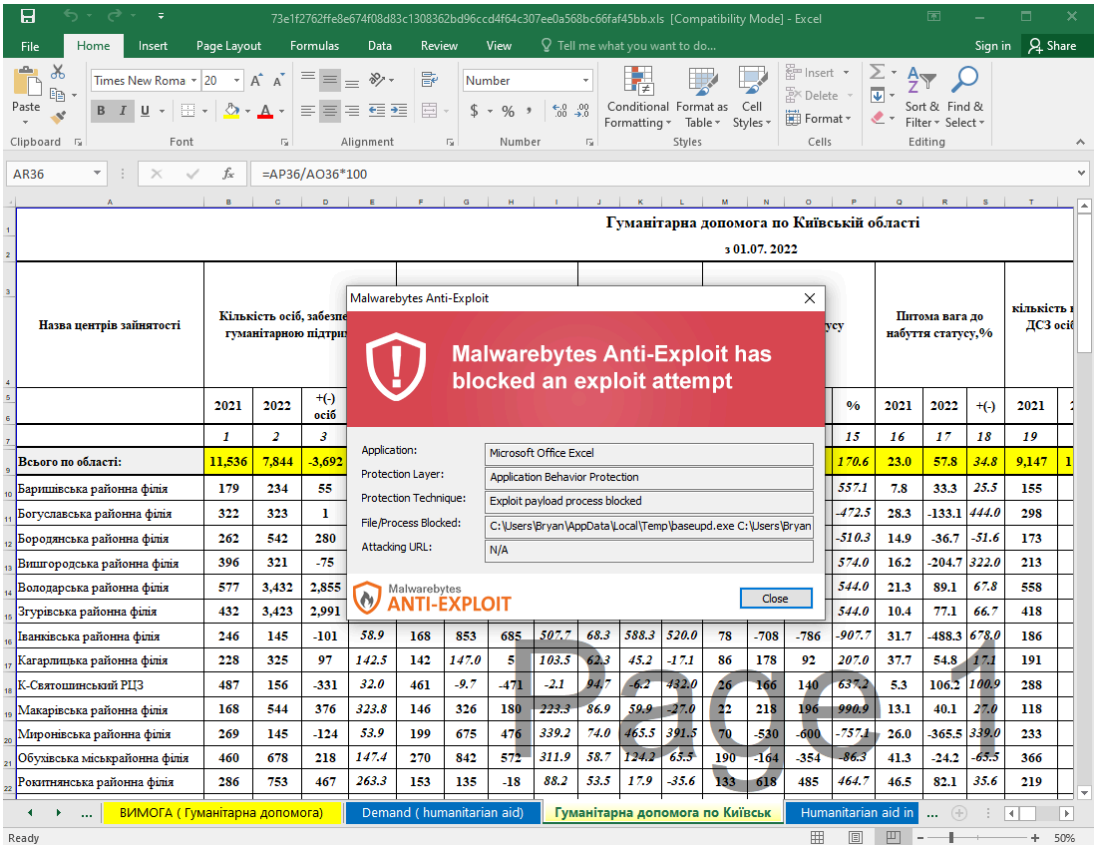
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

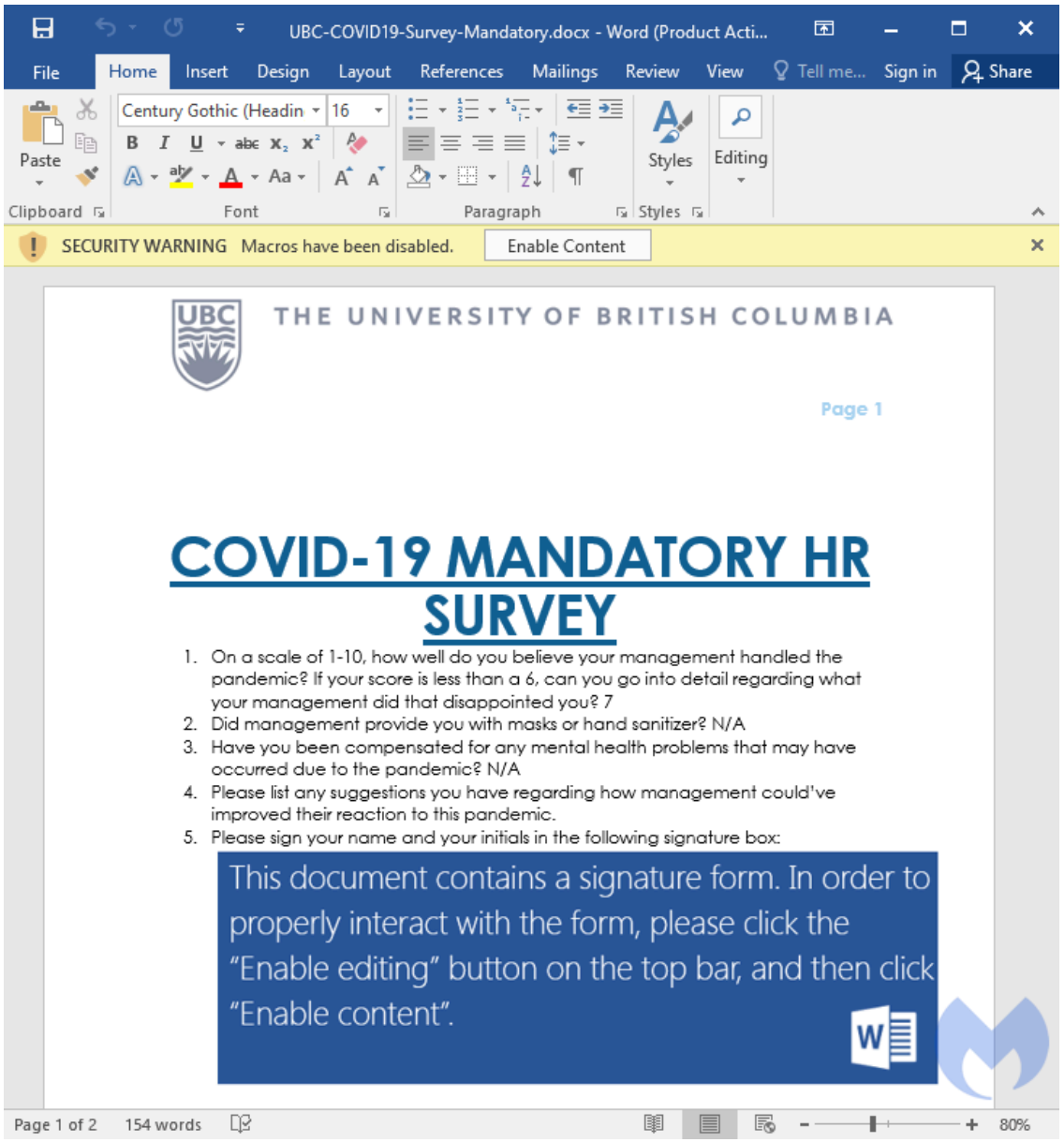
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

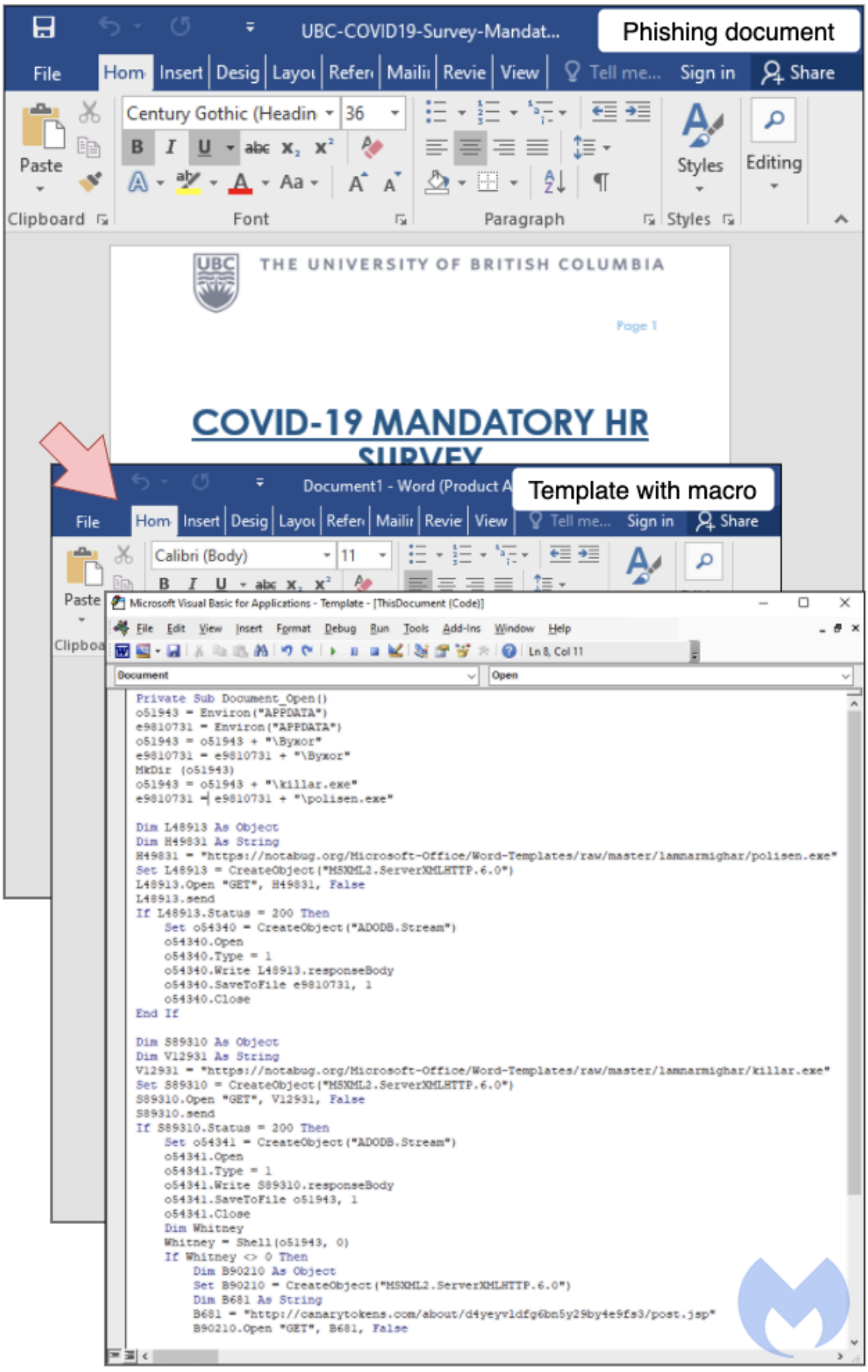
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

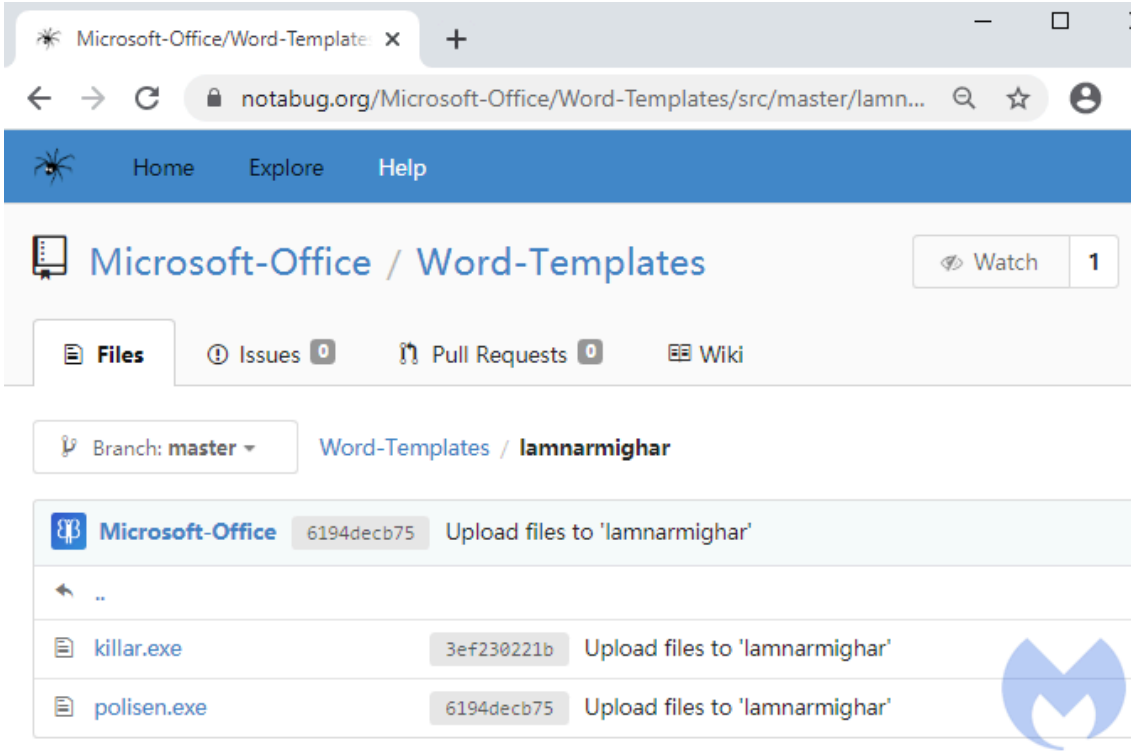
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



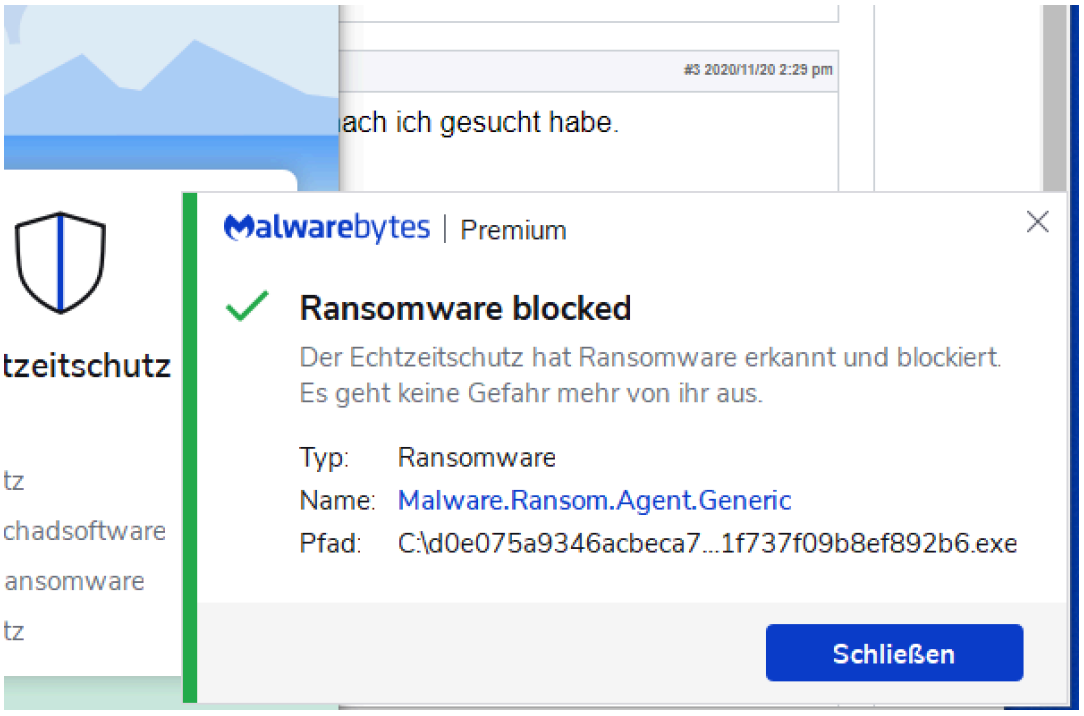
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

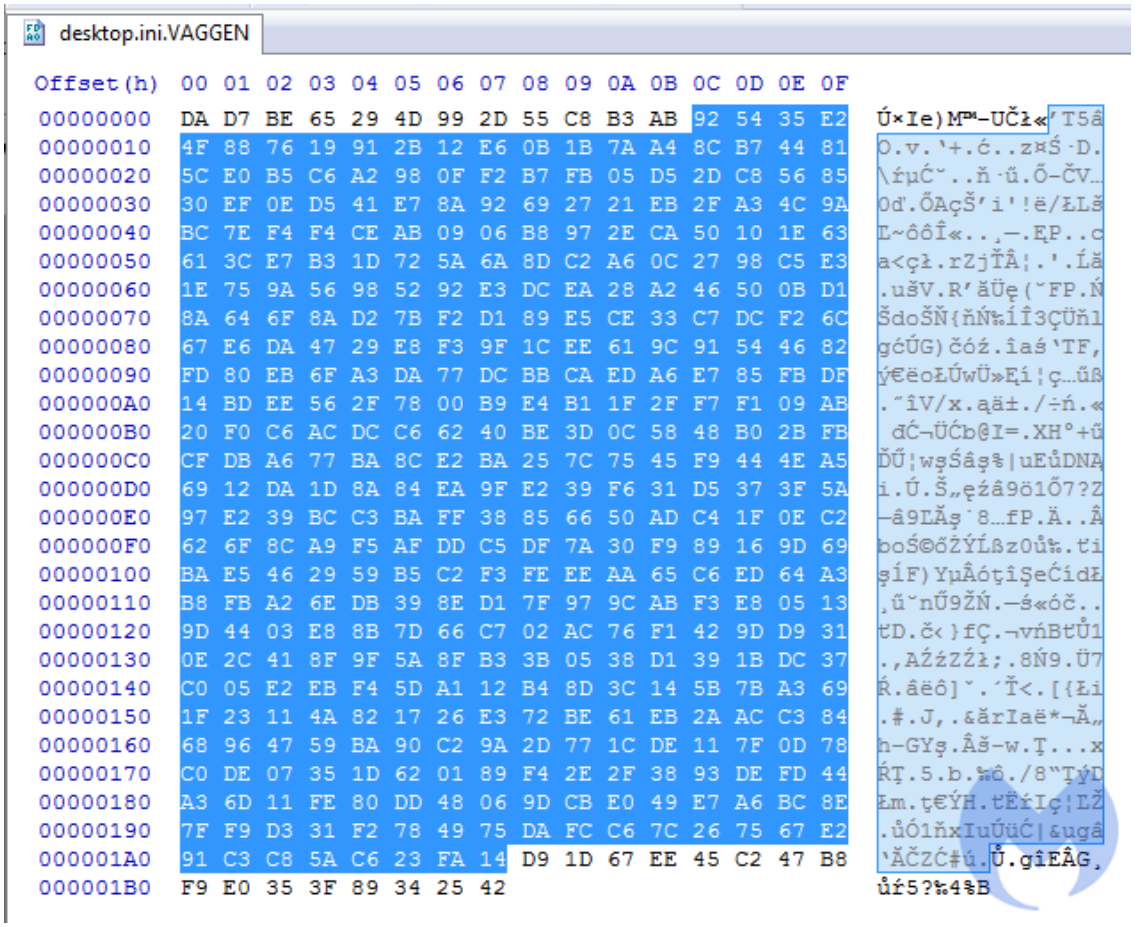
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



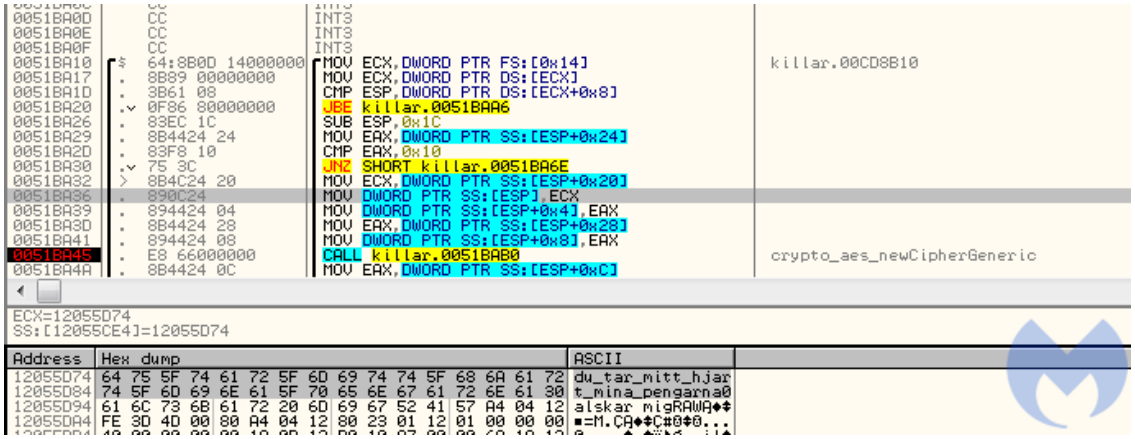
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

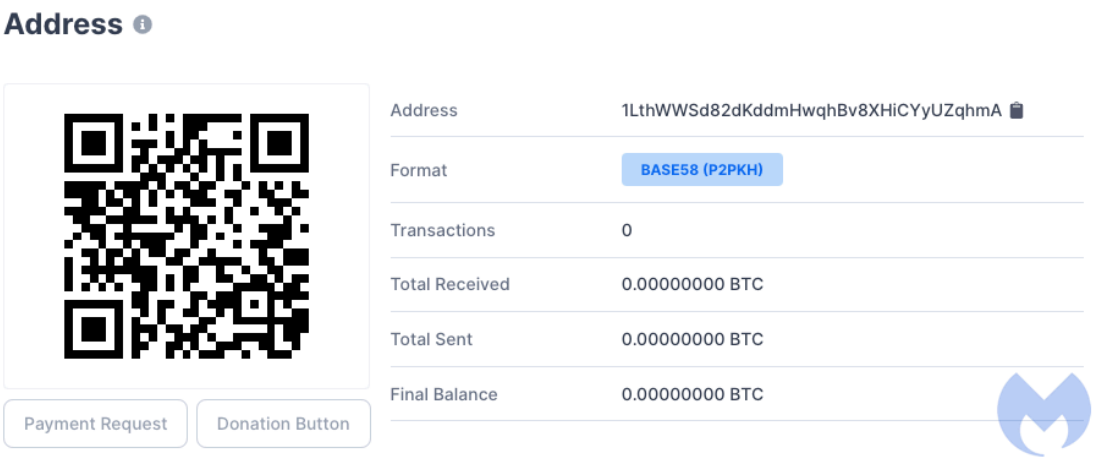
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

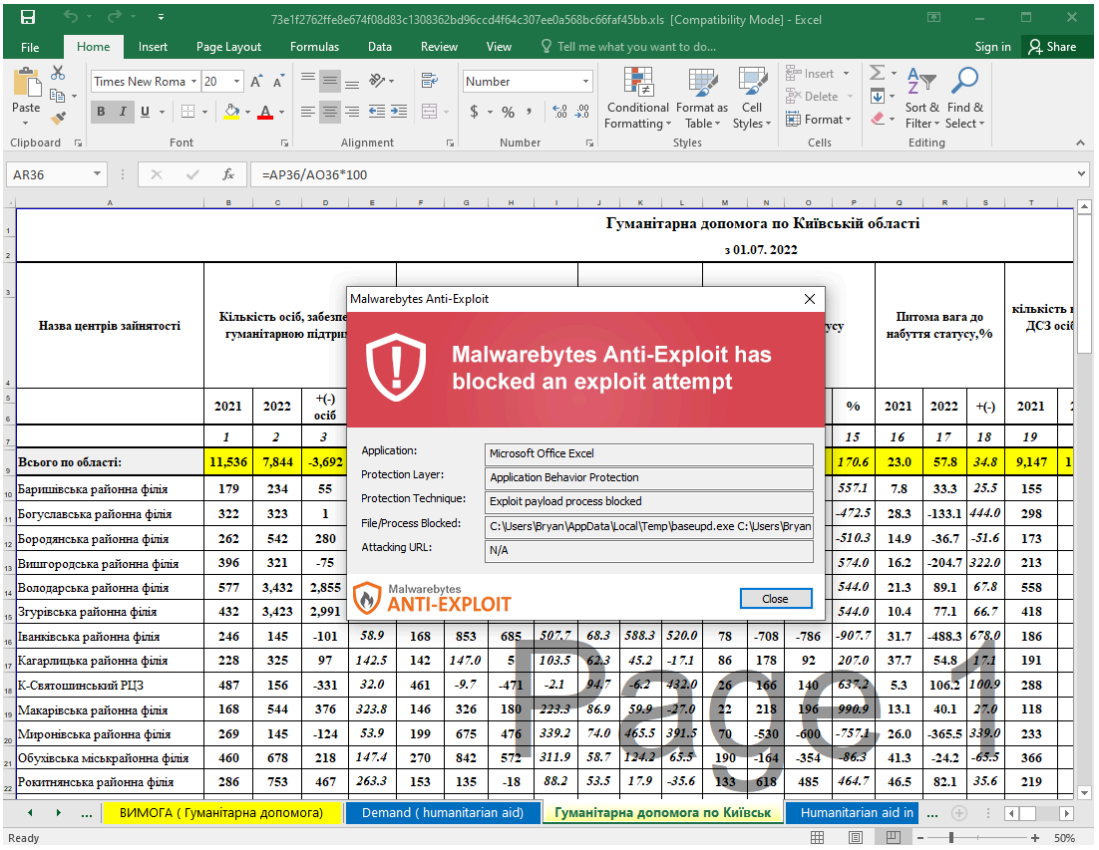
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

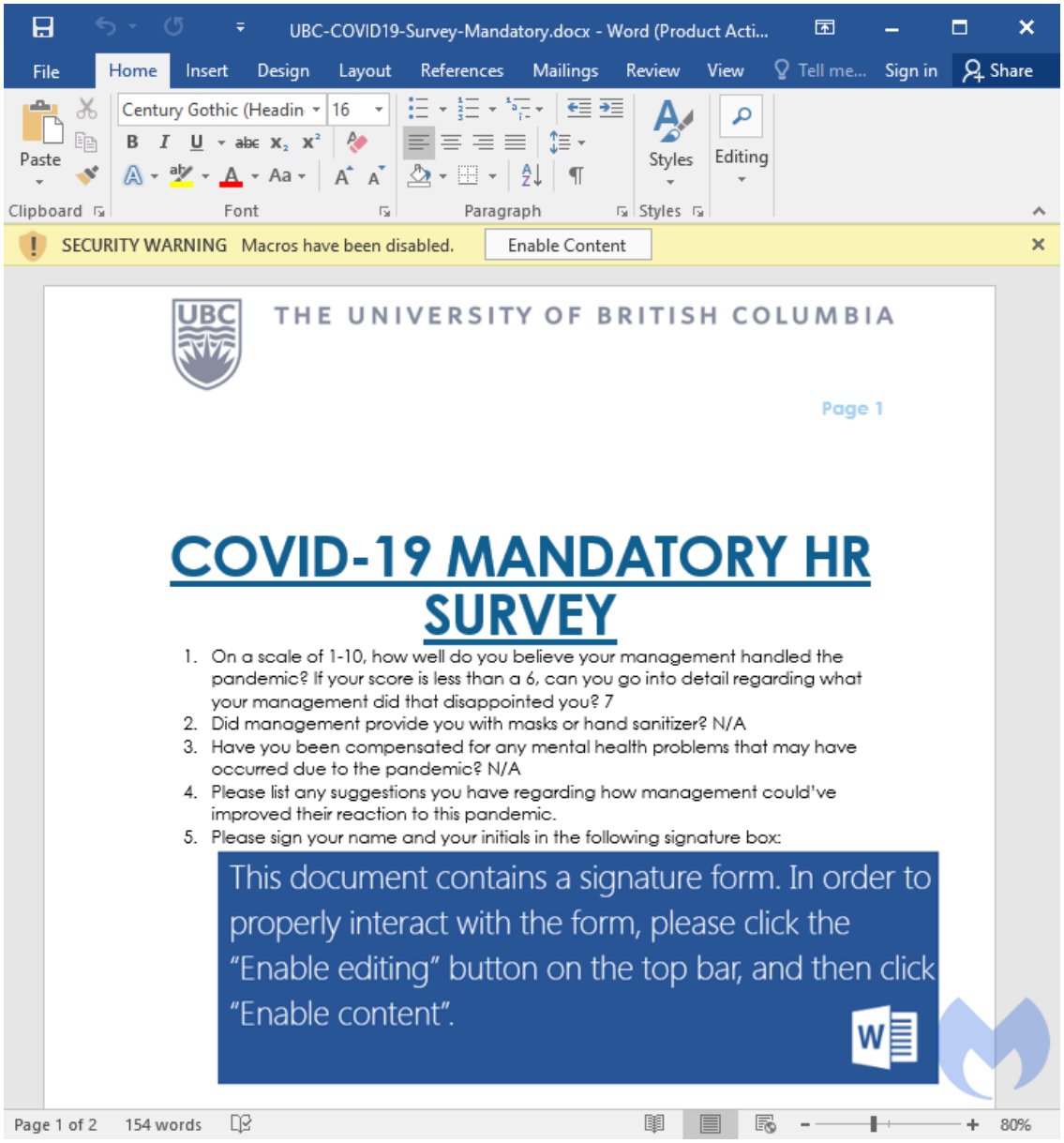
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

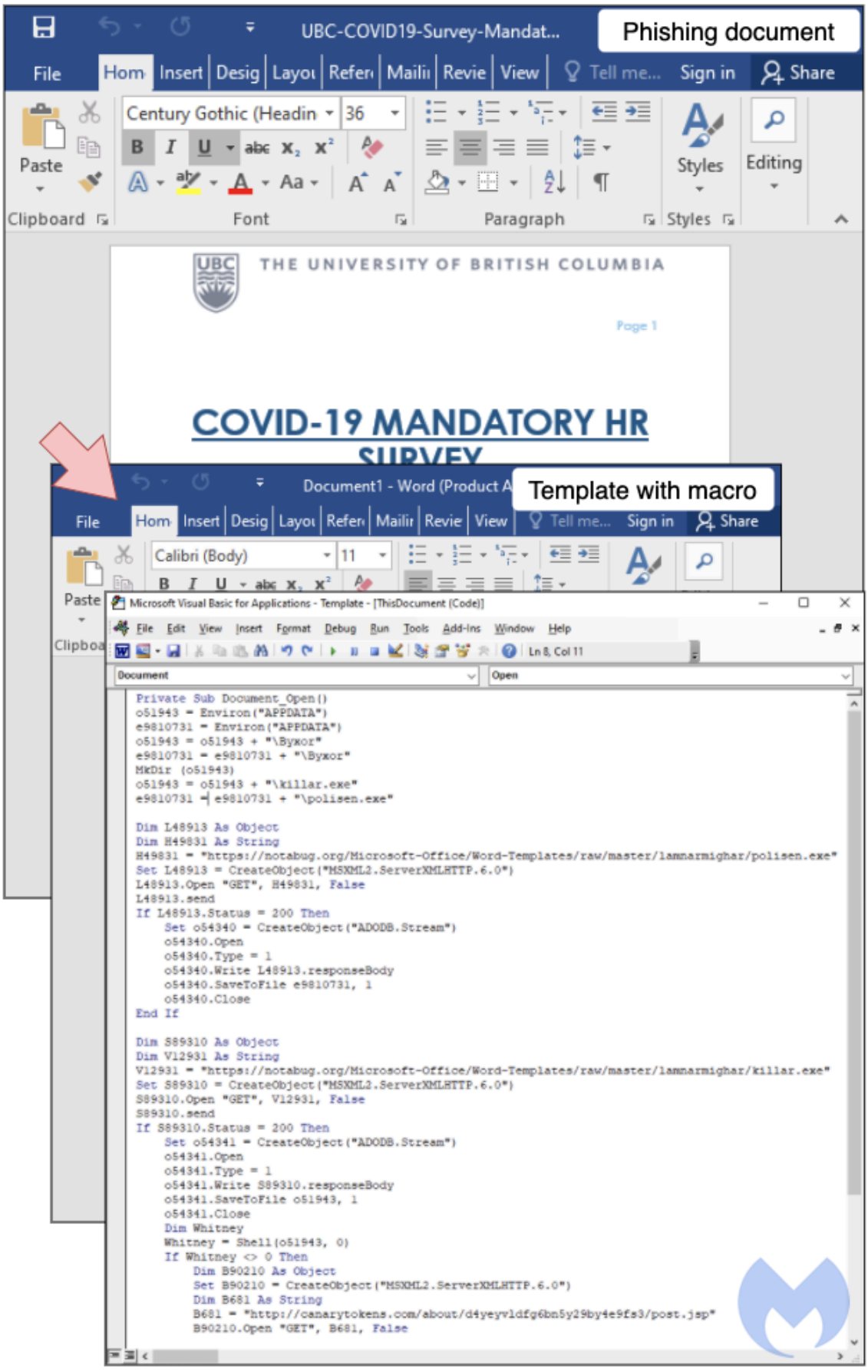
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

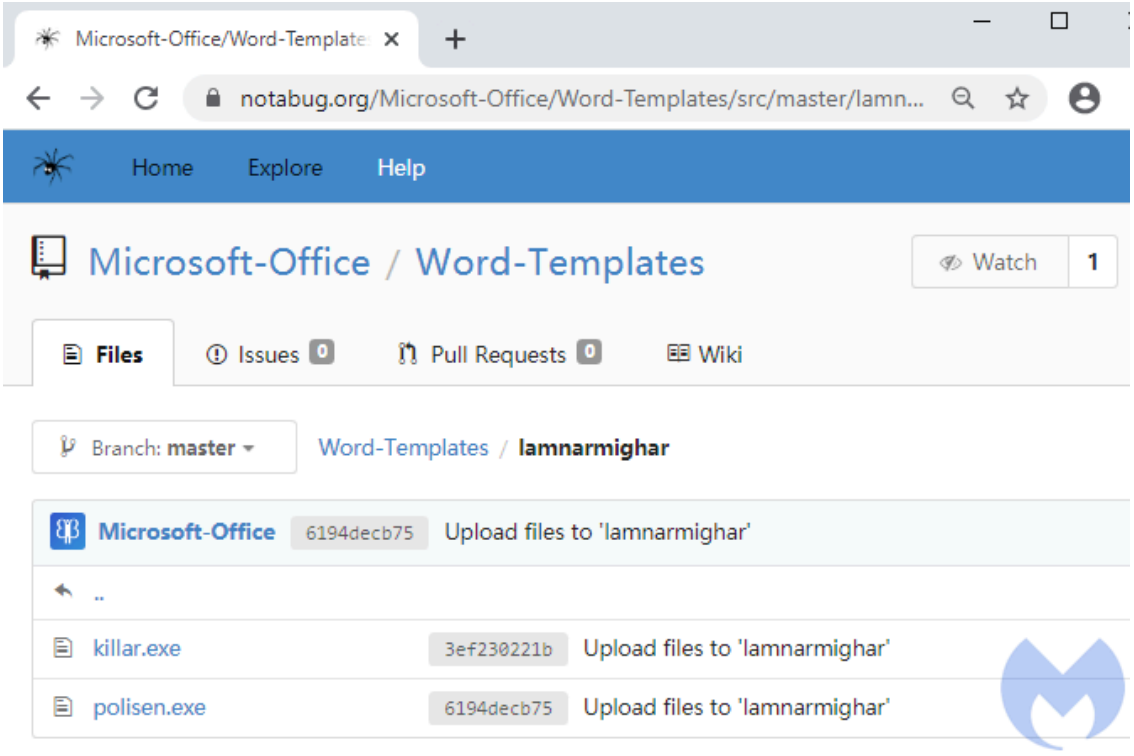
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



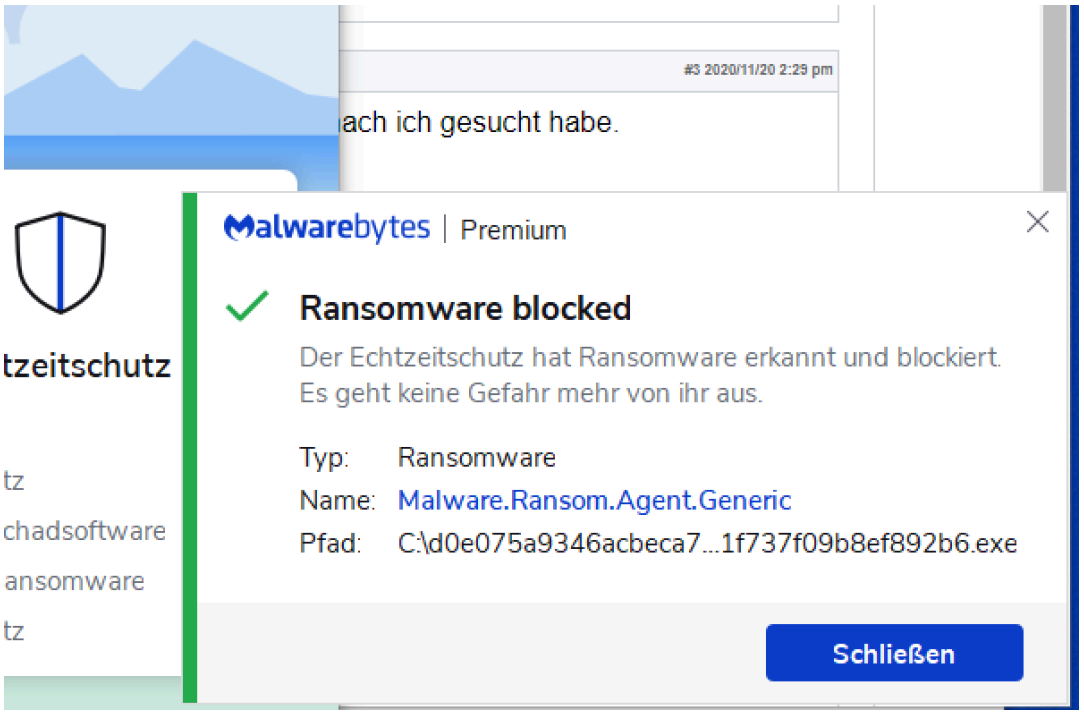
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

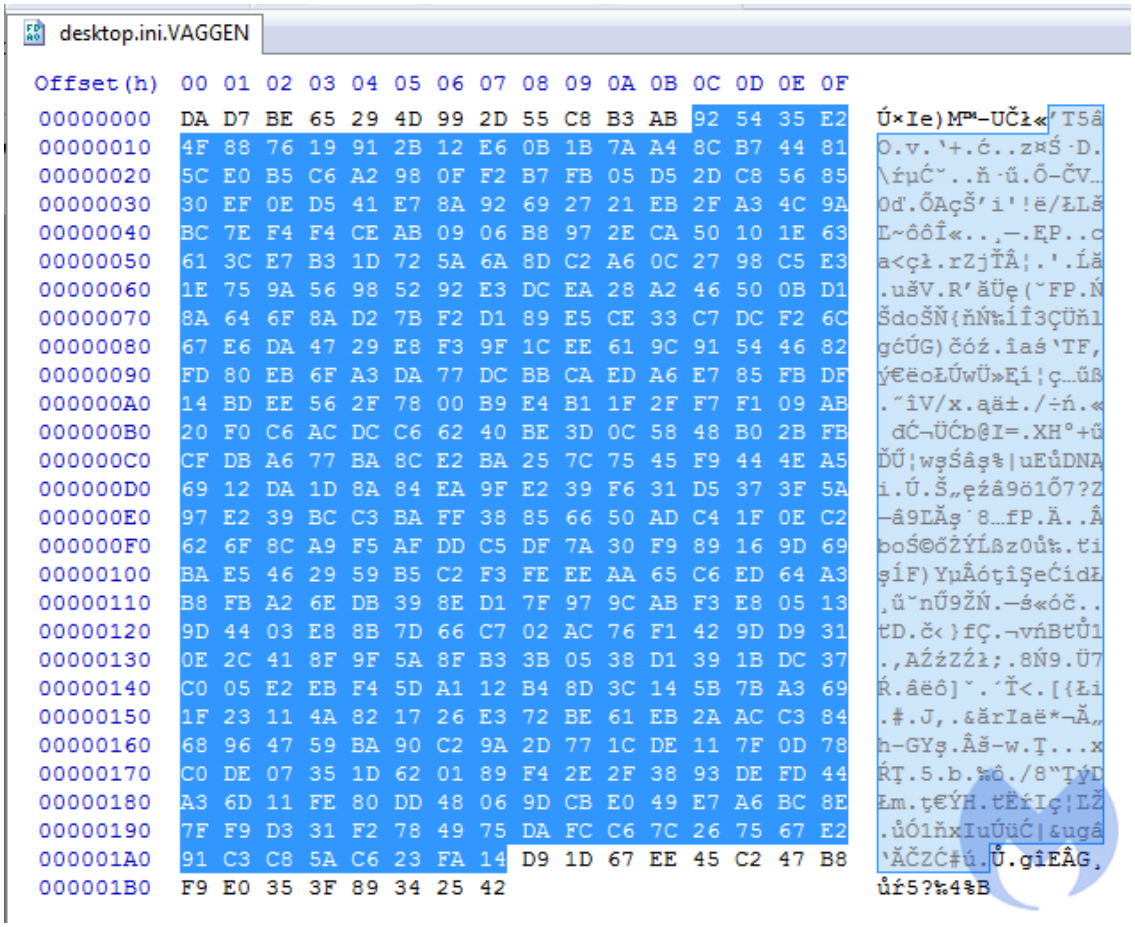
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



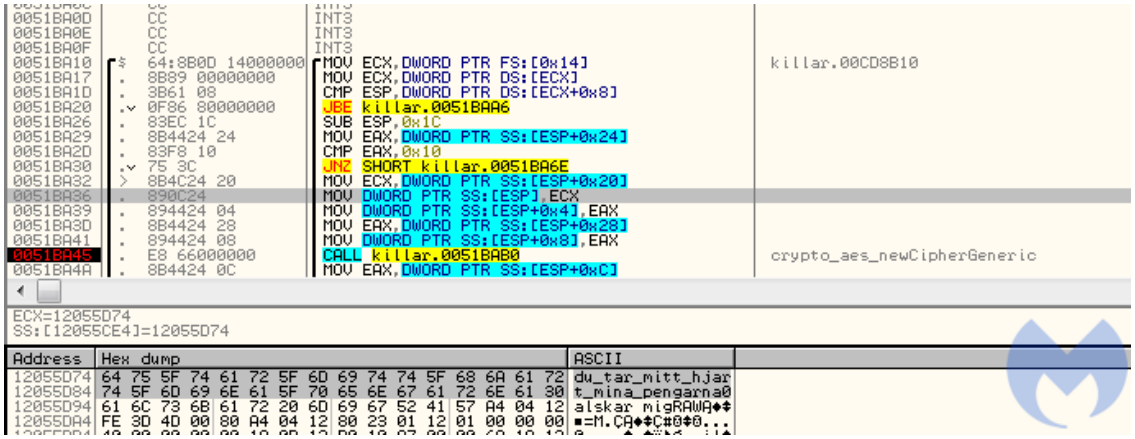
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

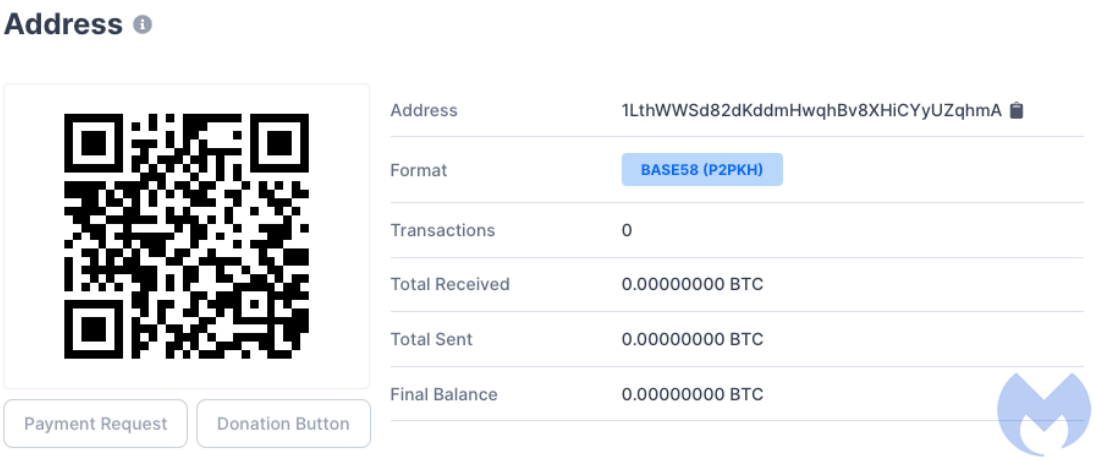
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

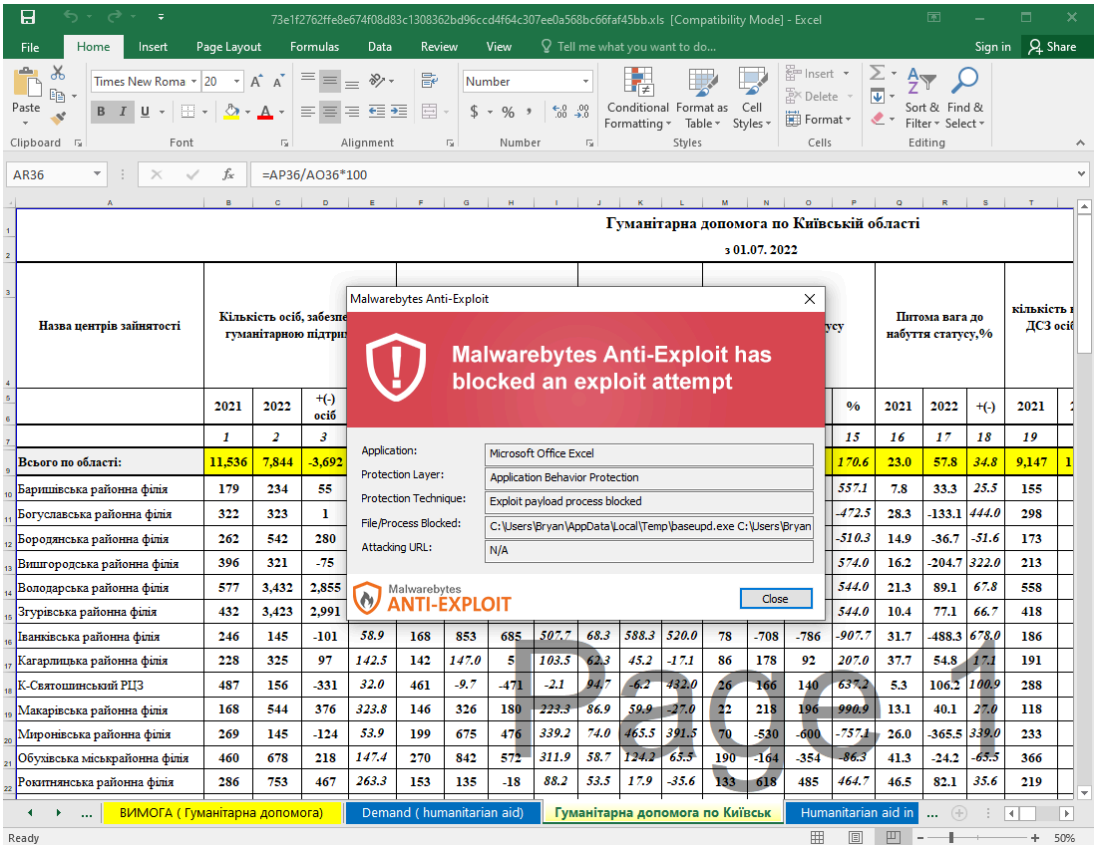
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

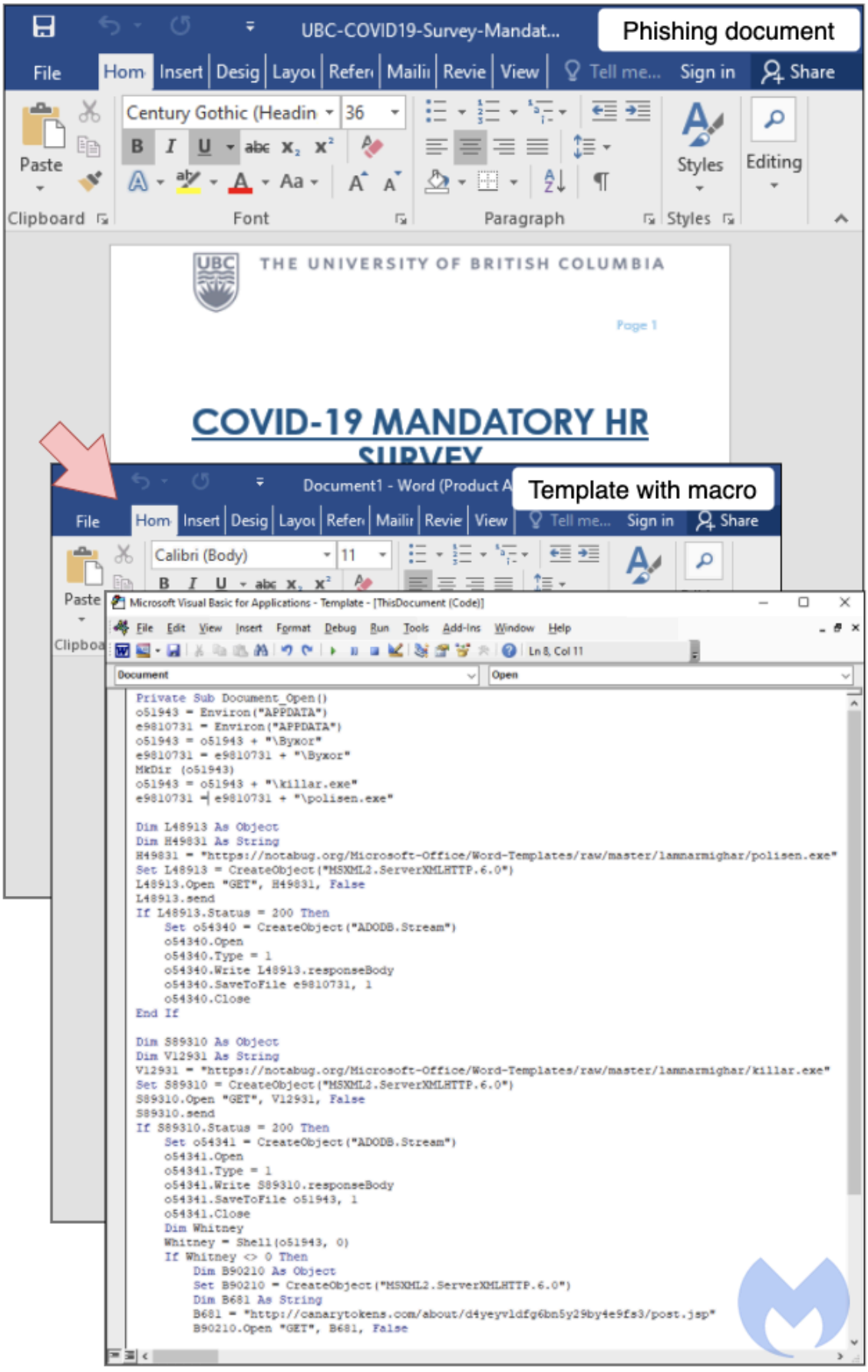
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

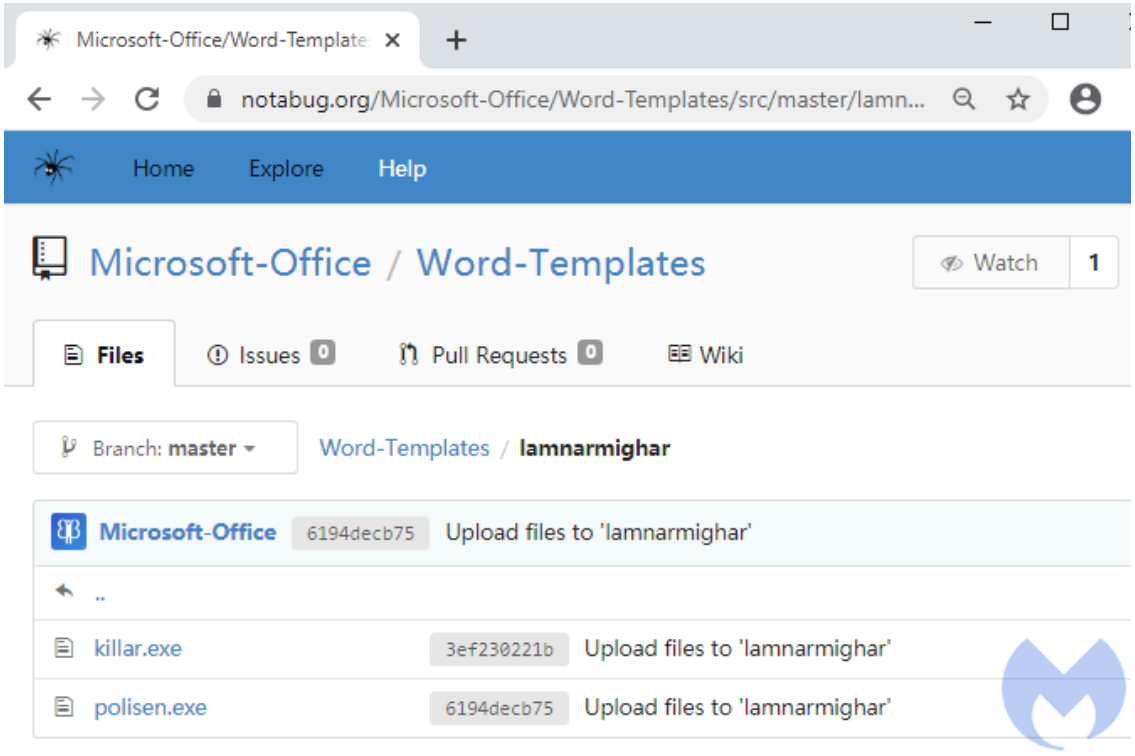
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



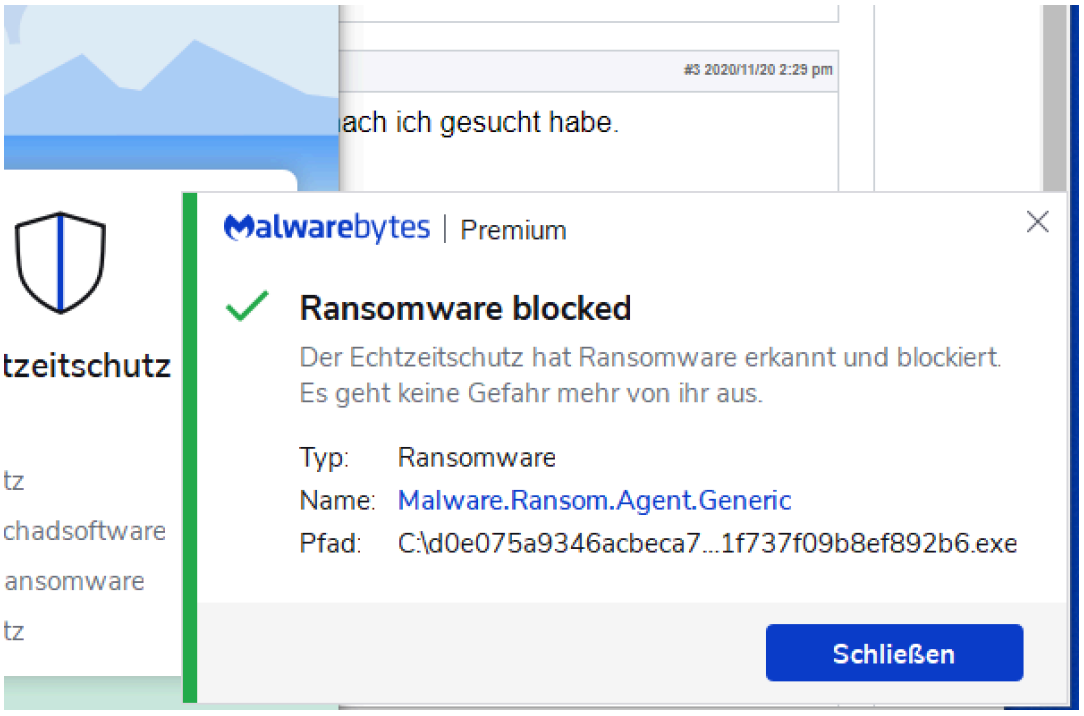
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

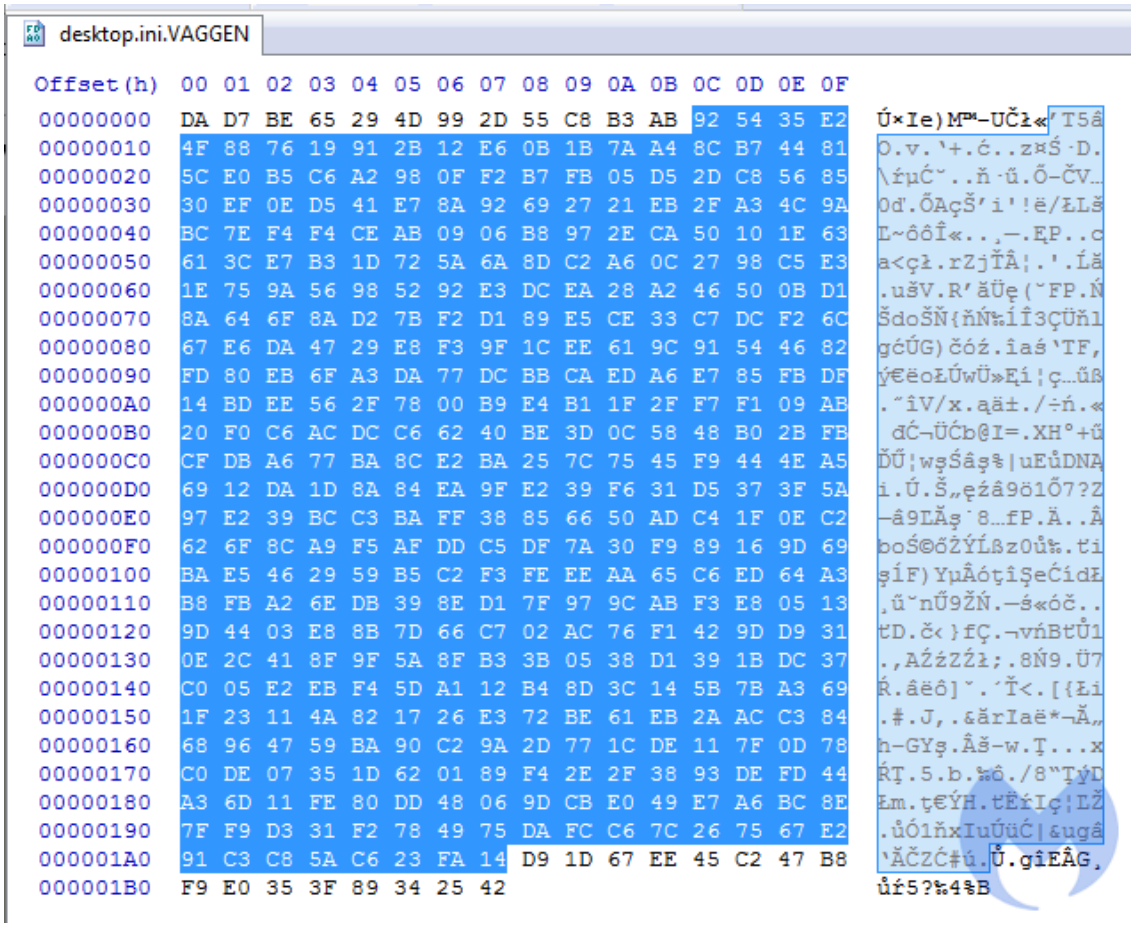
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



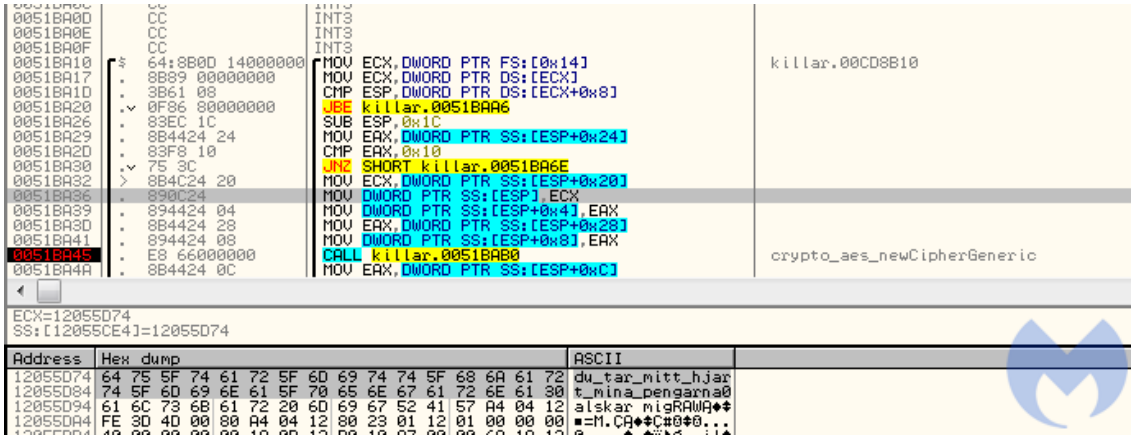
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

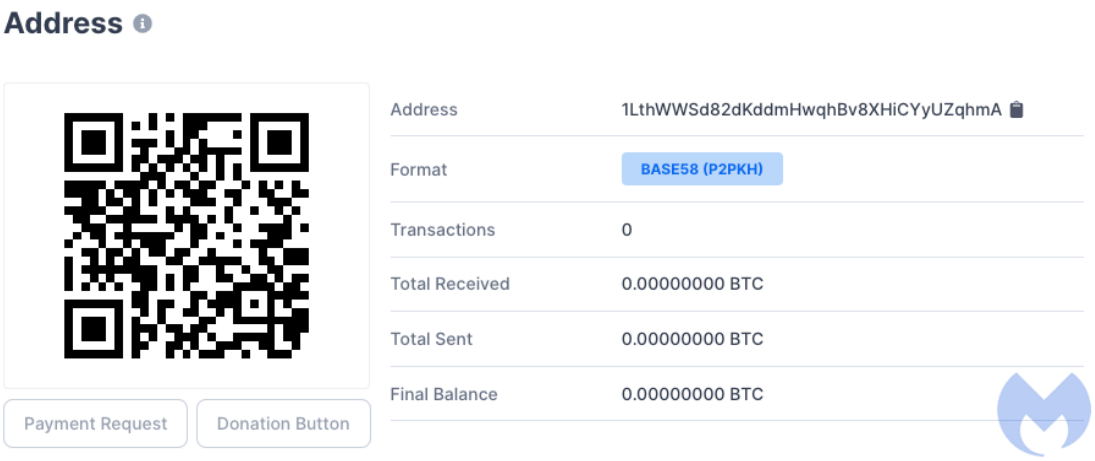
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

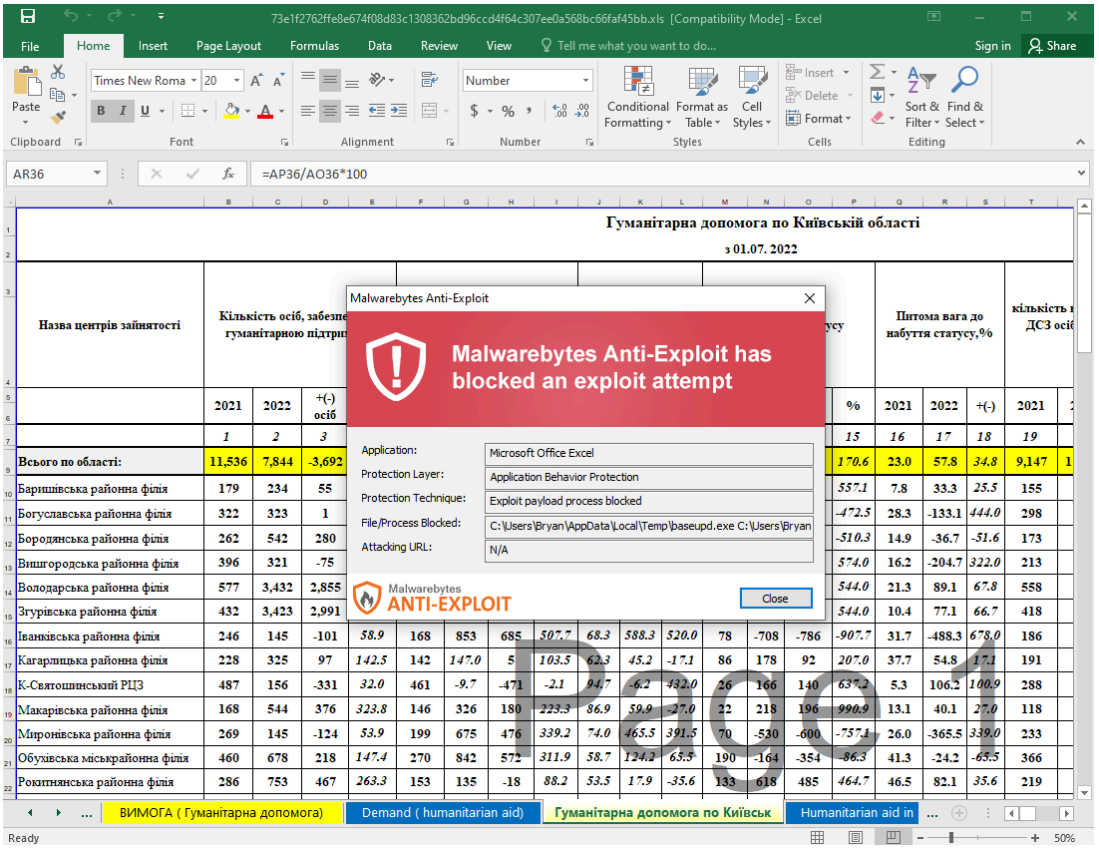
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

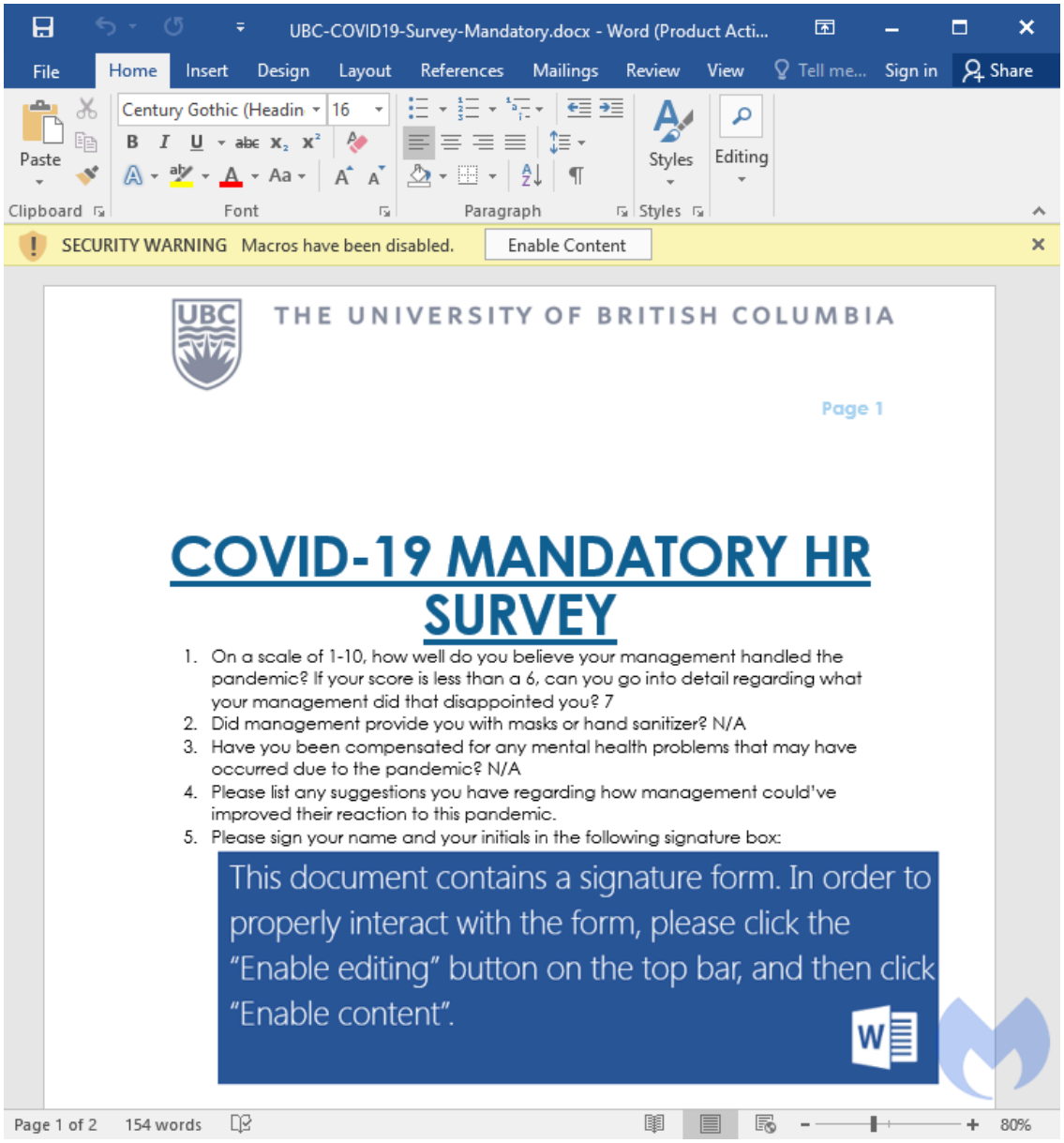
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

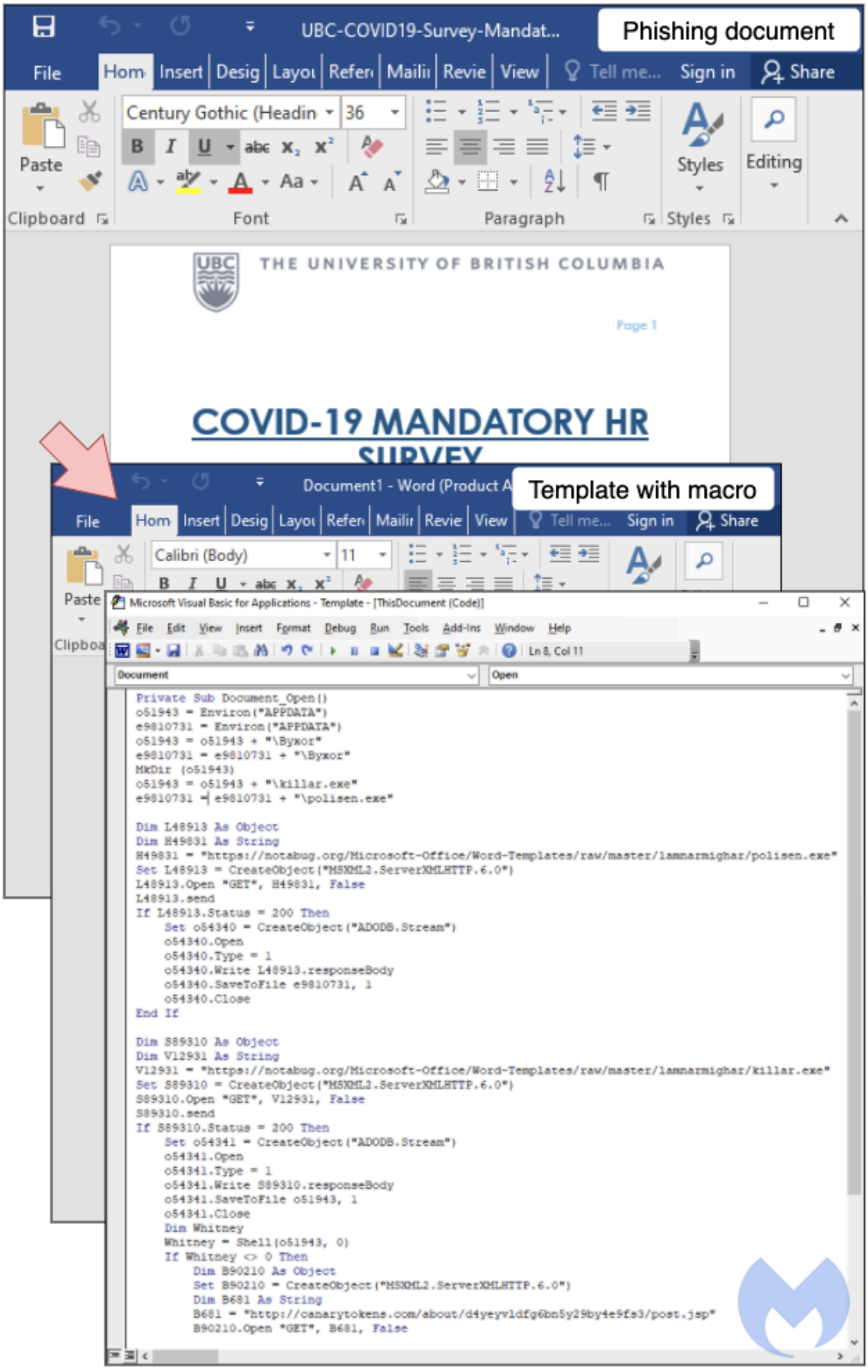
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

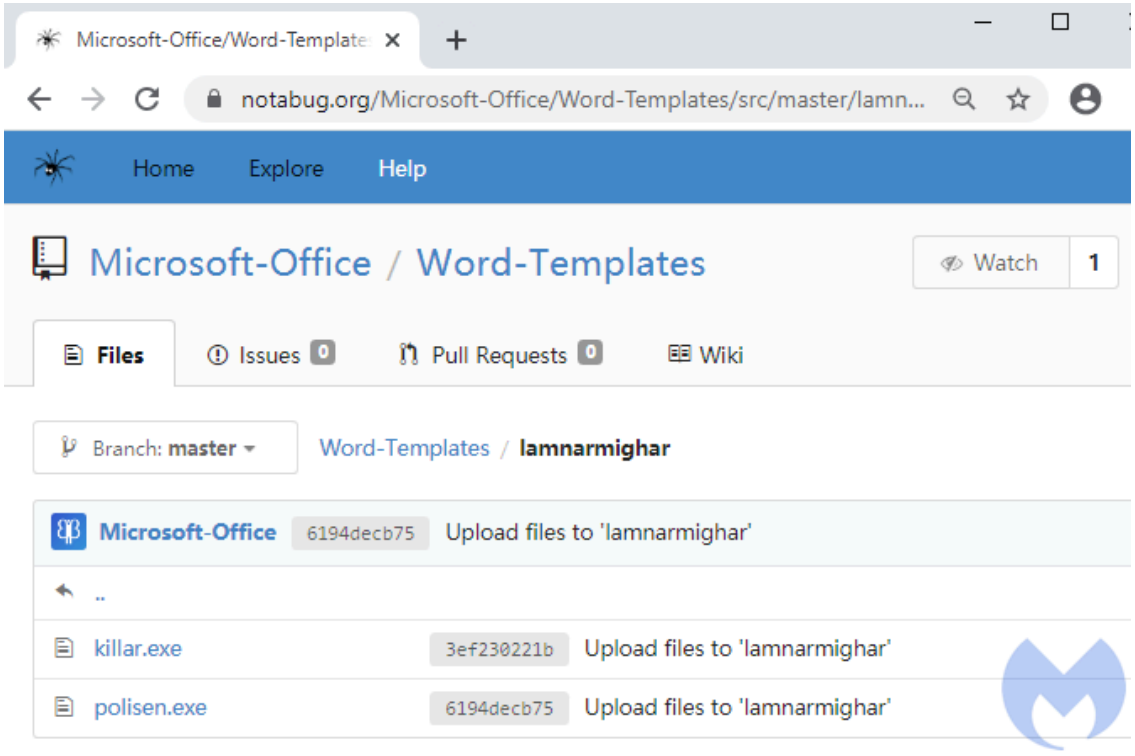
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



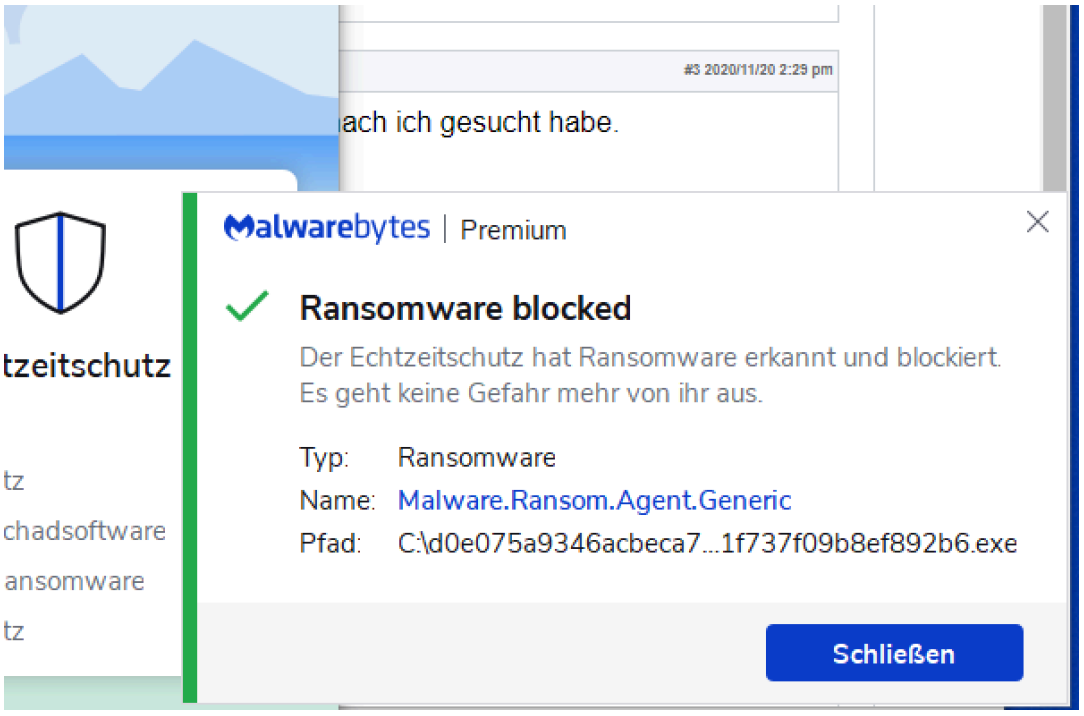
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

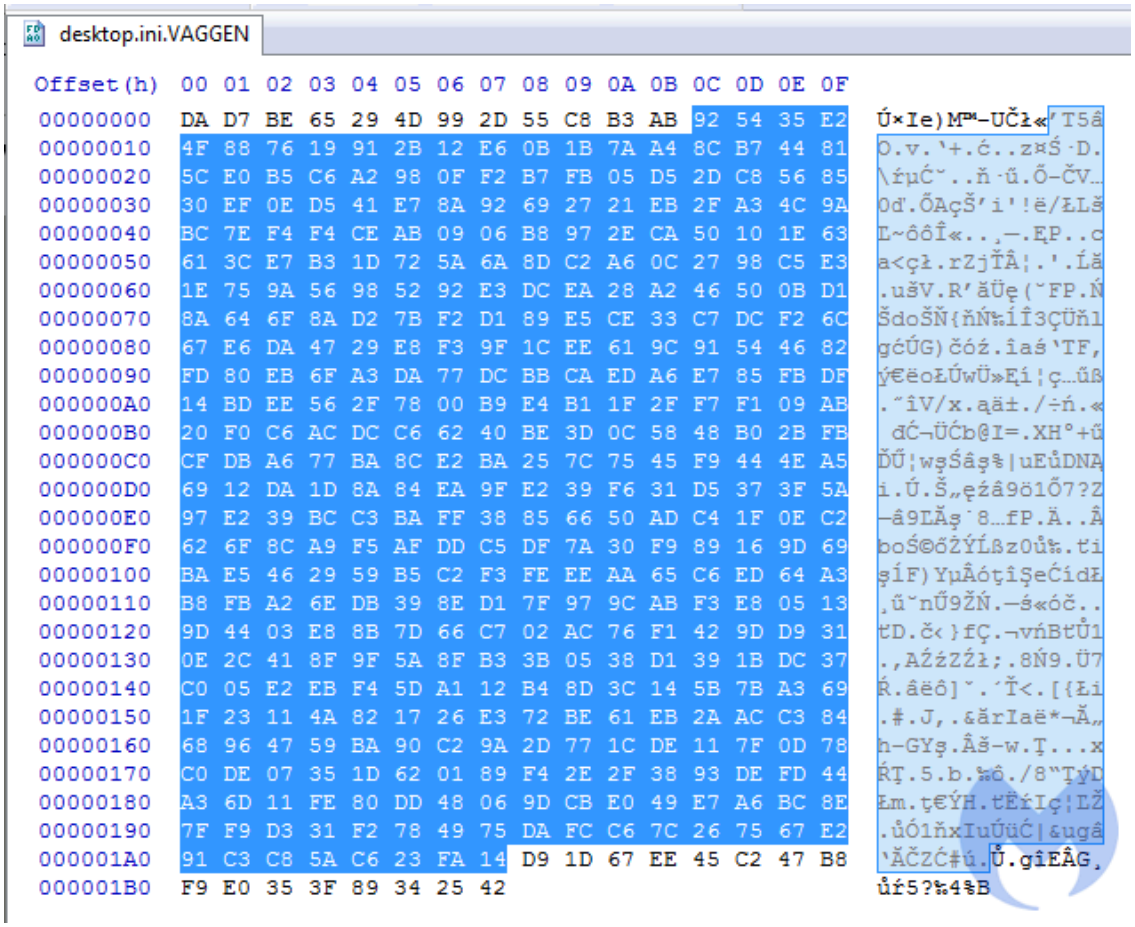
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

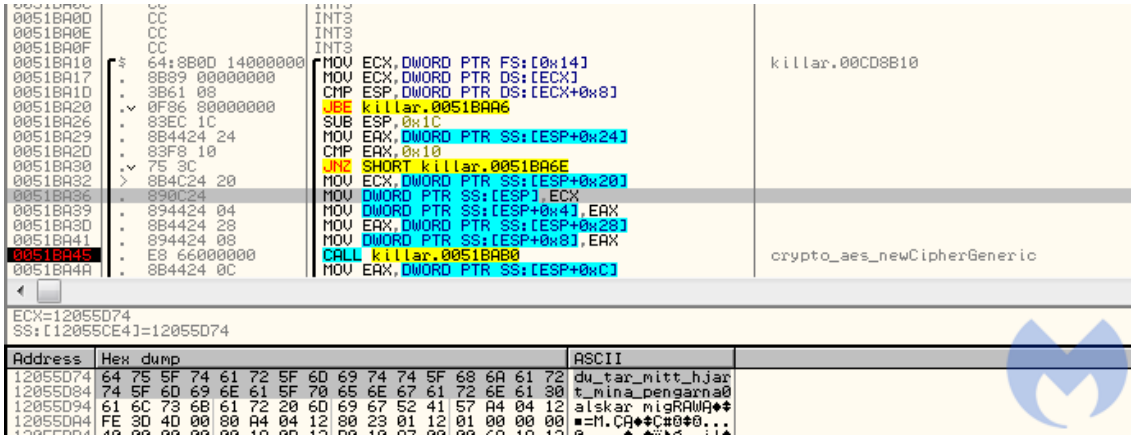


The content of the output file (with .VAGGEN extension) contains:

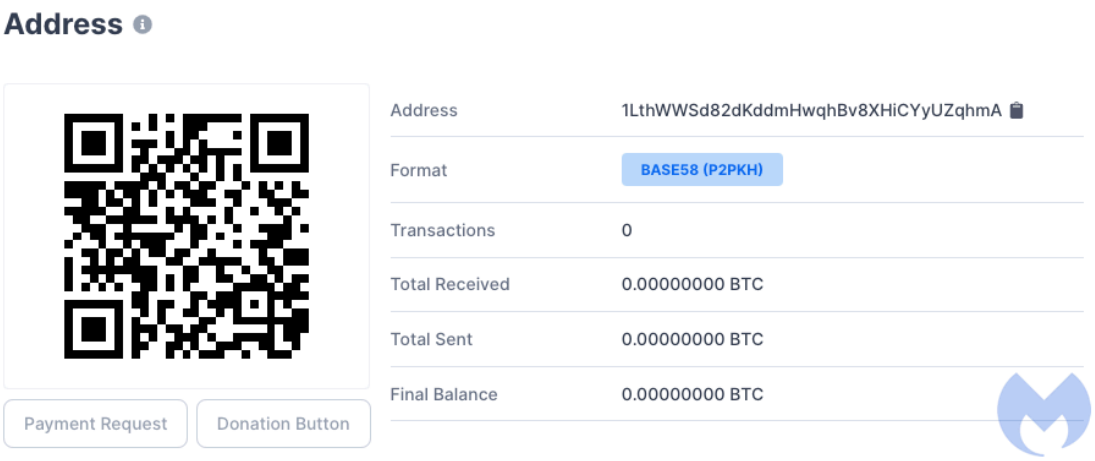
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

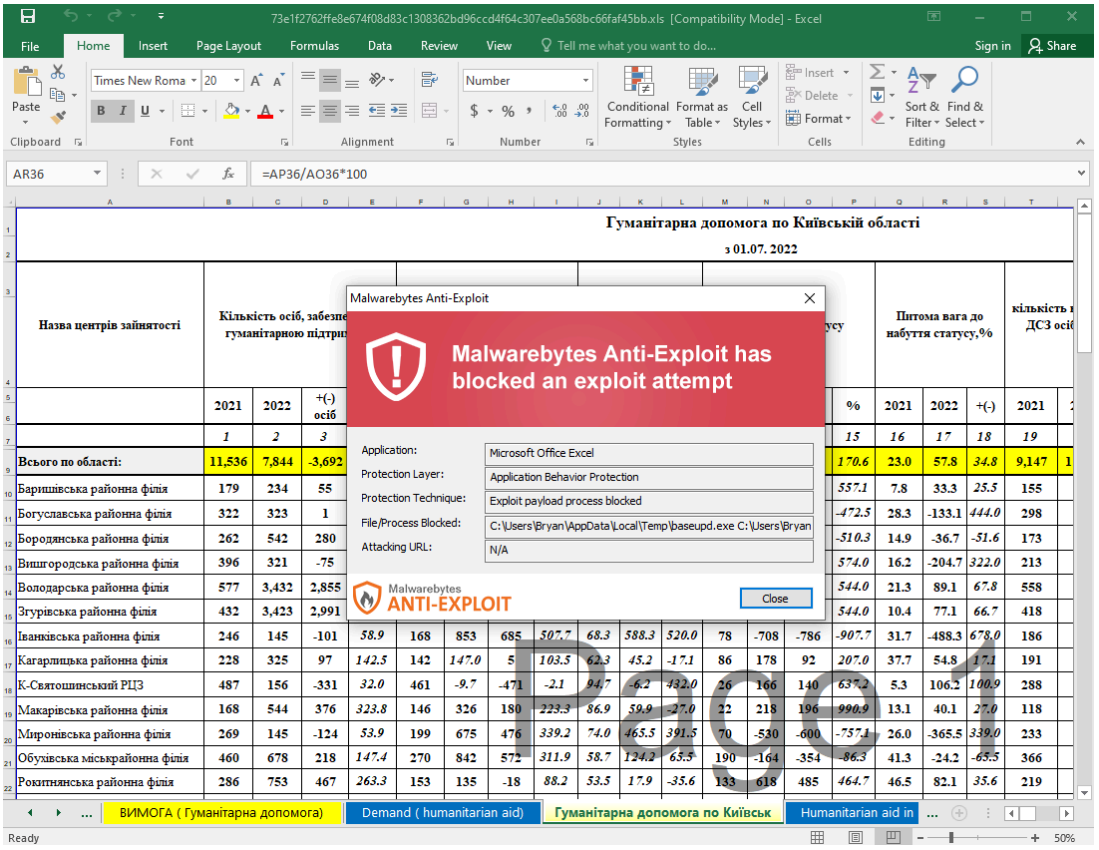
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

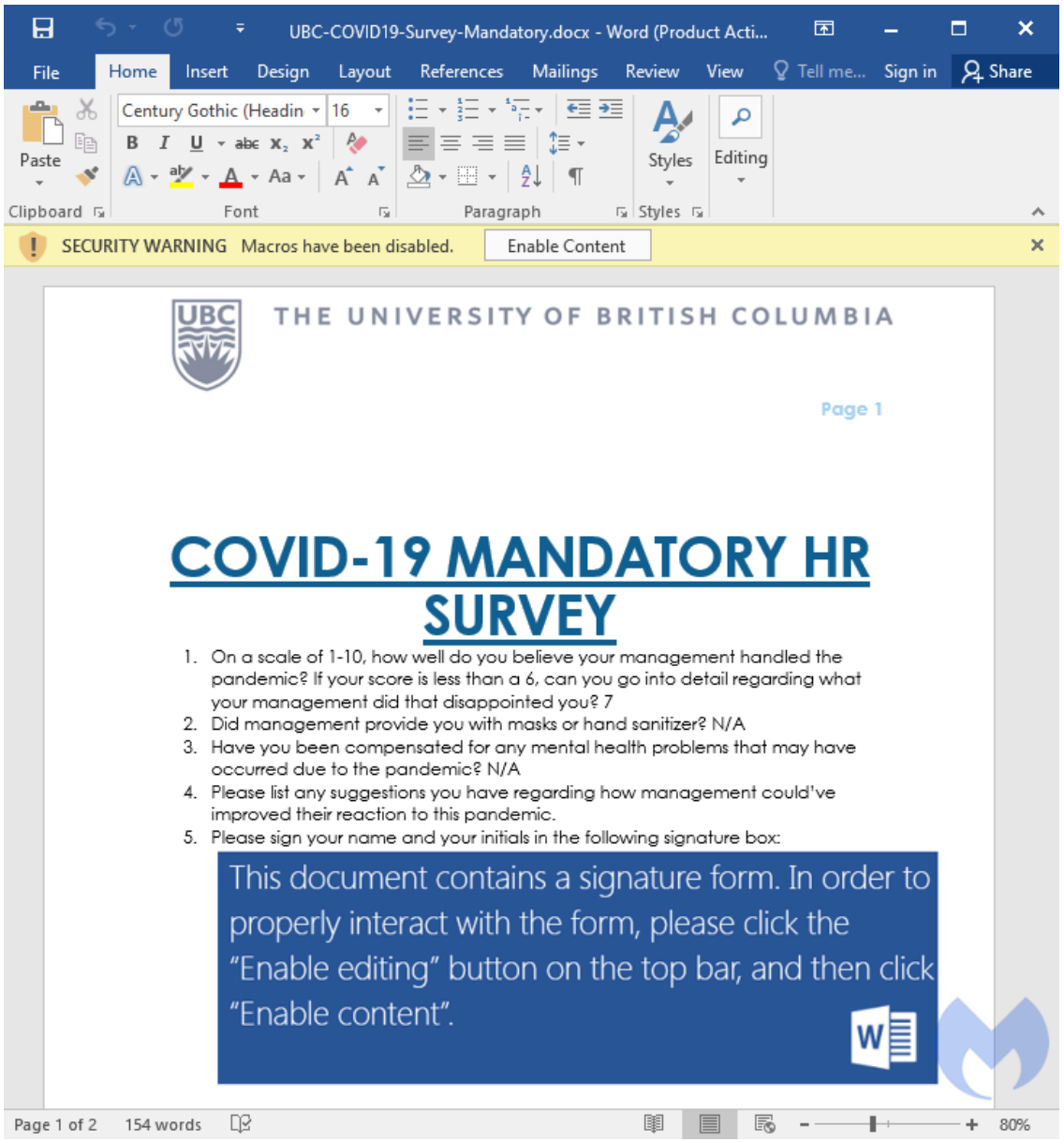
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

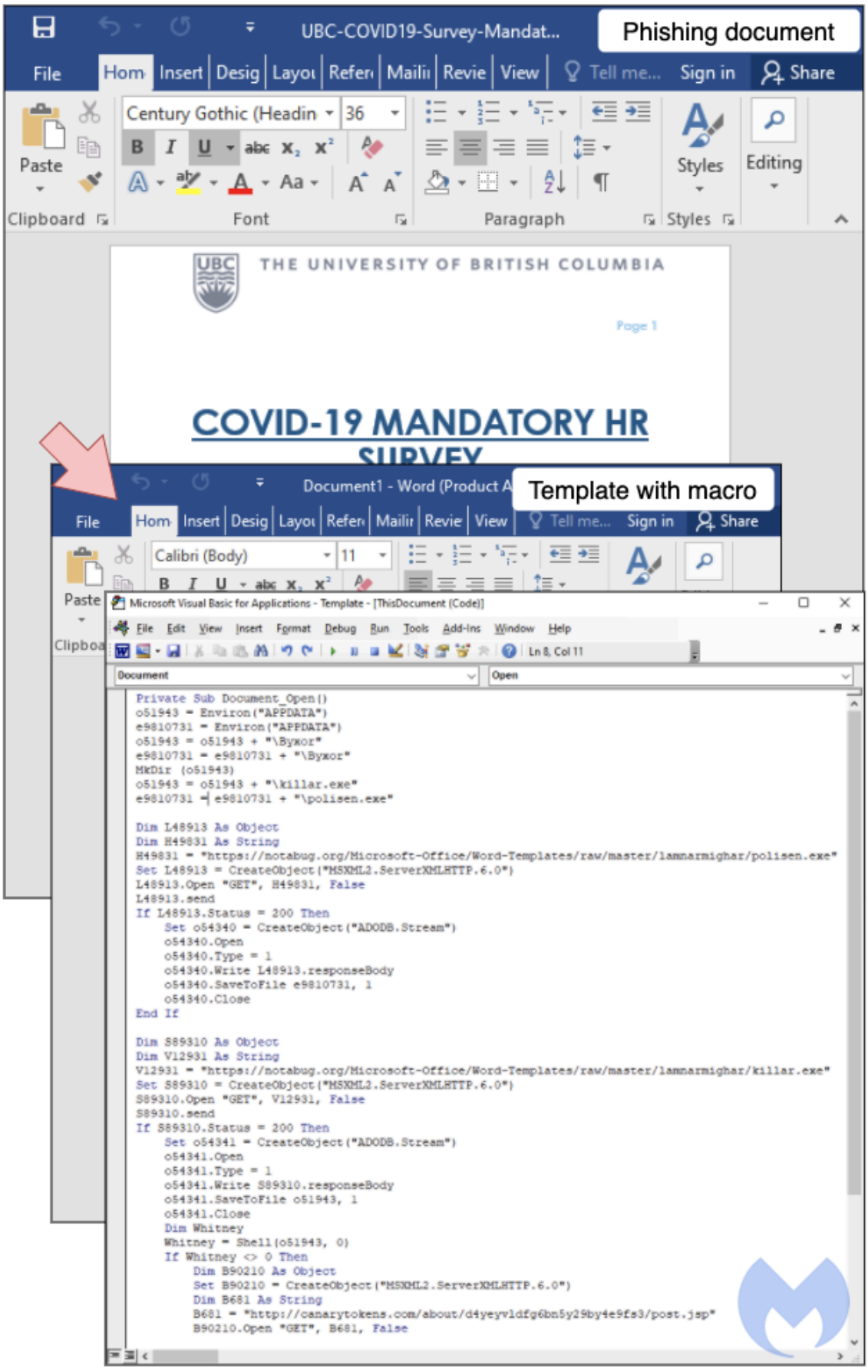
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

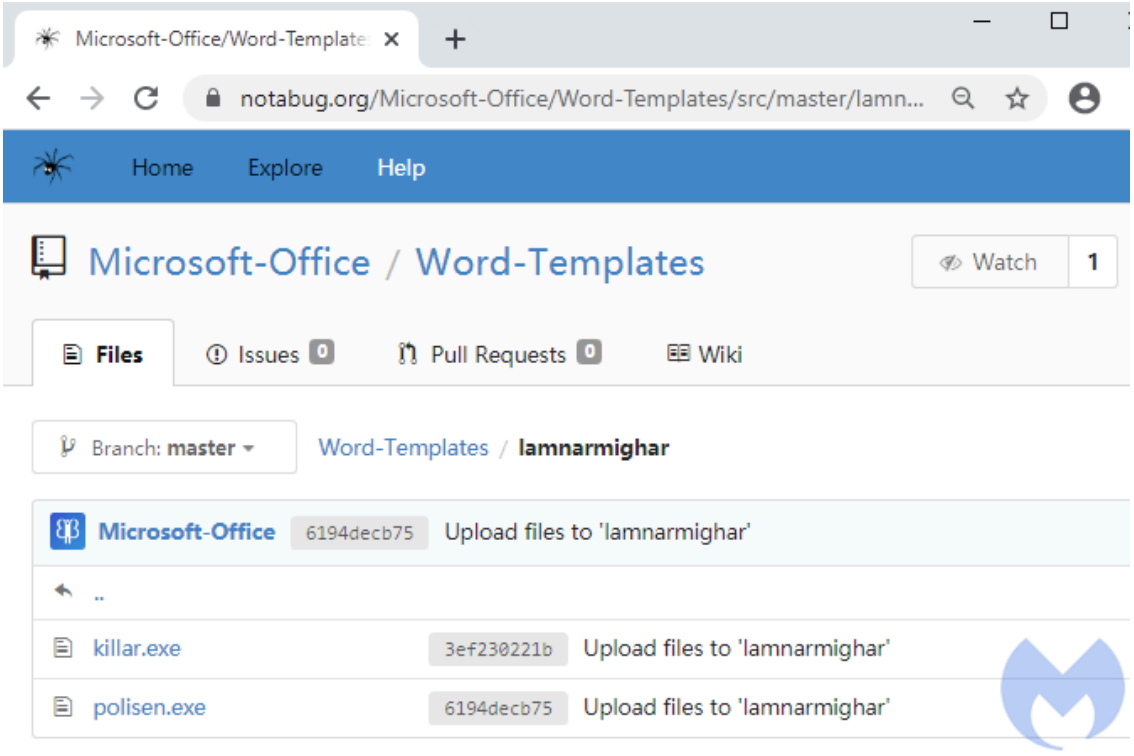
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



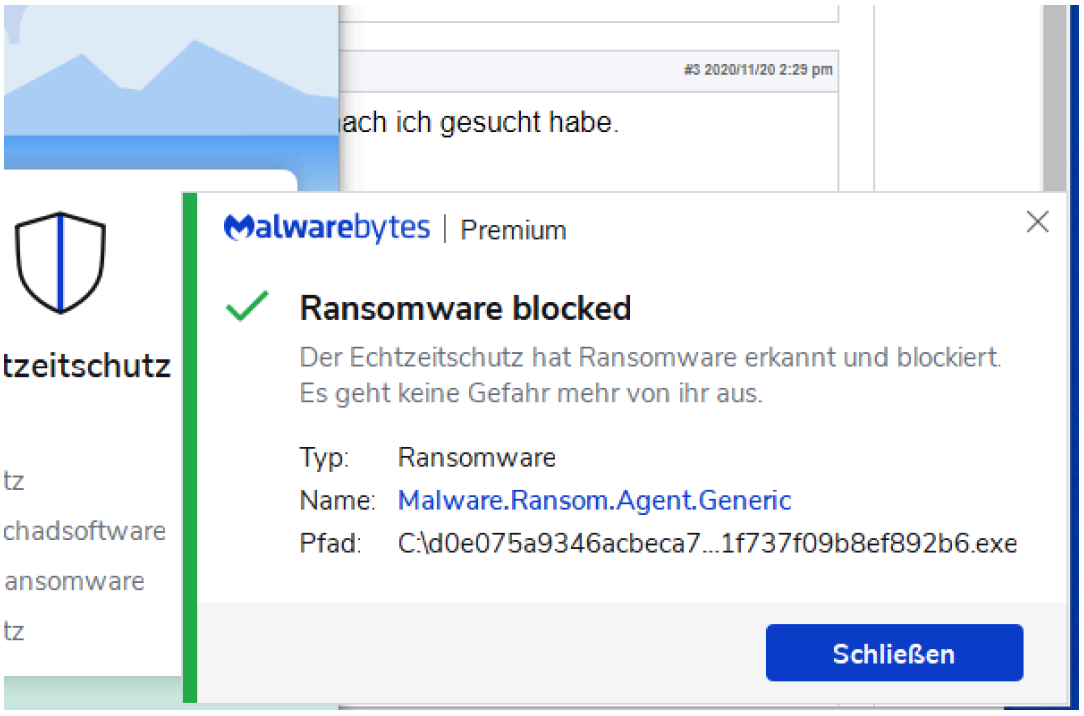
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

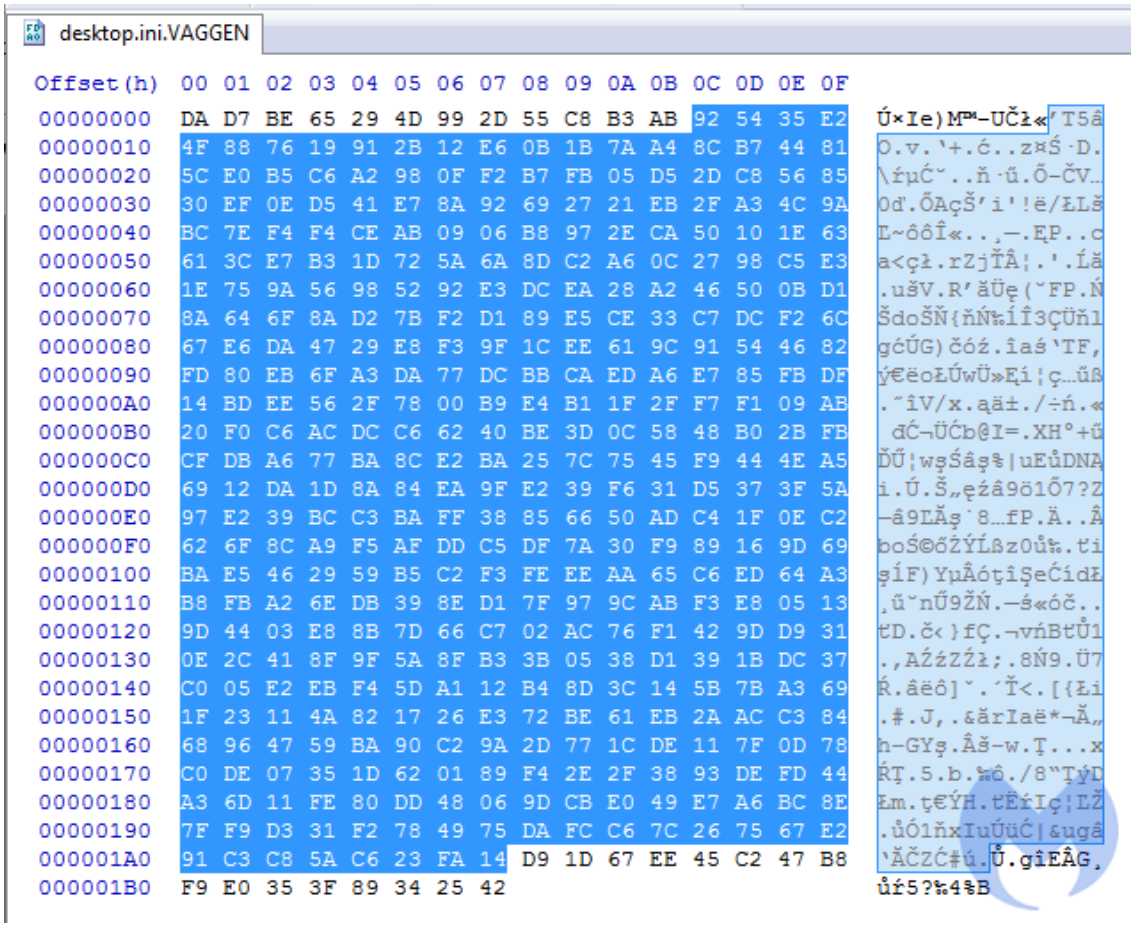
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



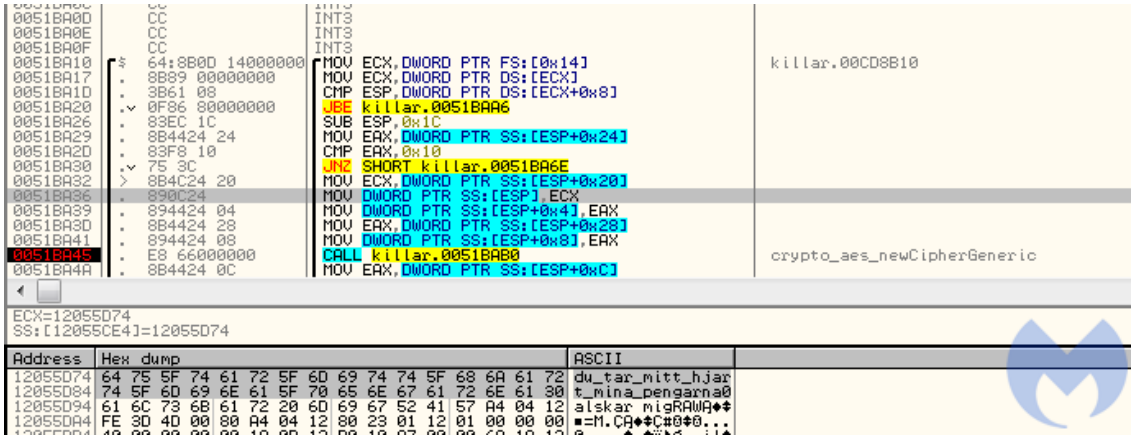
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

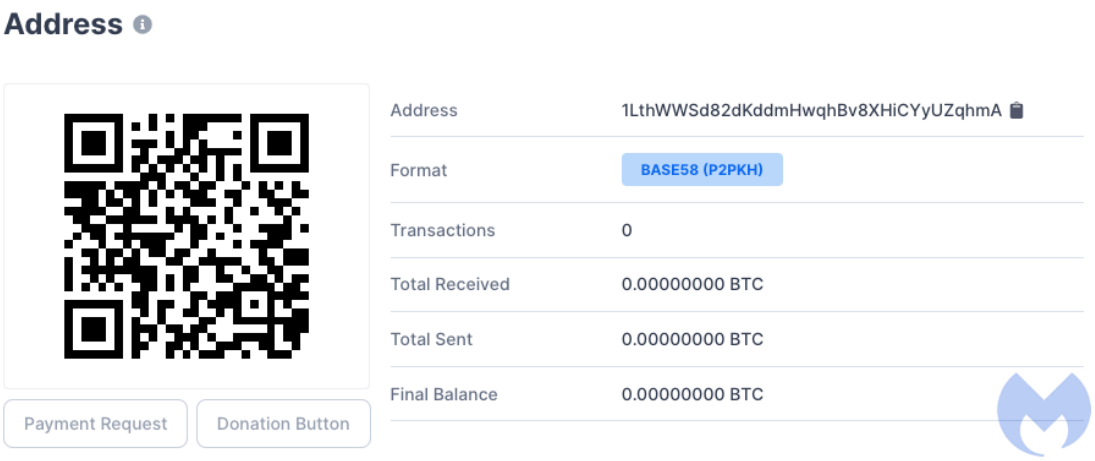
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

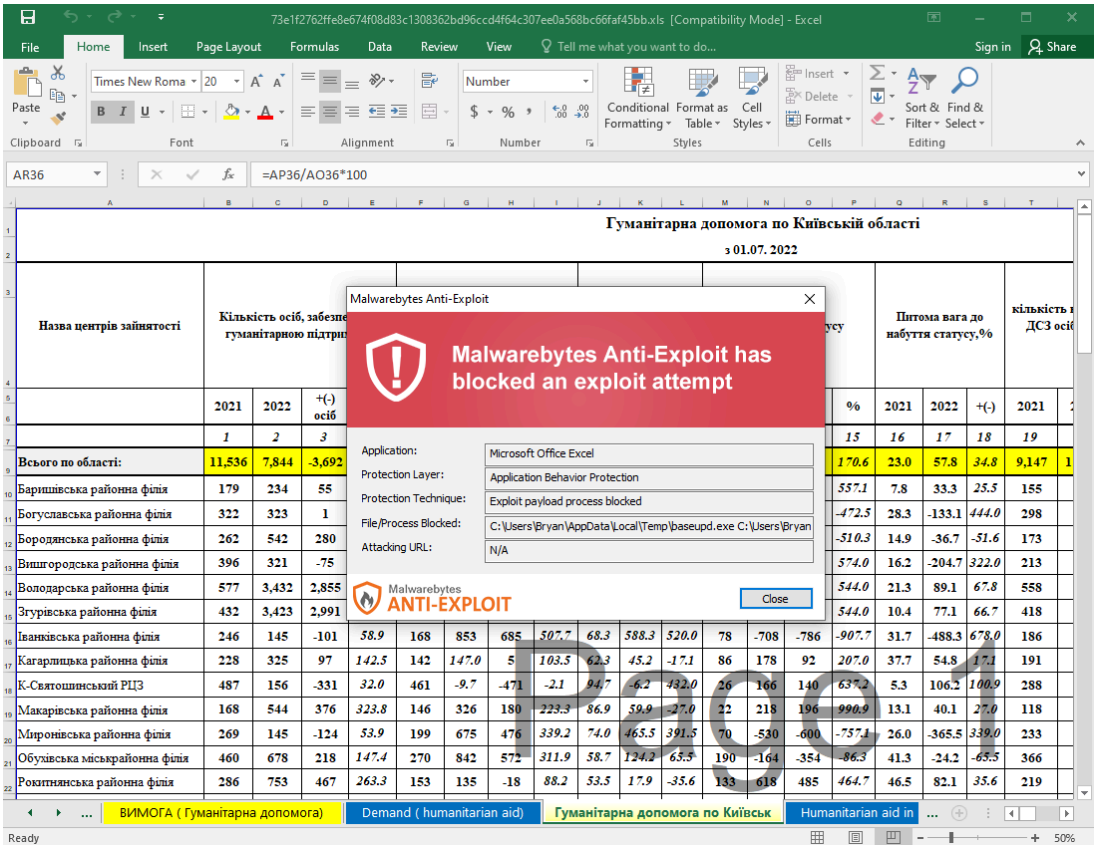
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

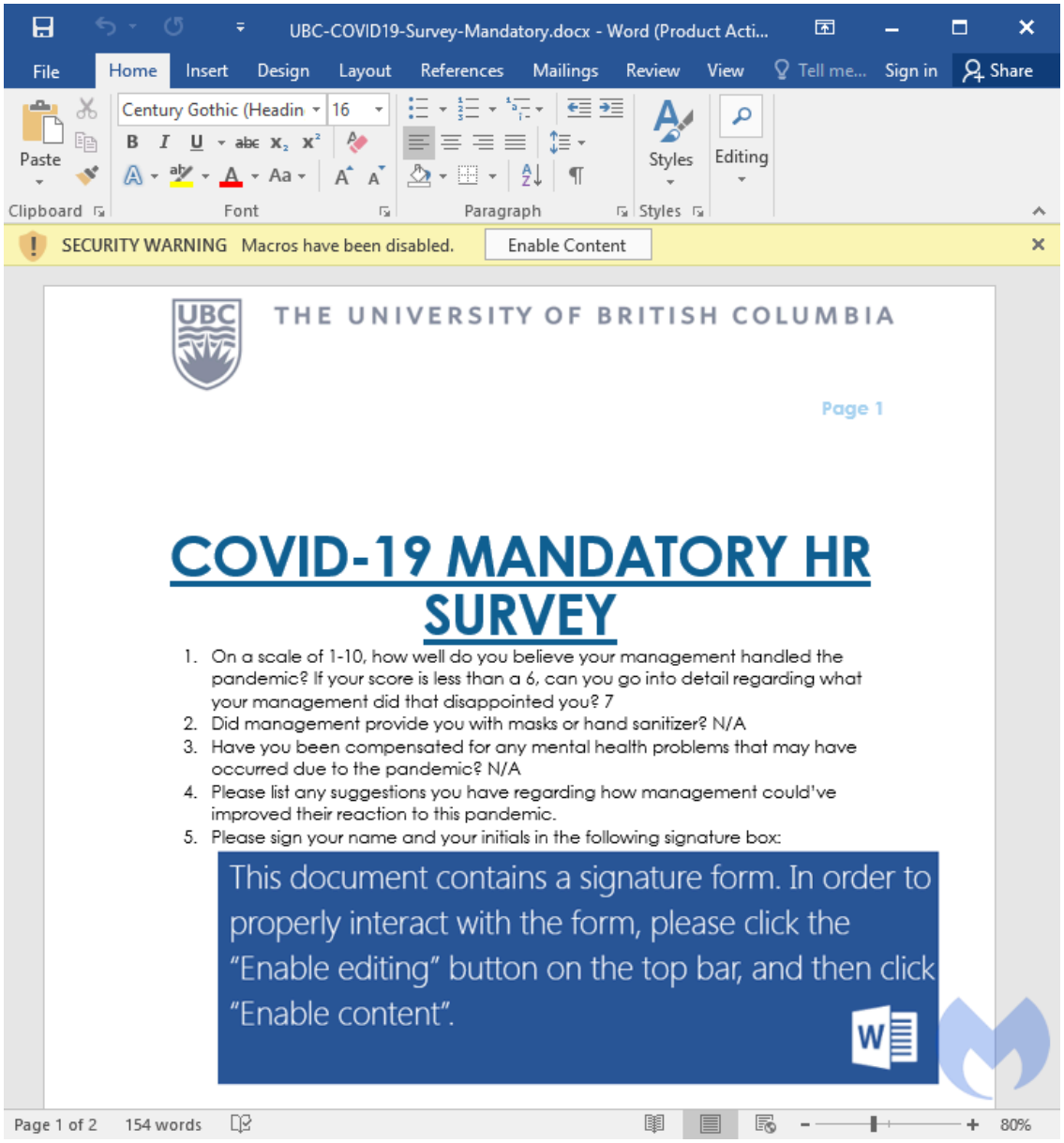
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

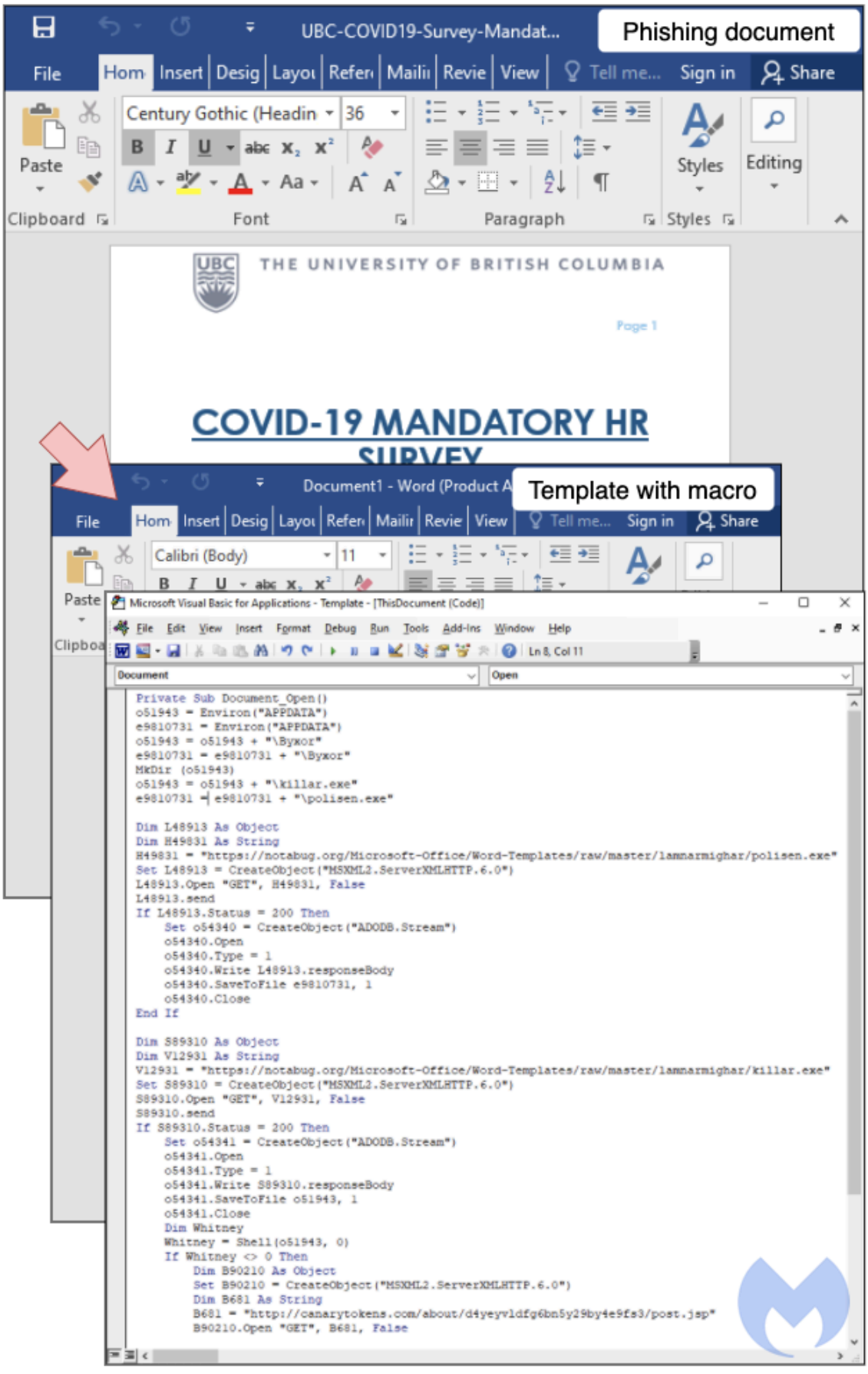
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

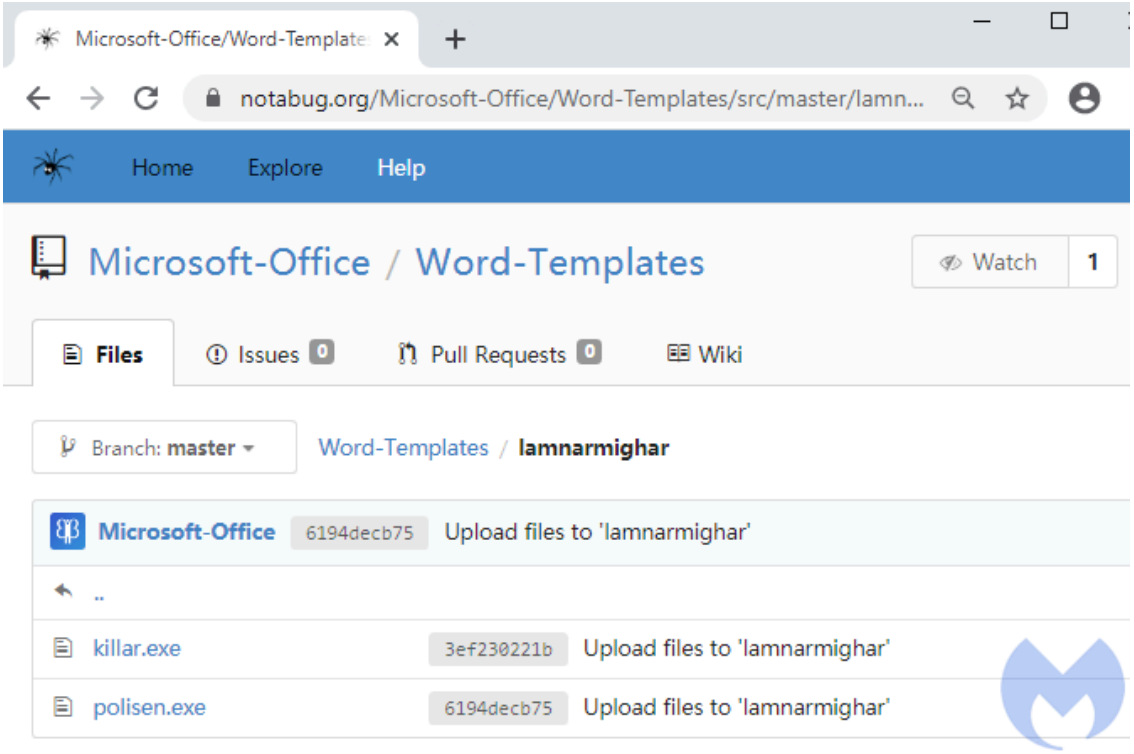
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



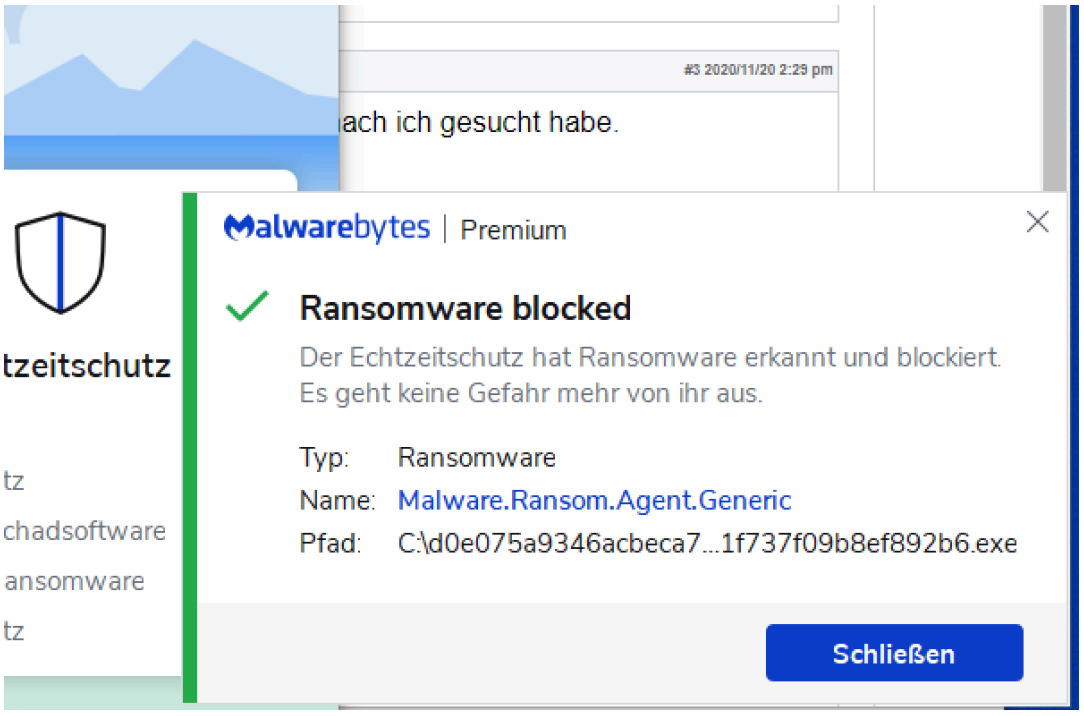
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

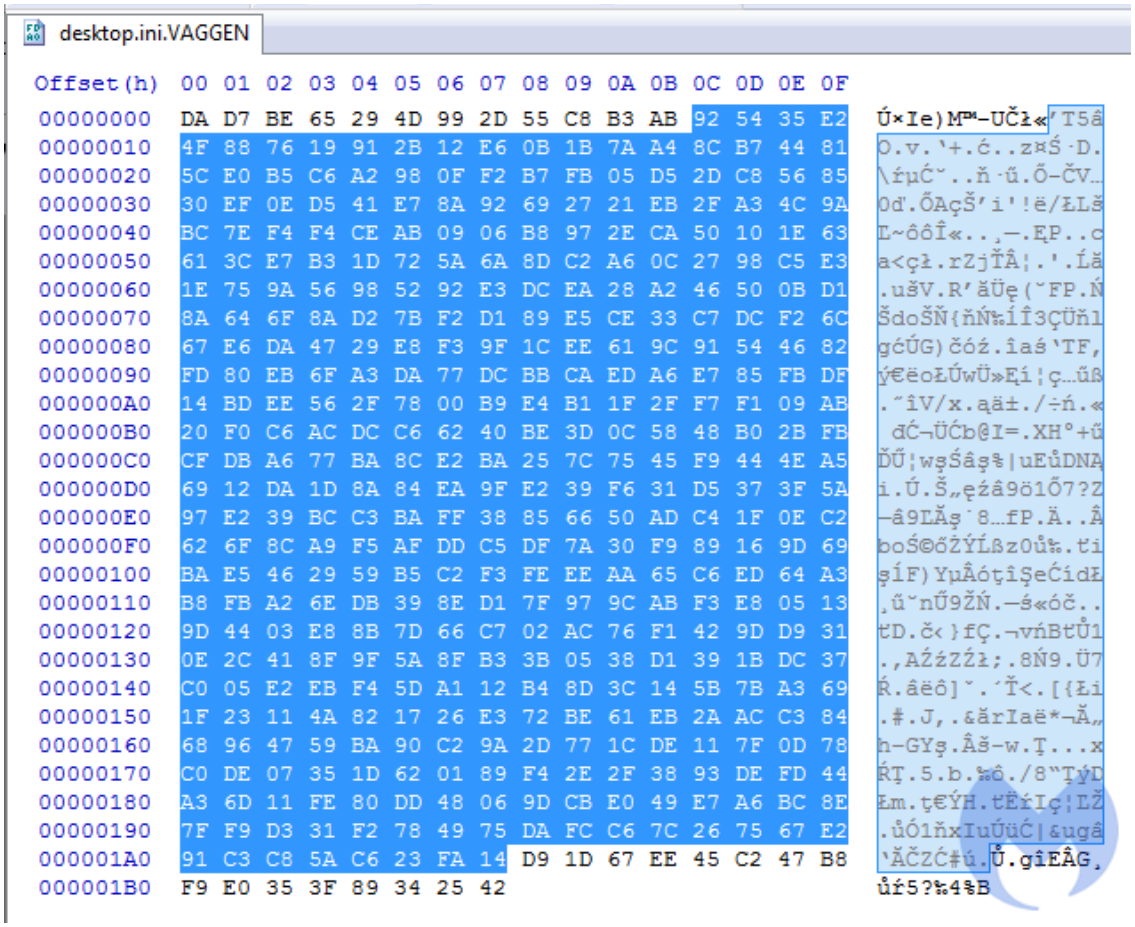
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

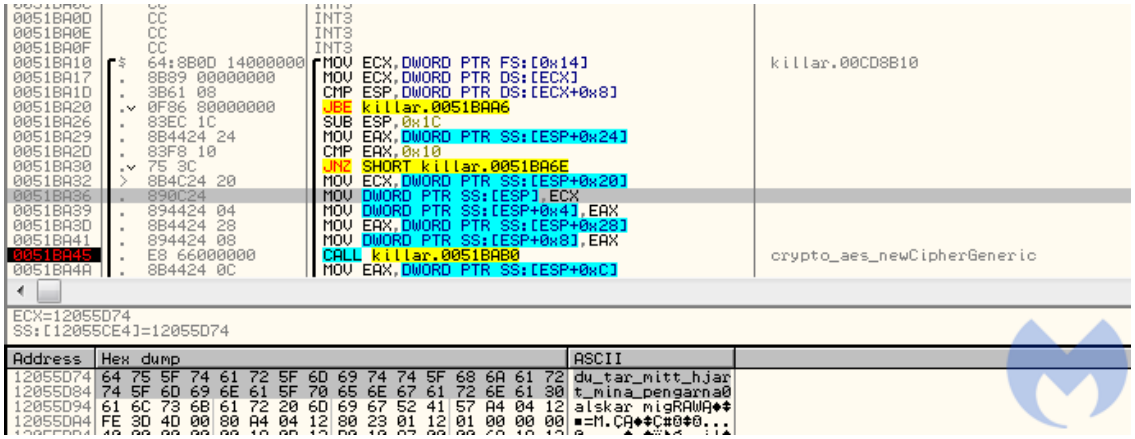


The content of the output file (with .VAGGEN extension) contains:

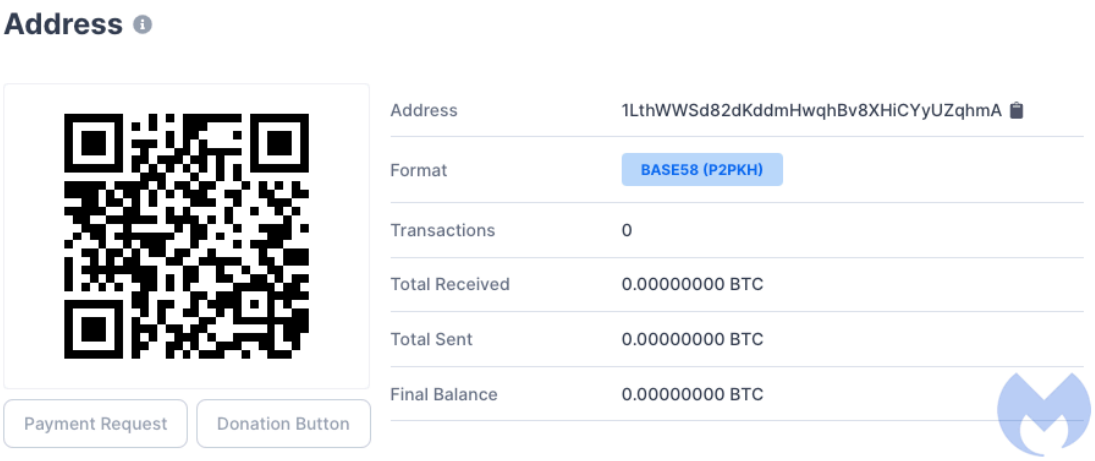
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

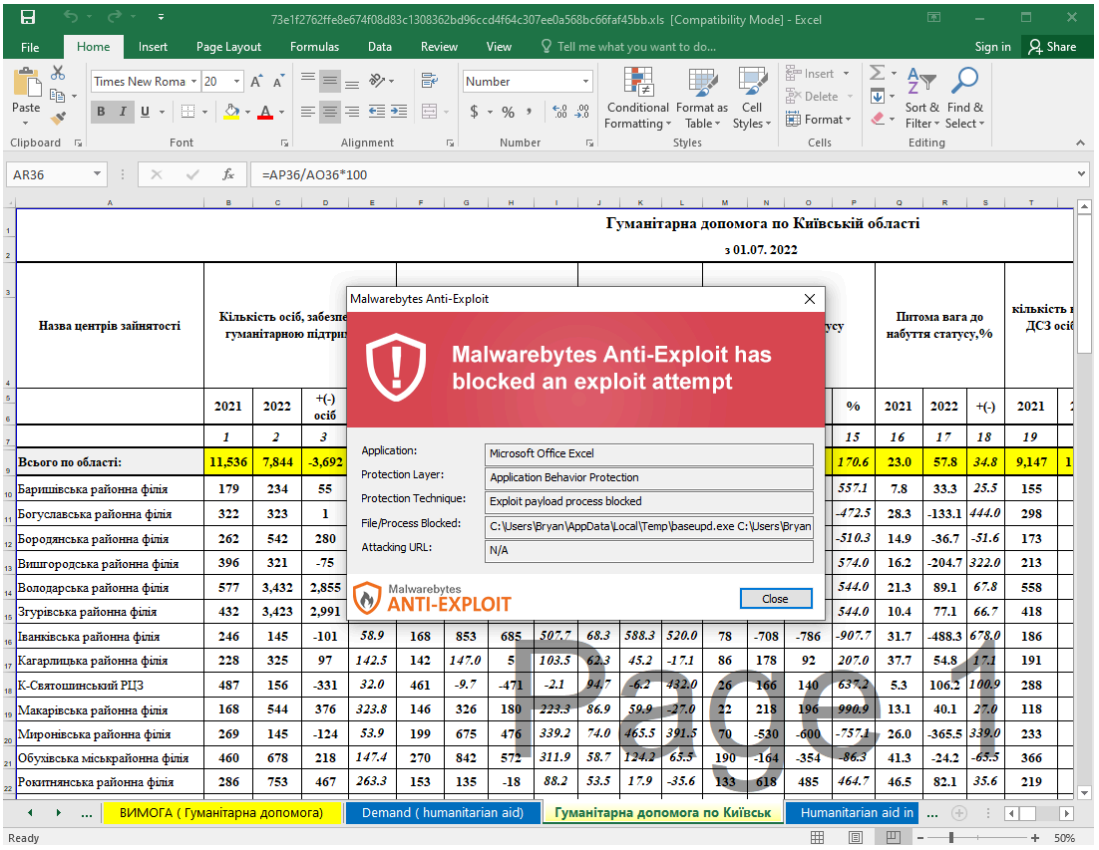
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

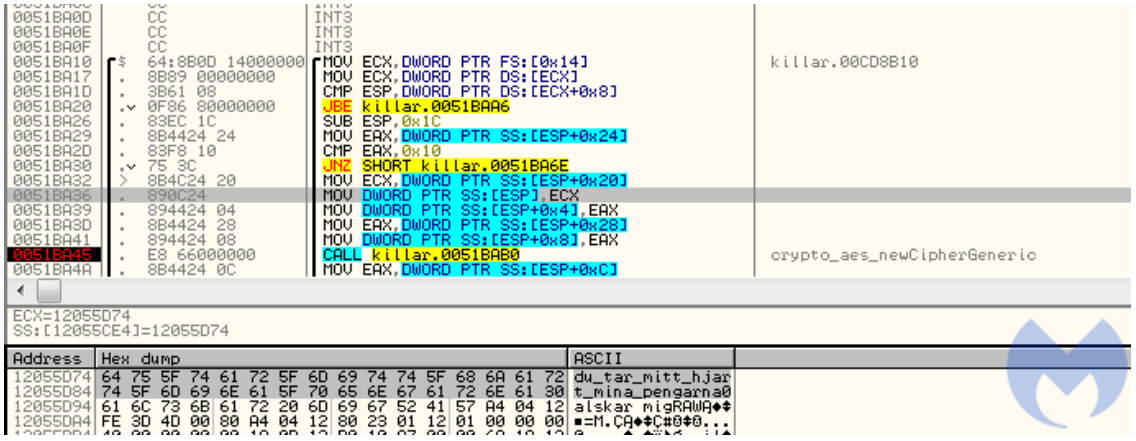
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

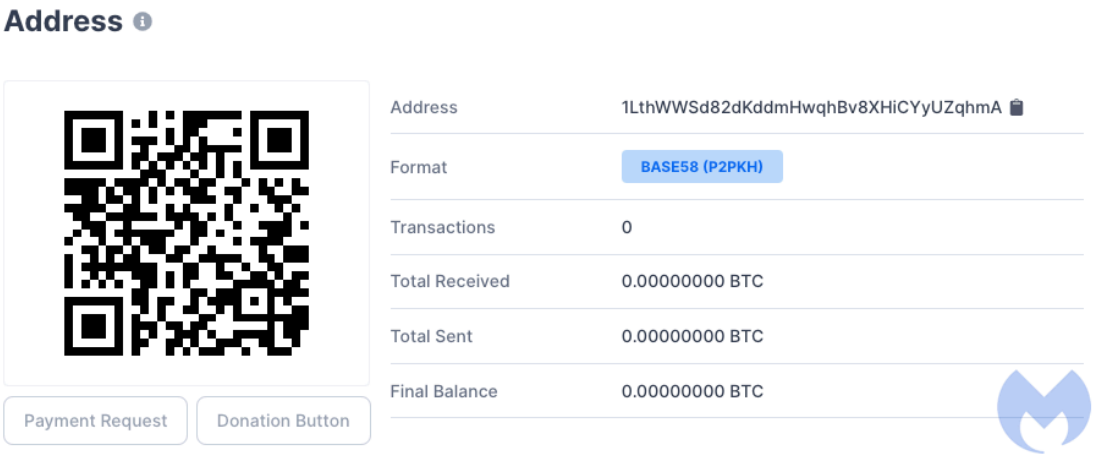
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

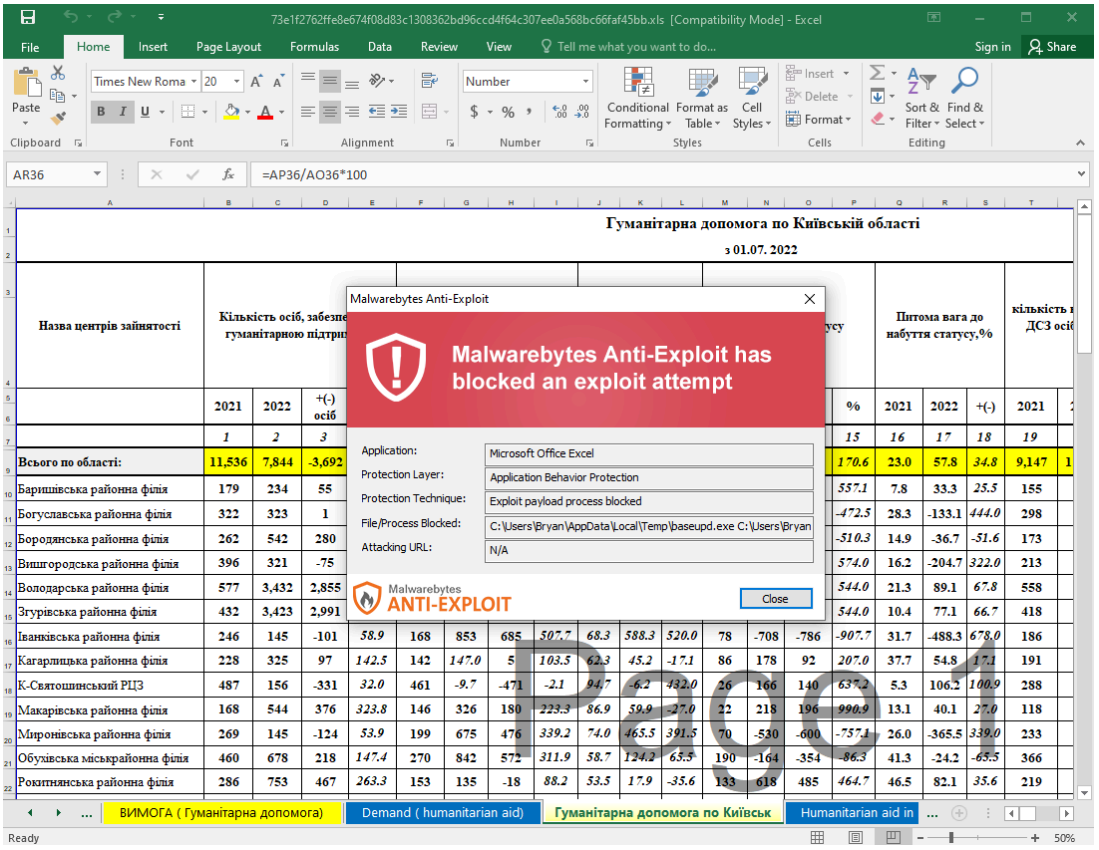
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

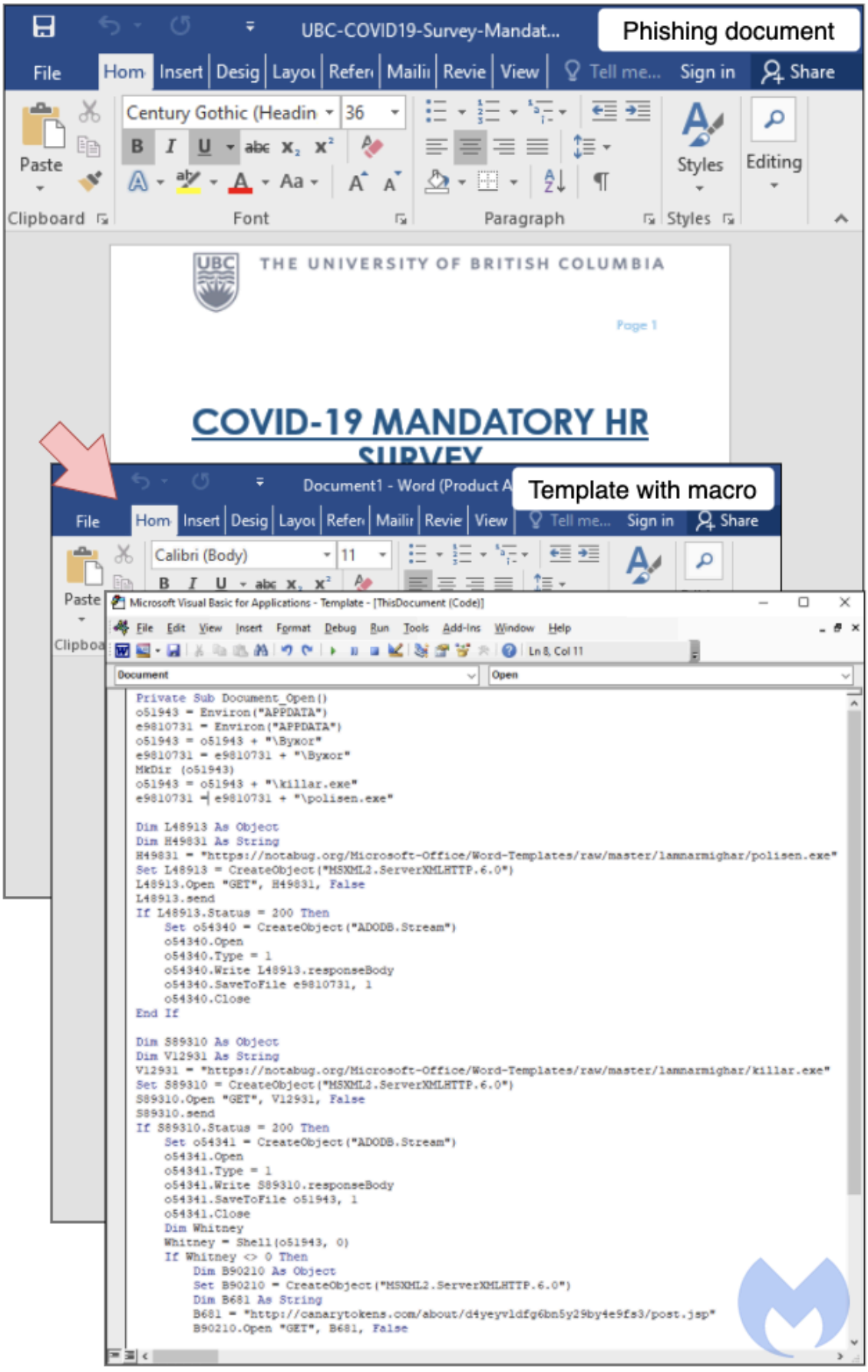
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

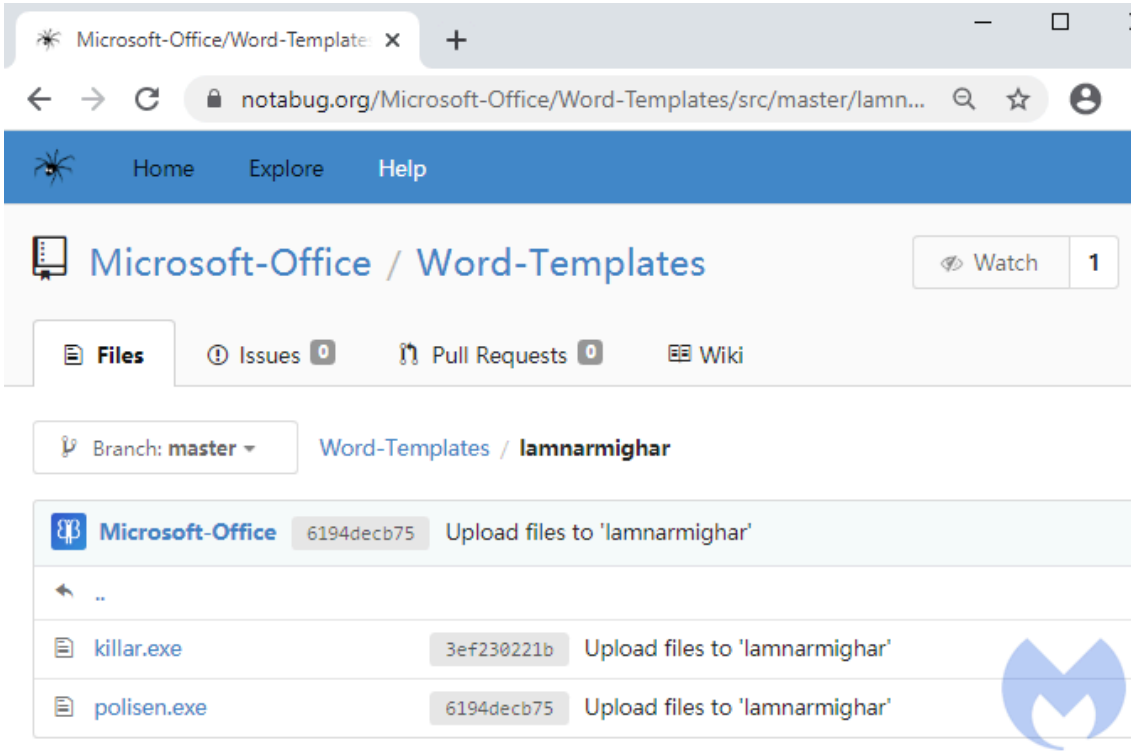
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



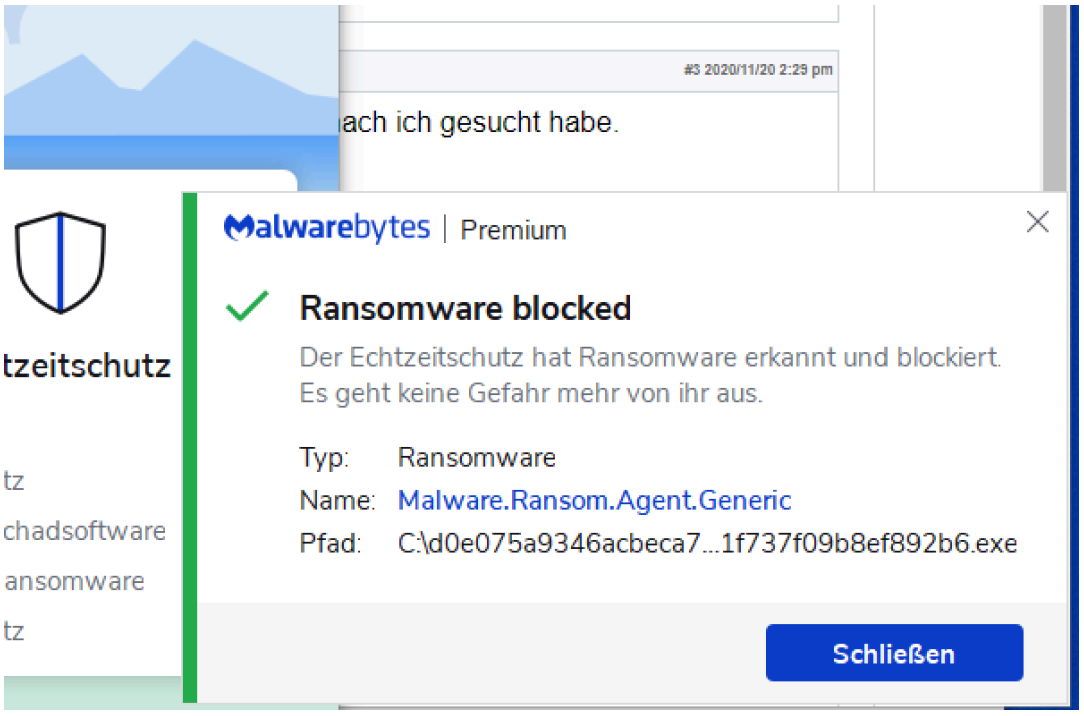
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

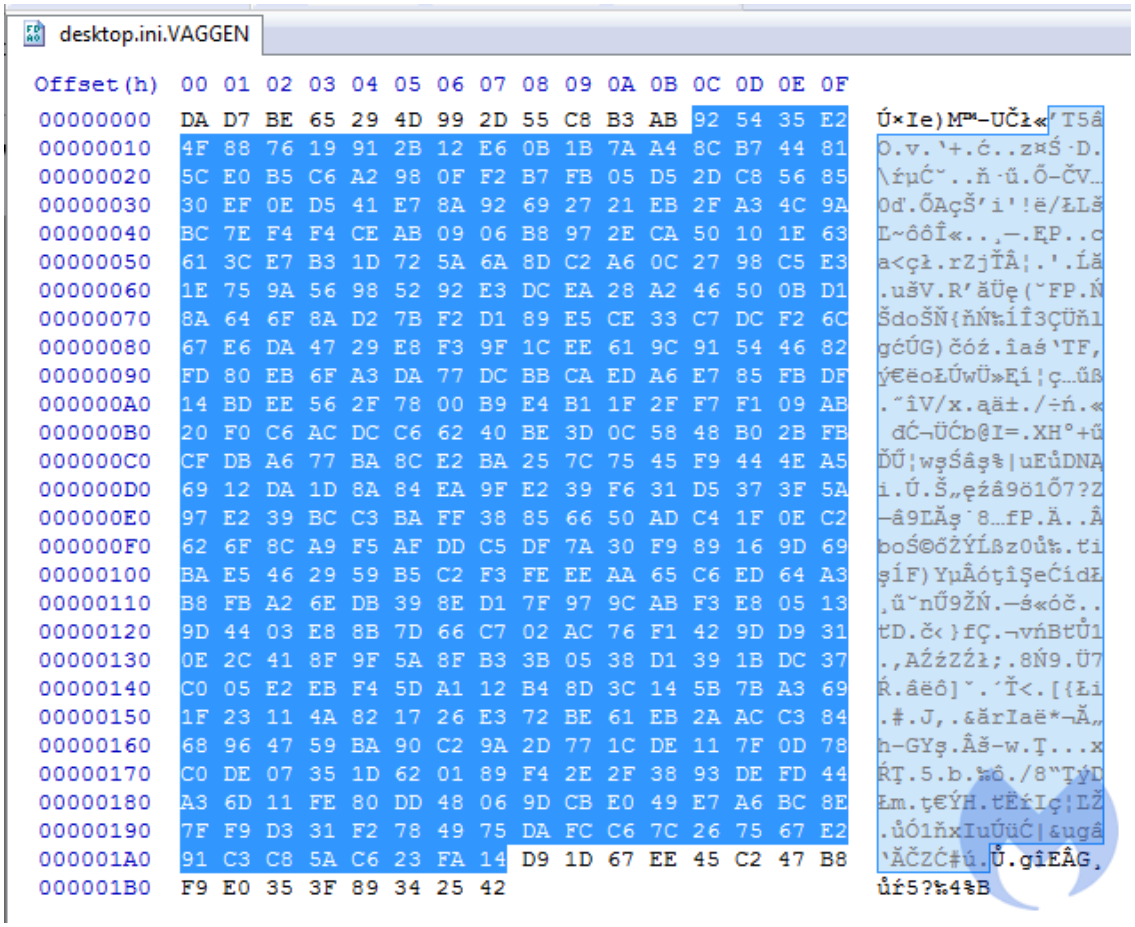
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

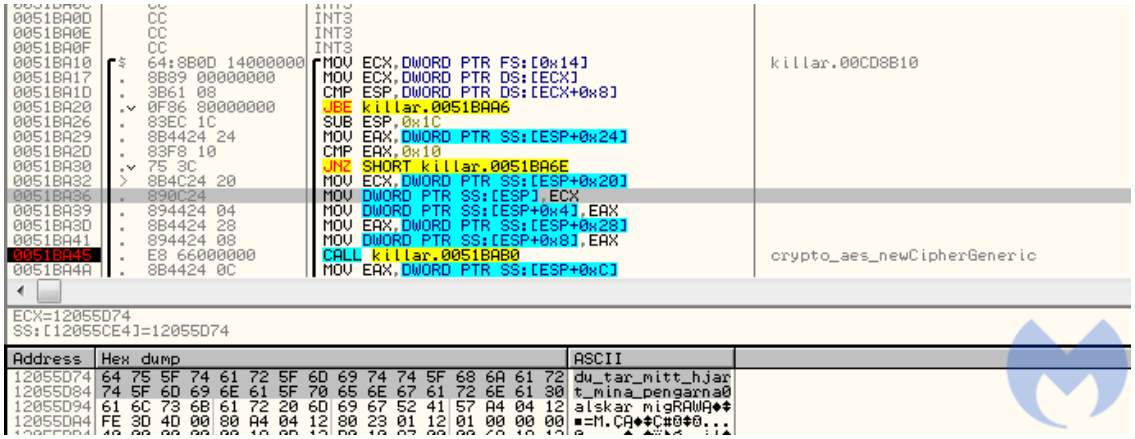


The content of the output file (with .VAGGEN extension) contains:

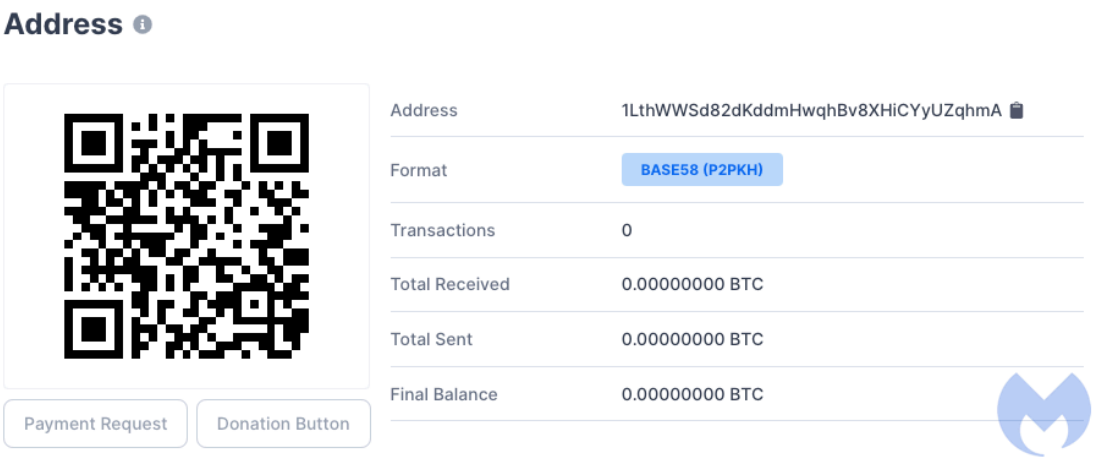
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

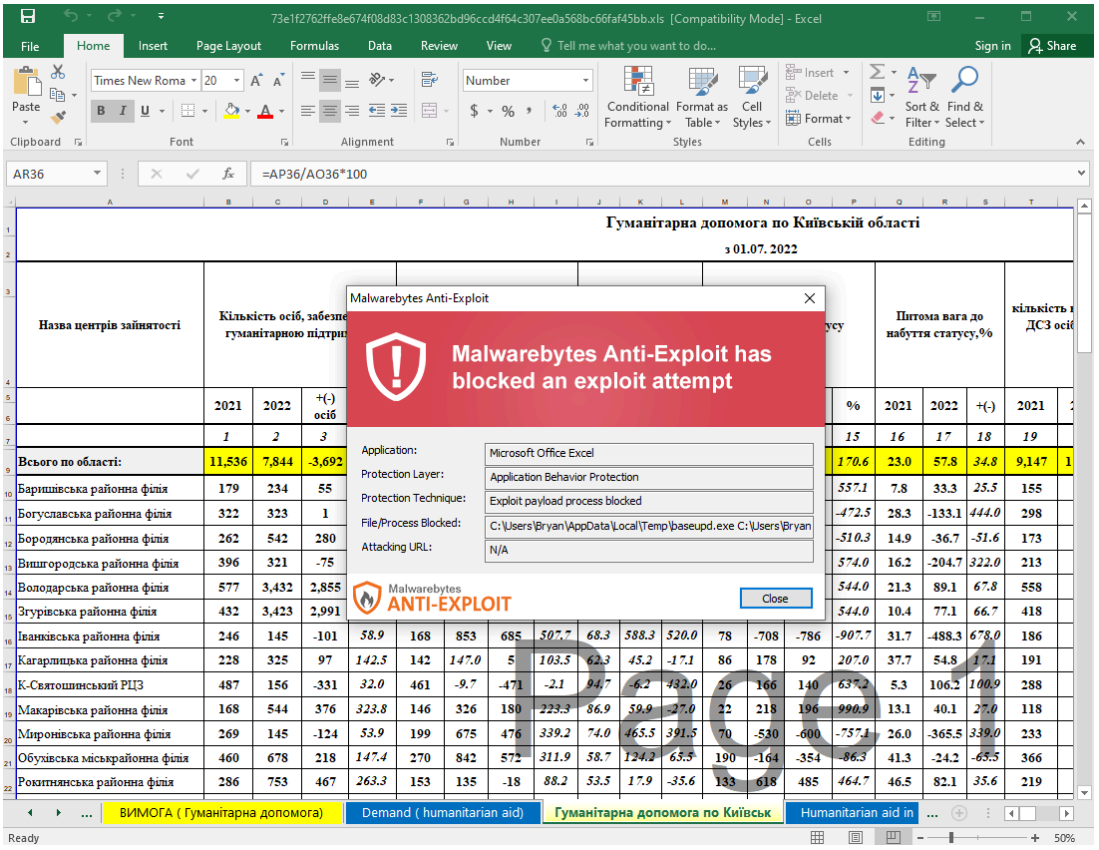
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

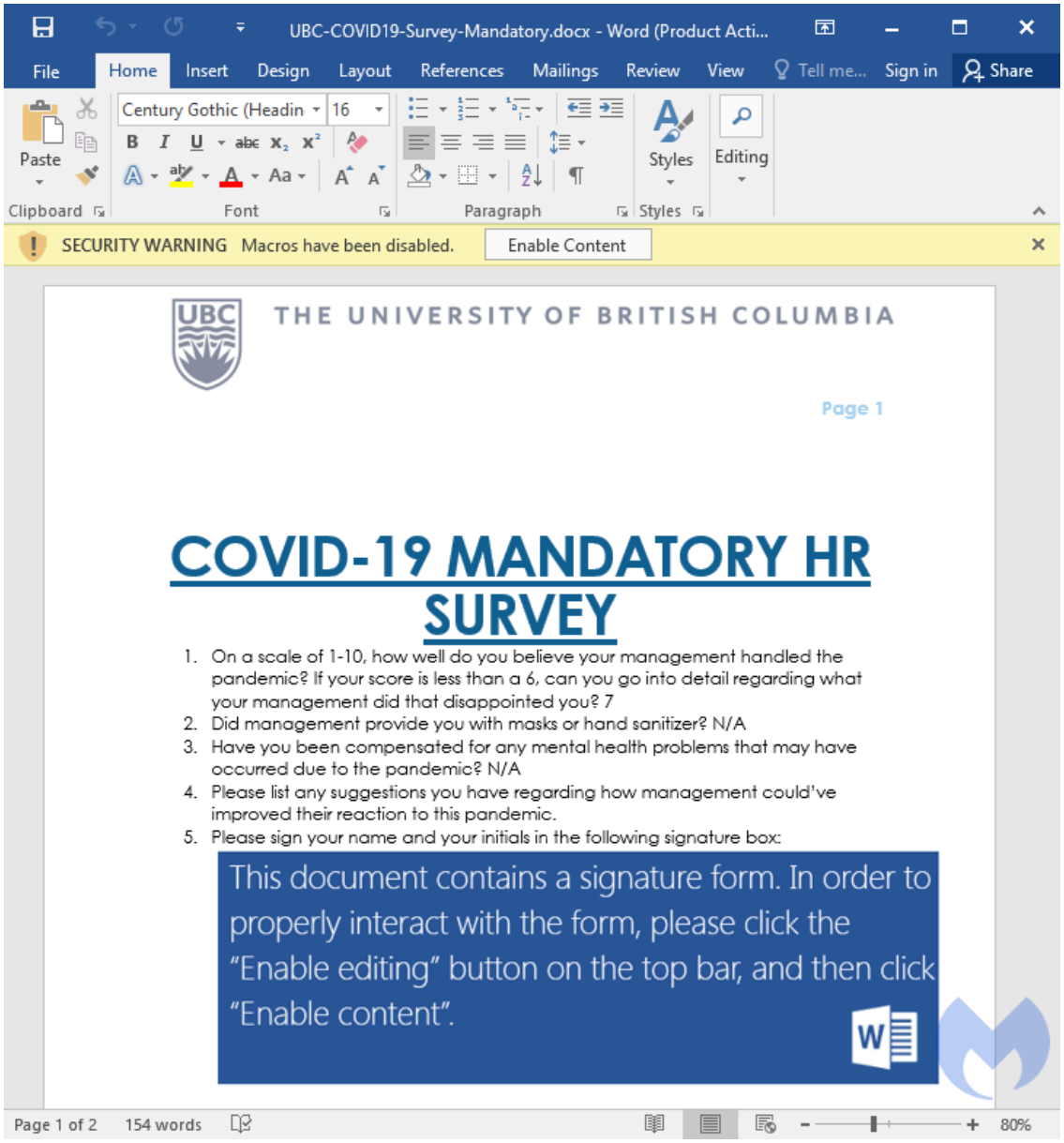
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

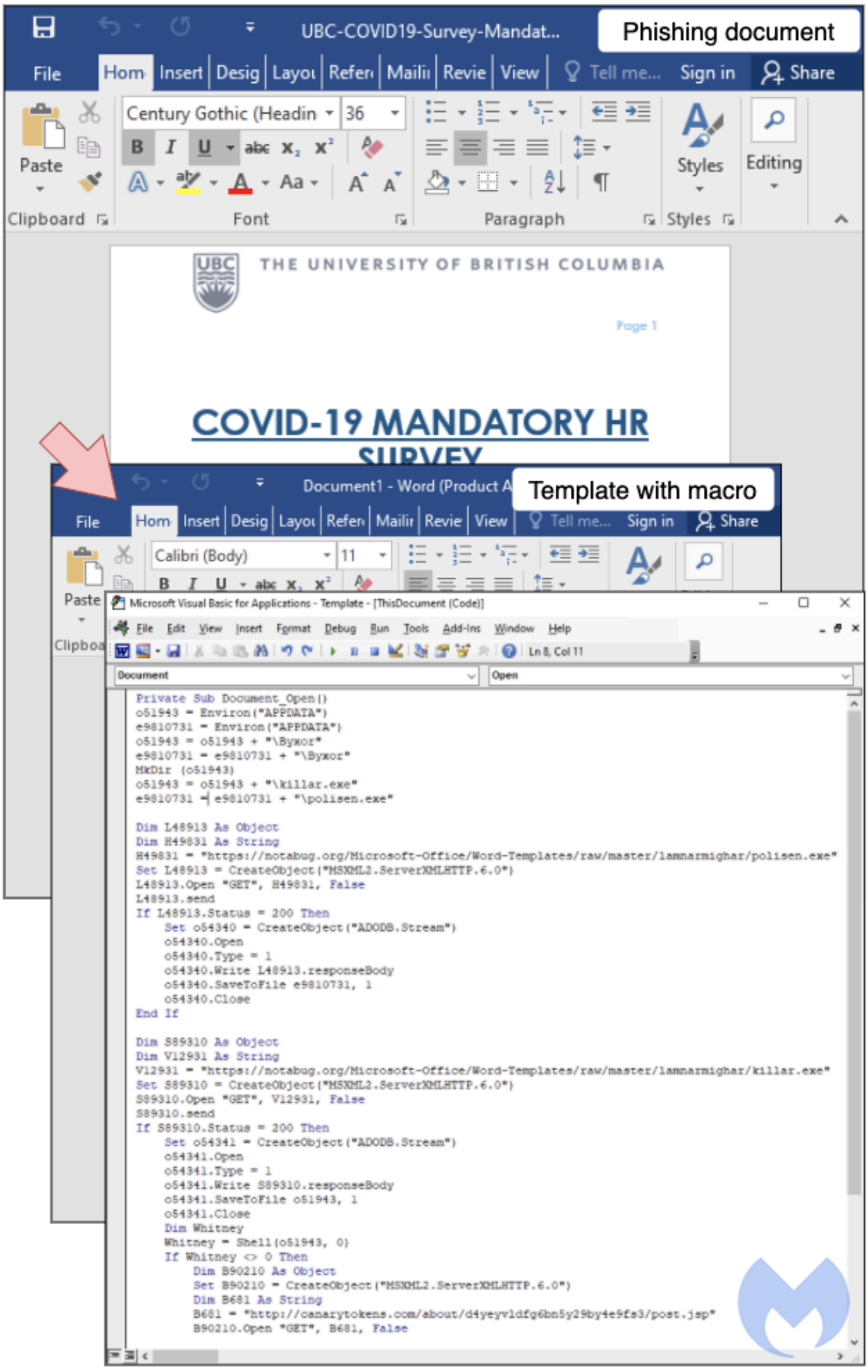
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

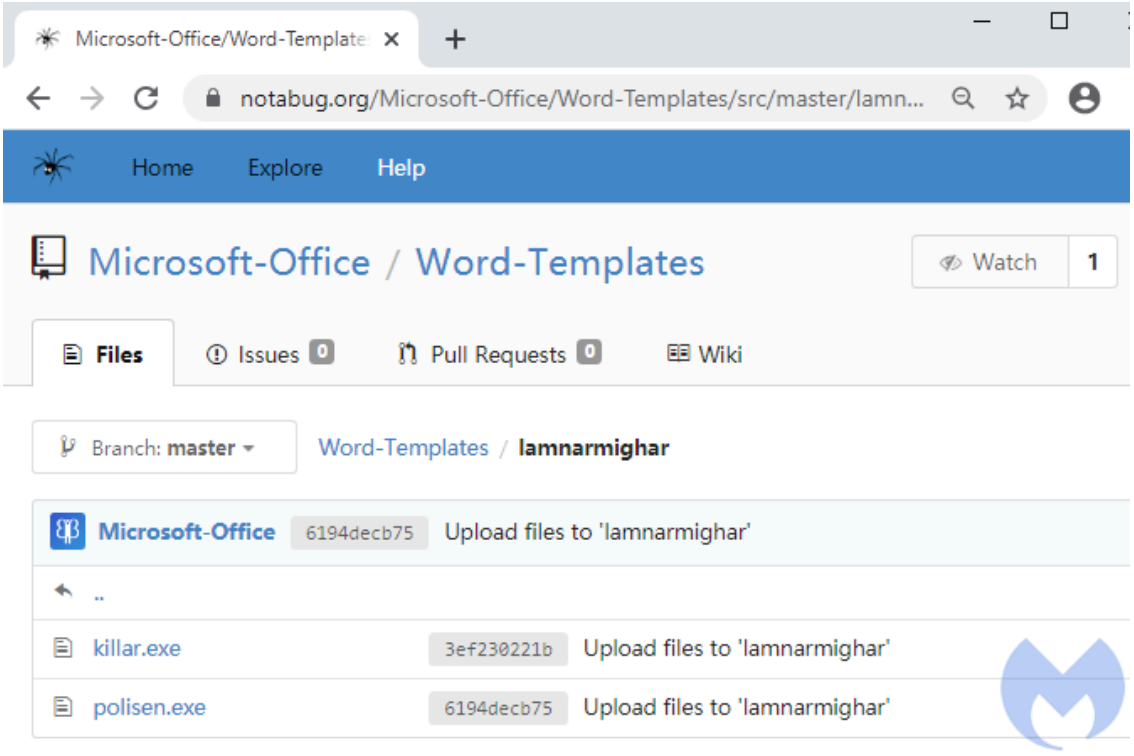
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



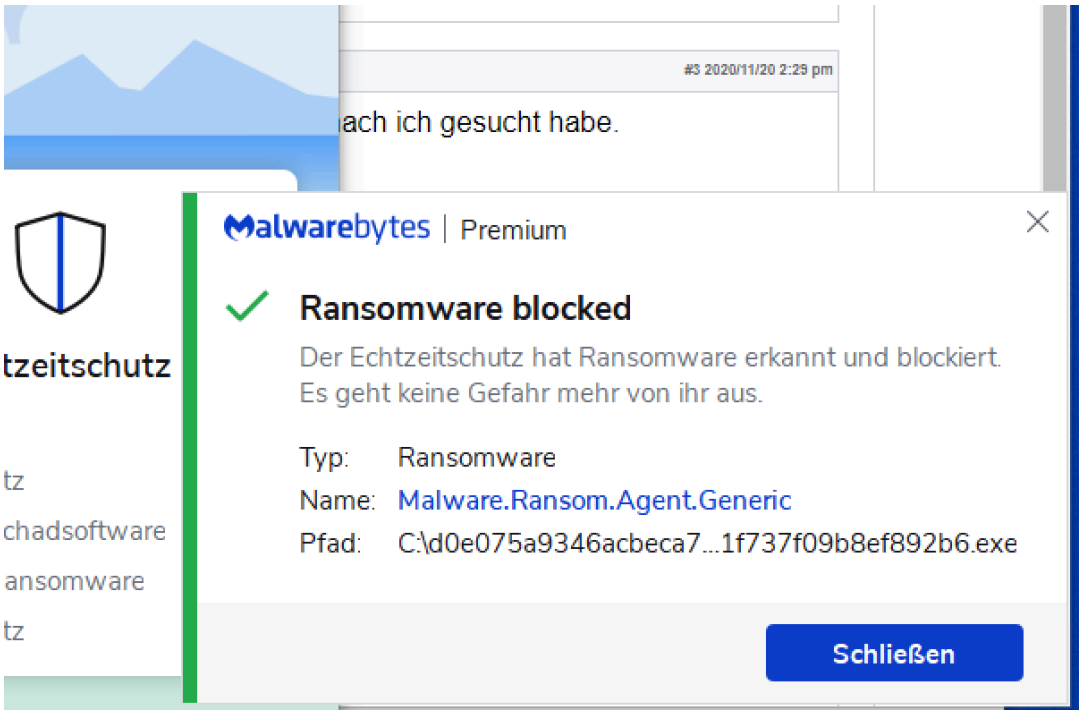
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

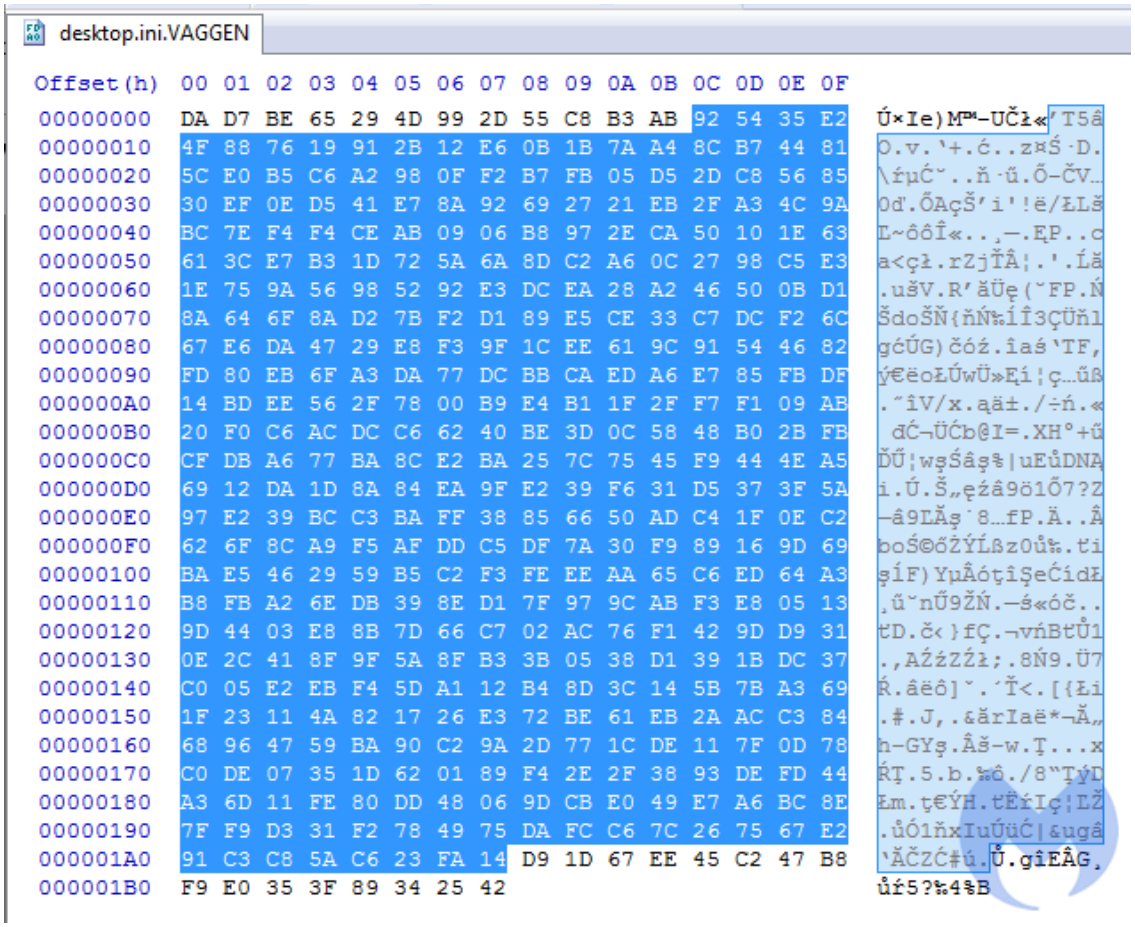
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

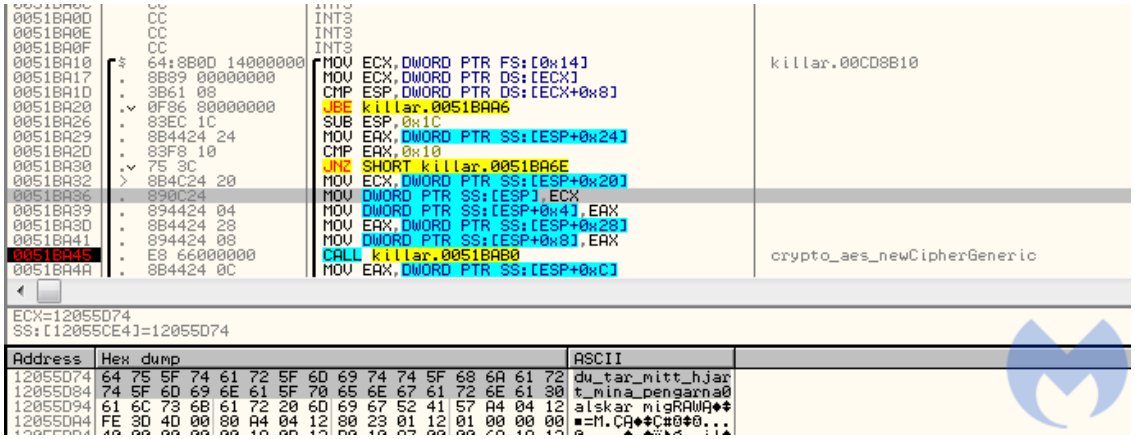


The content of the output file (with .VAGGEN extension) contains:

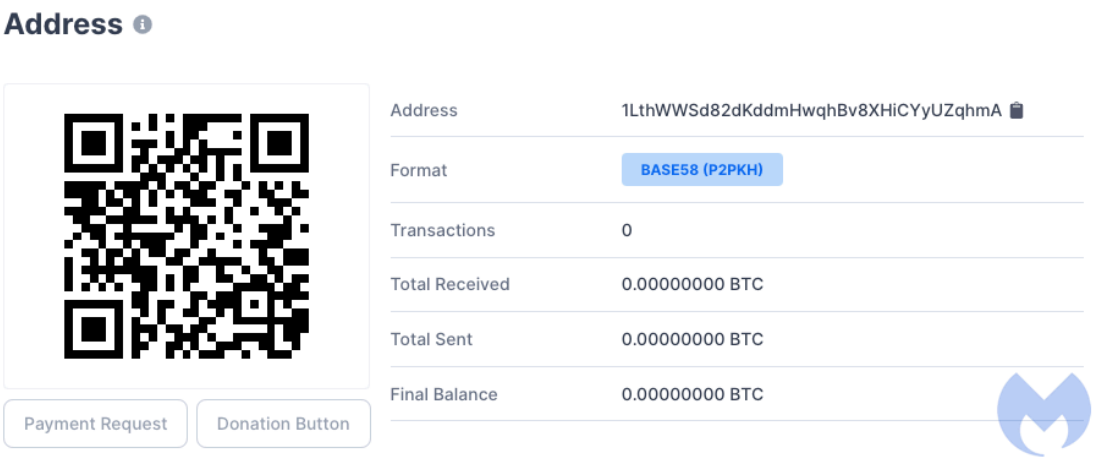
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

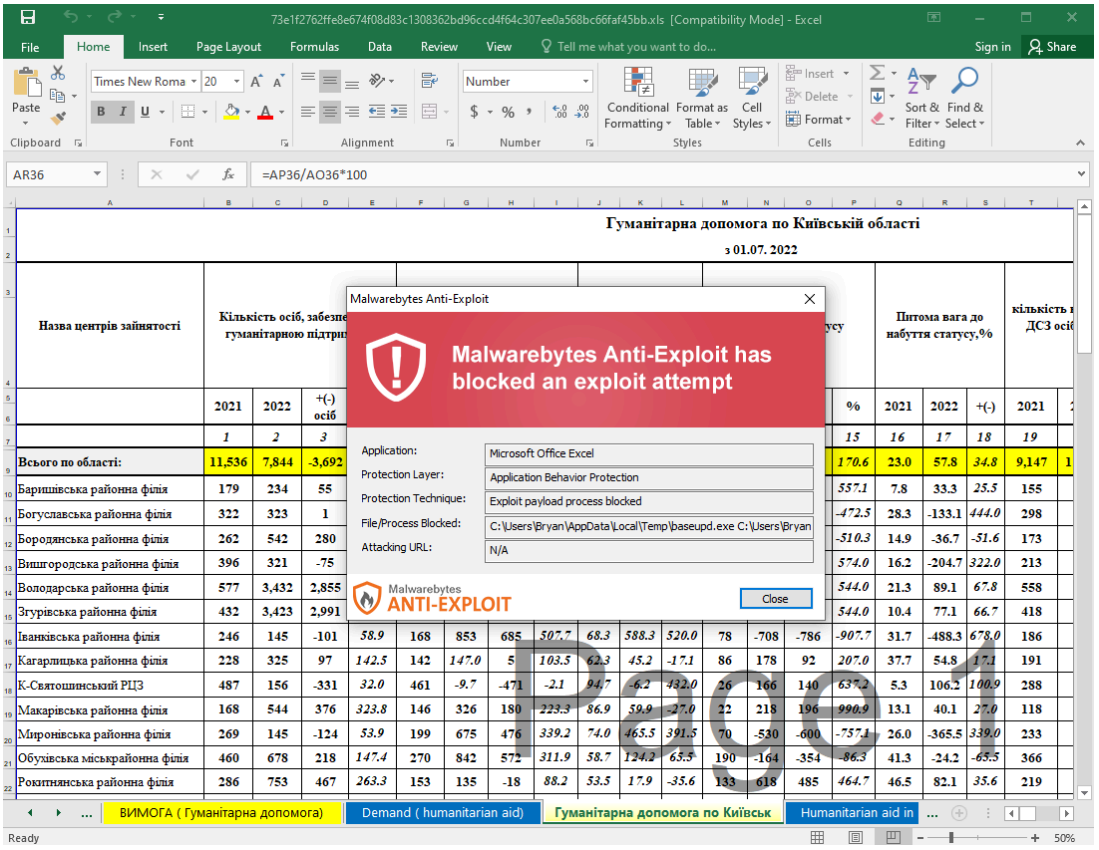
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

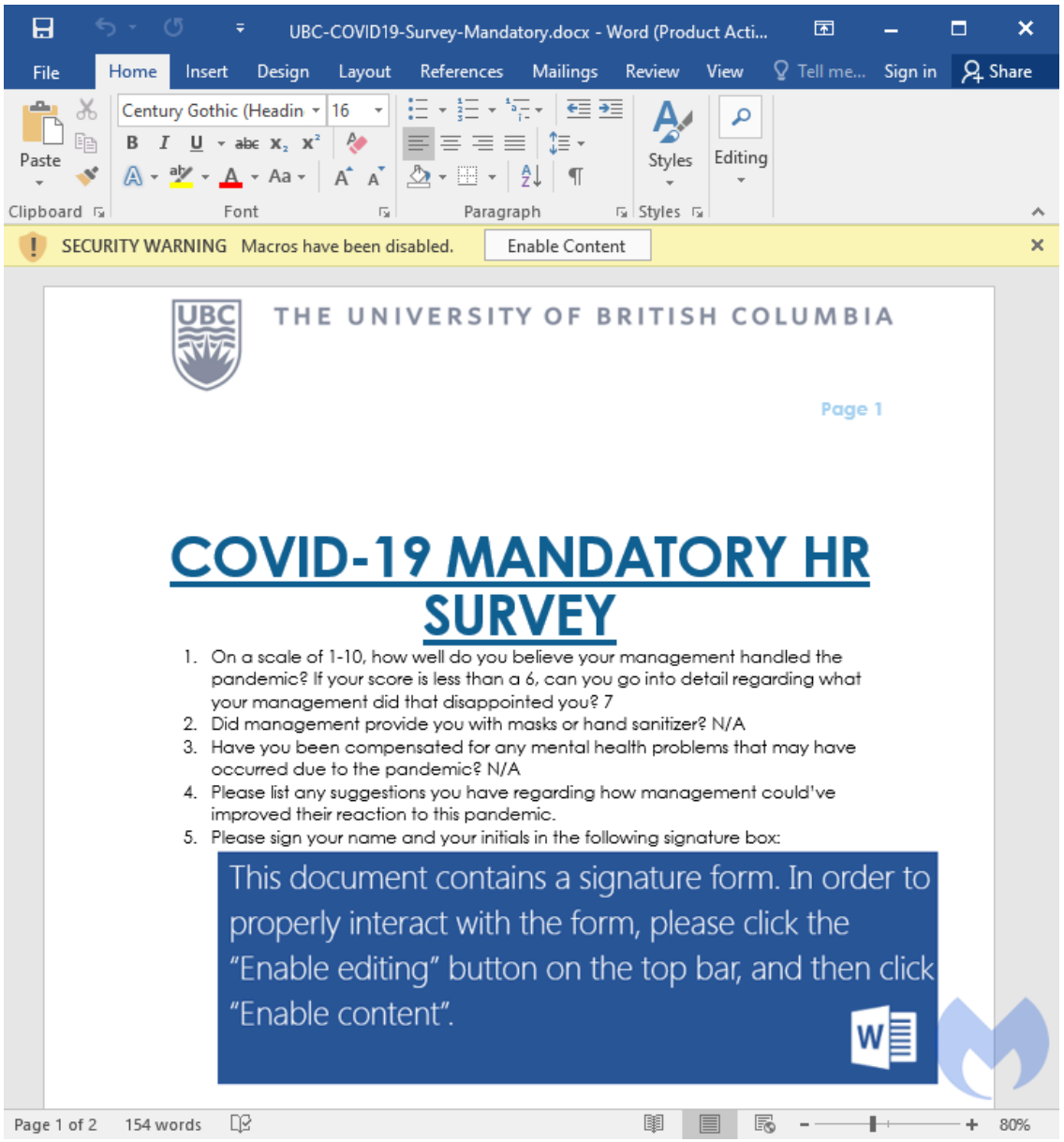
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

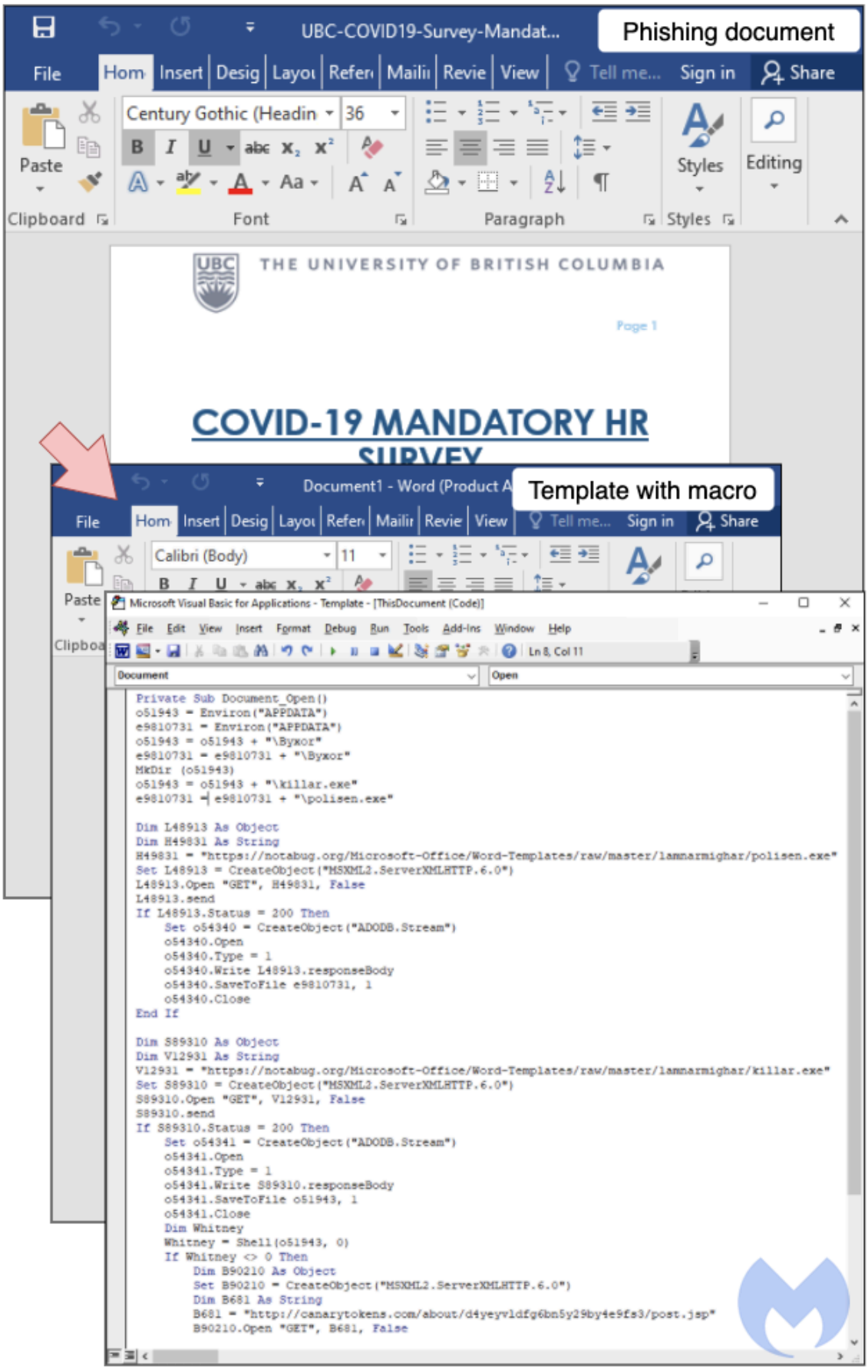
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

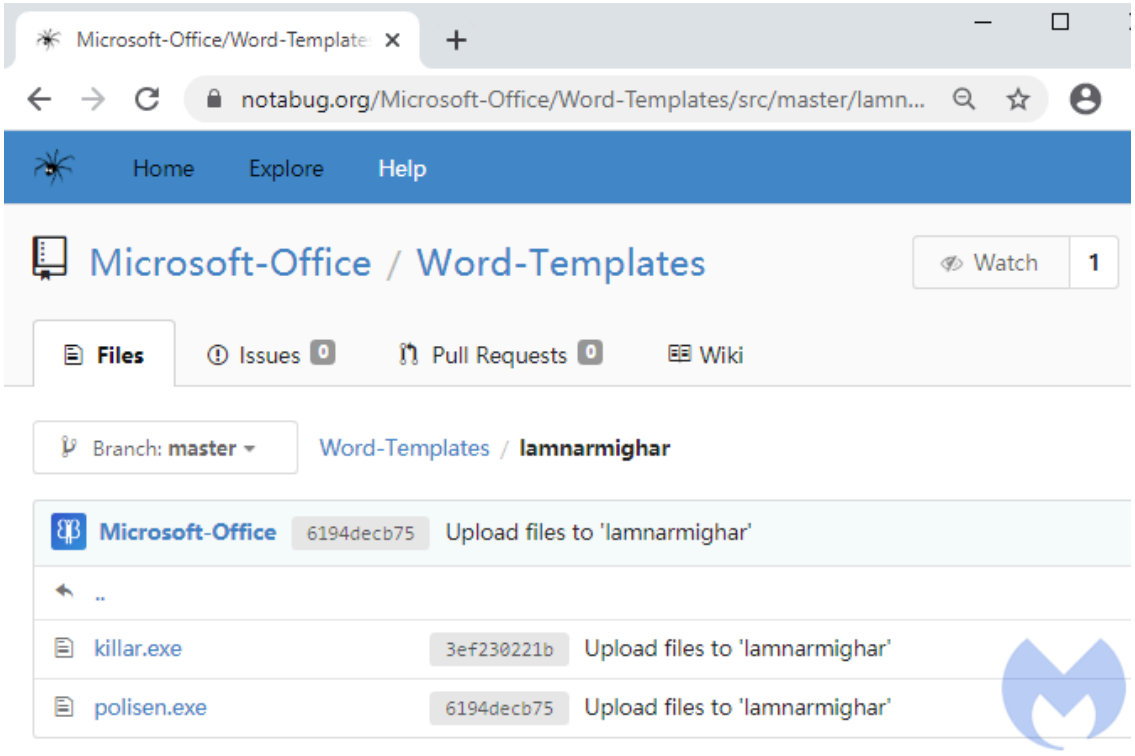
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



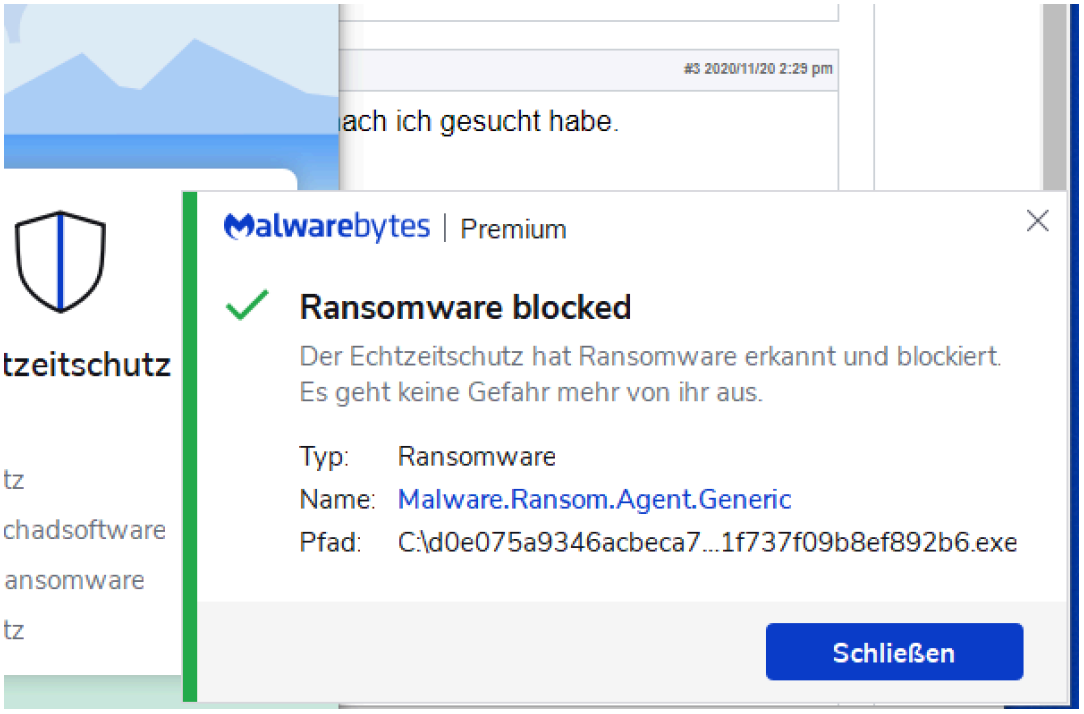
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

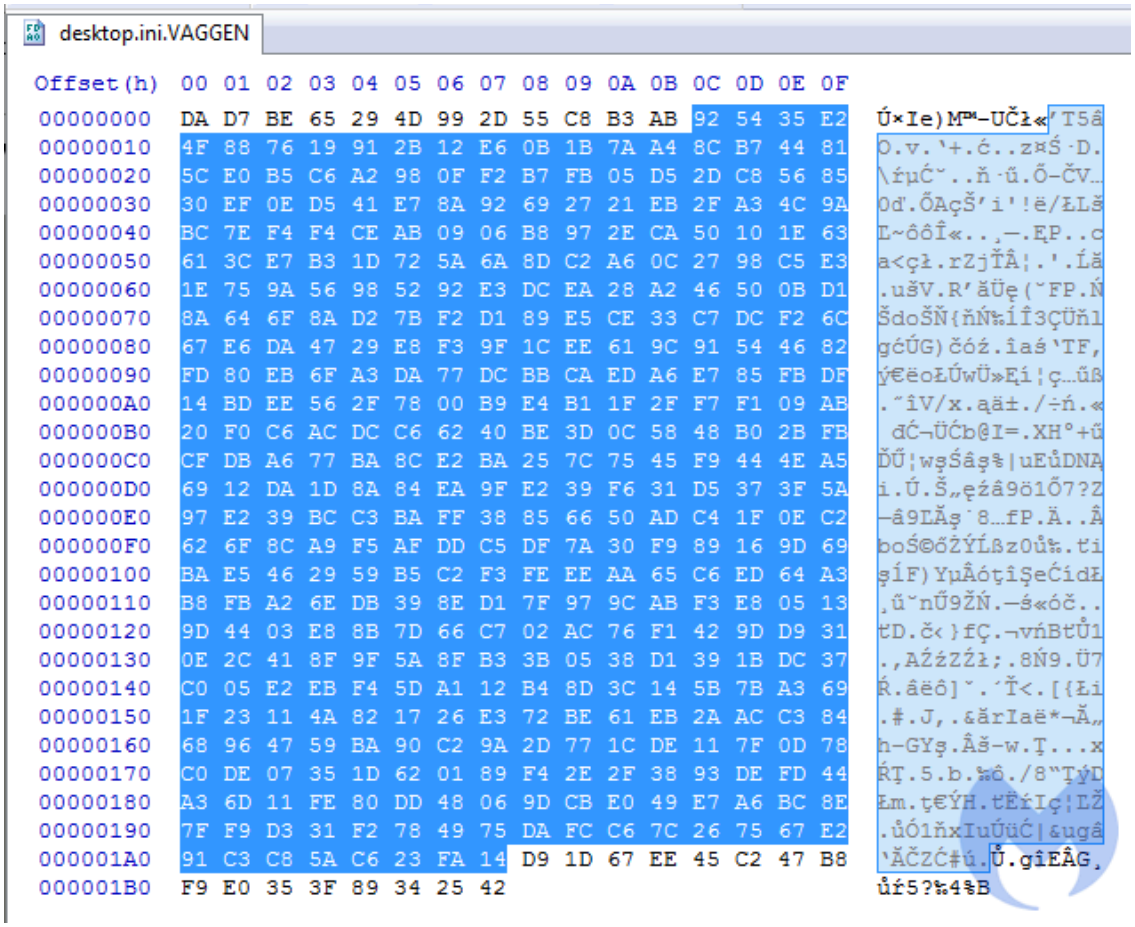
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

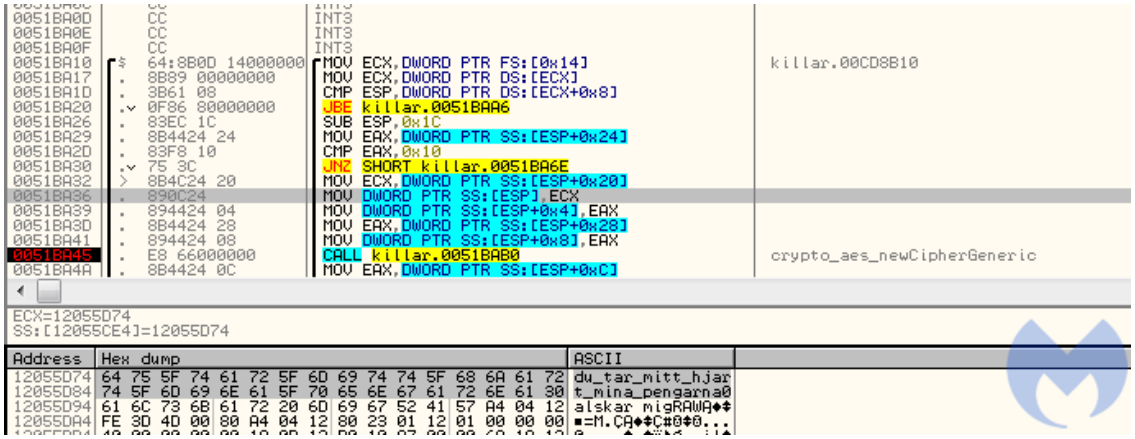


The content of the output file (with .VAGGEN extension) contains:

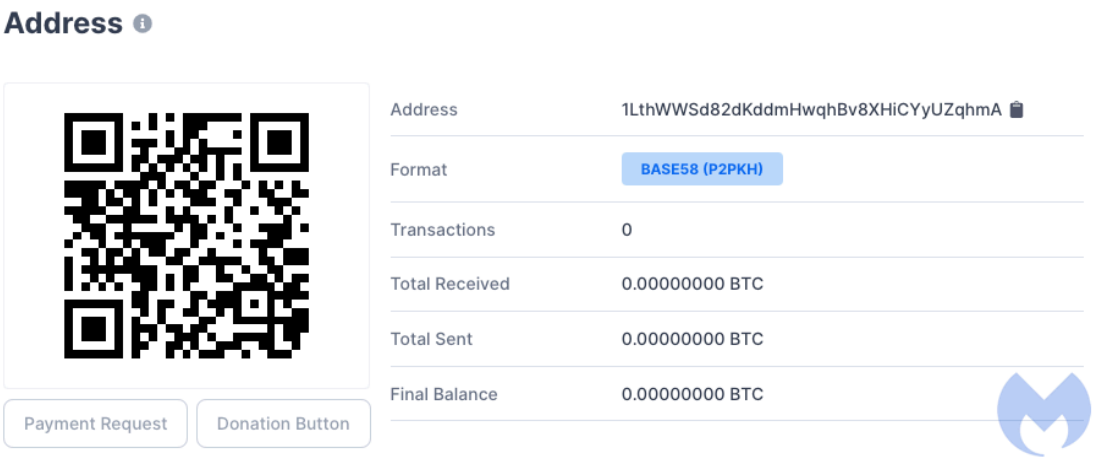
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

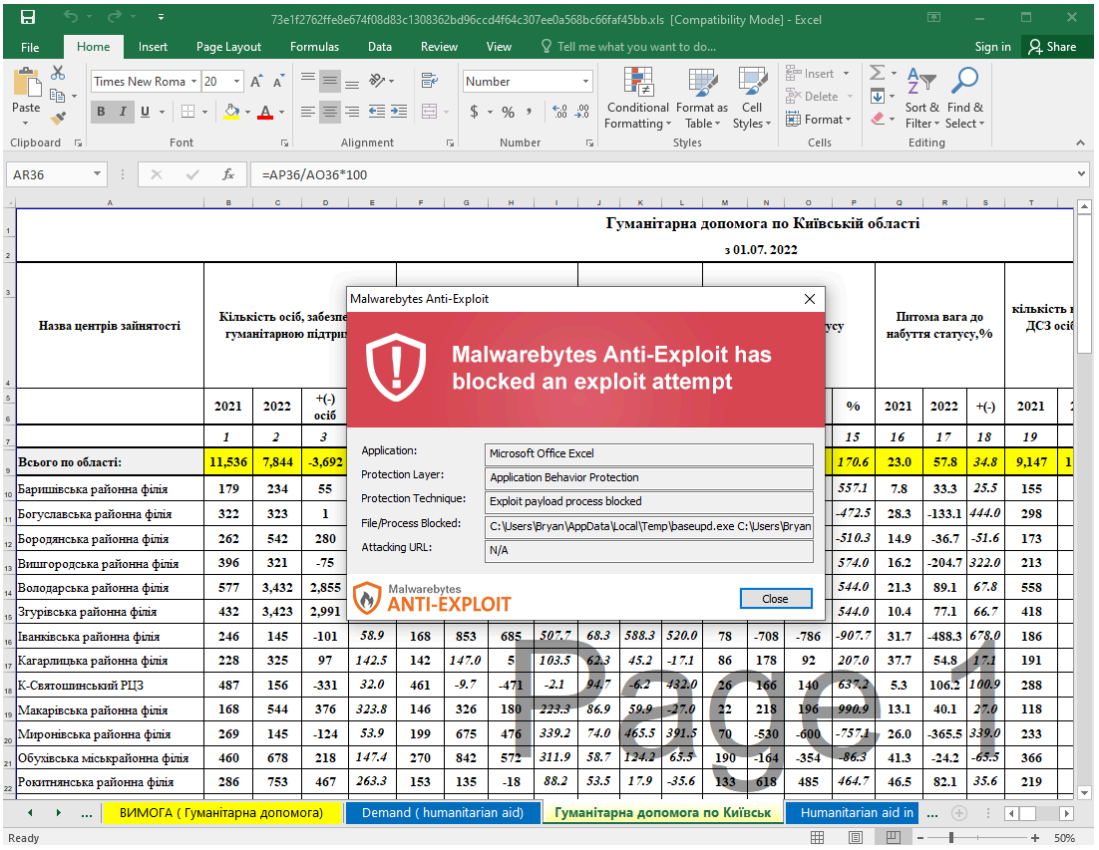
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

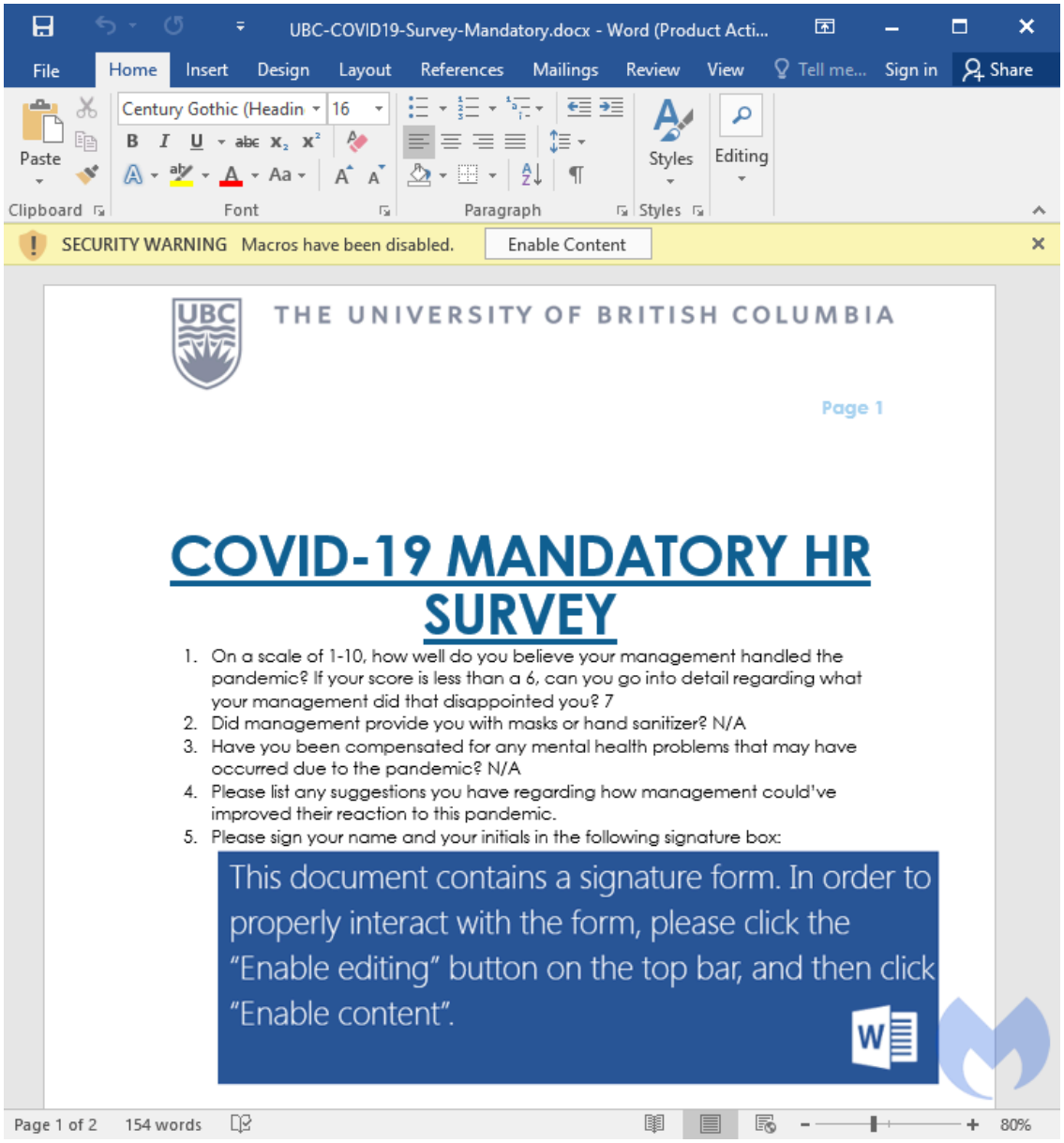
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

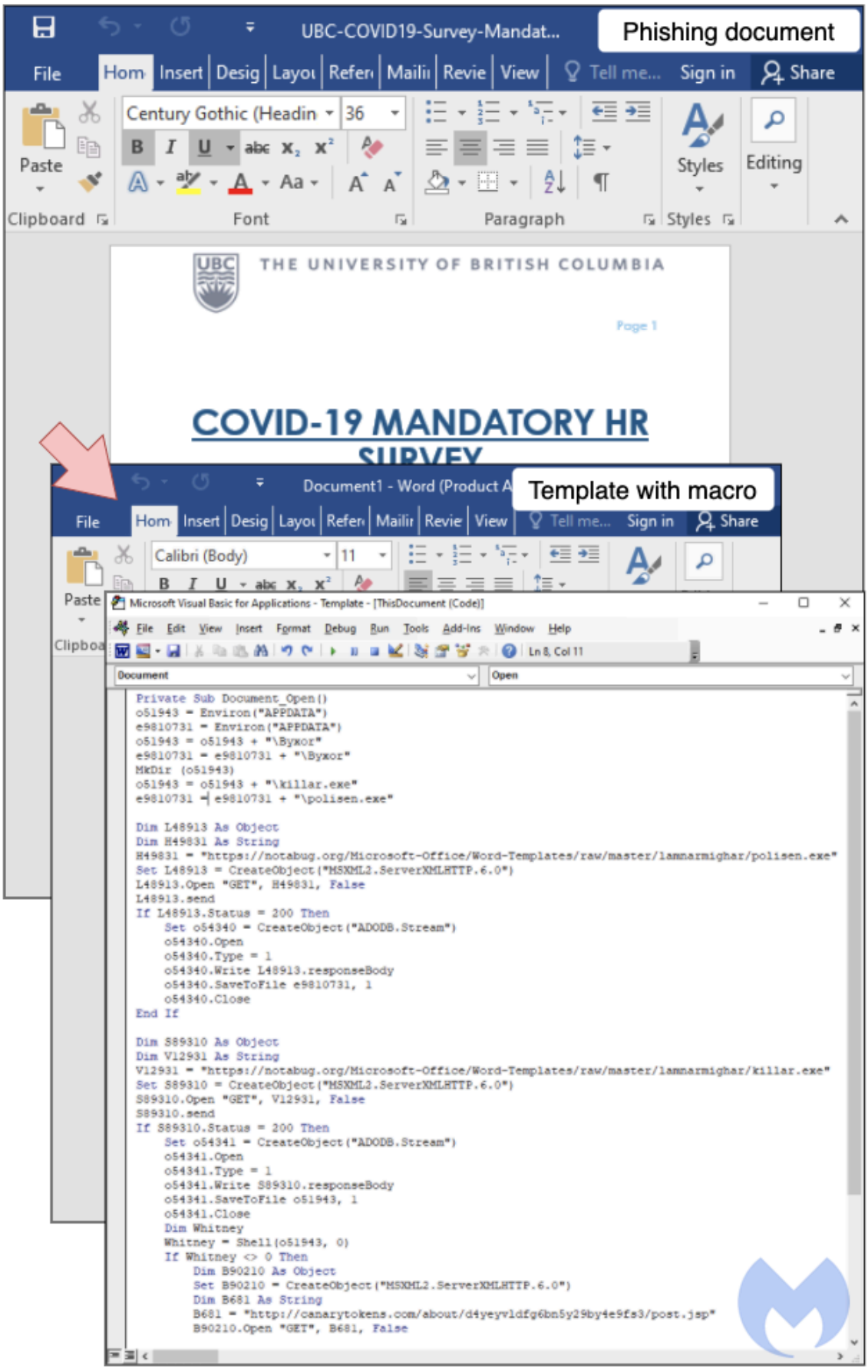
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

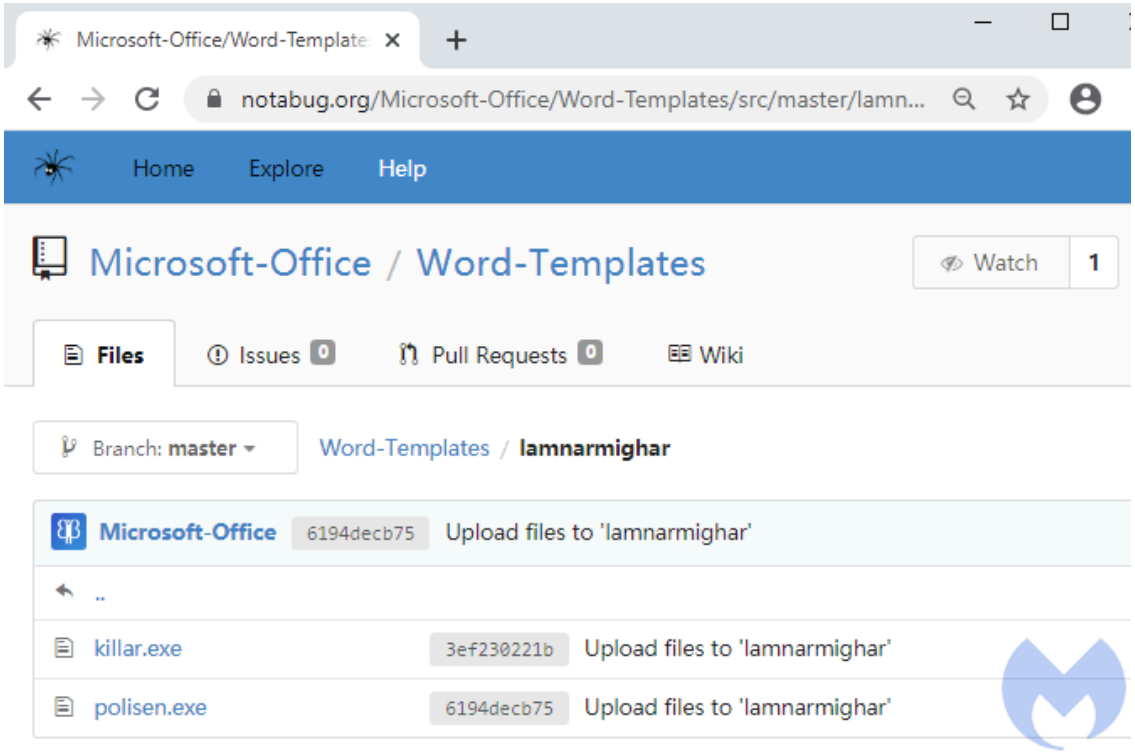
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



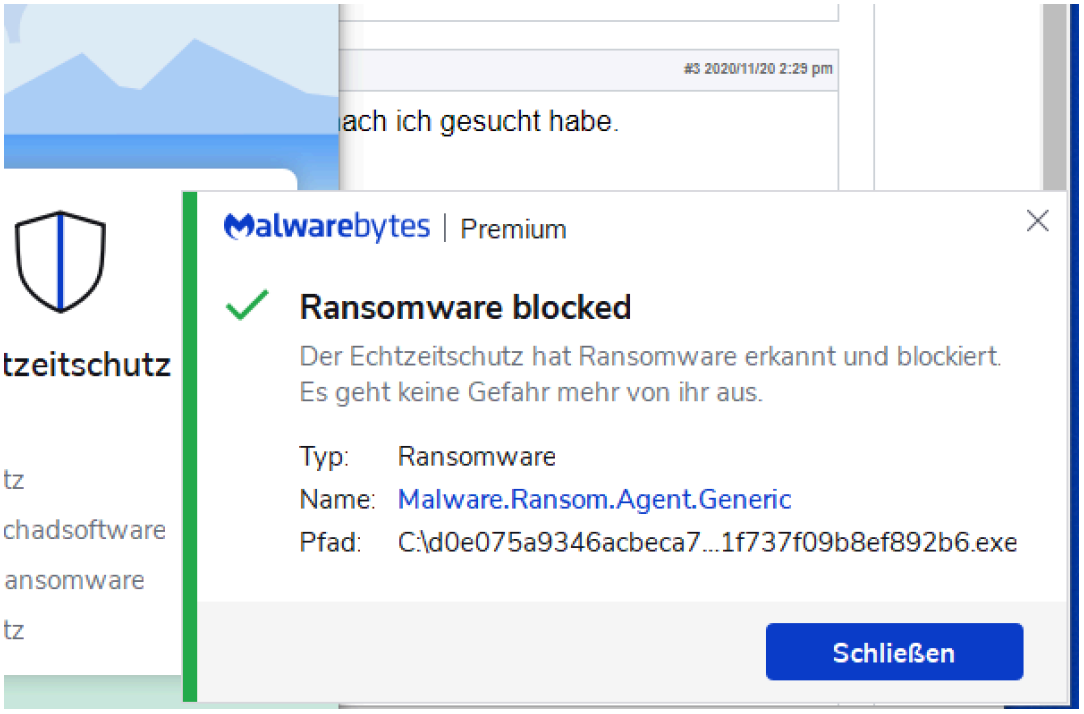
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

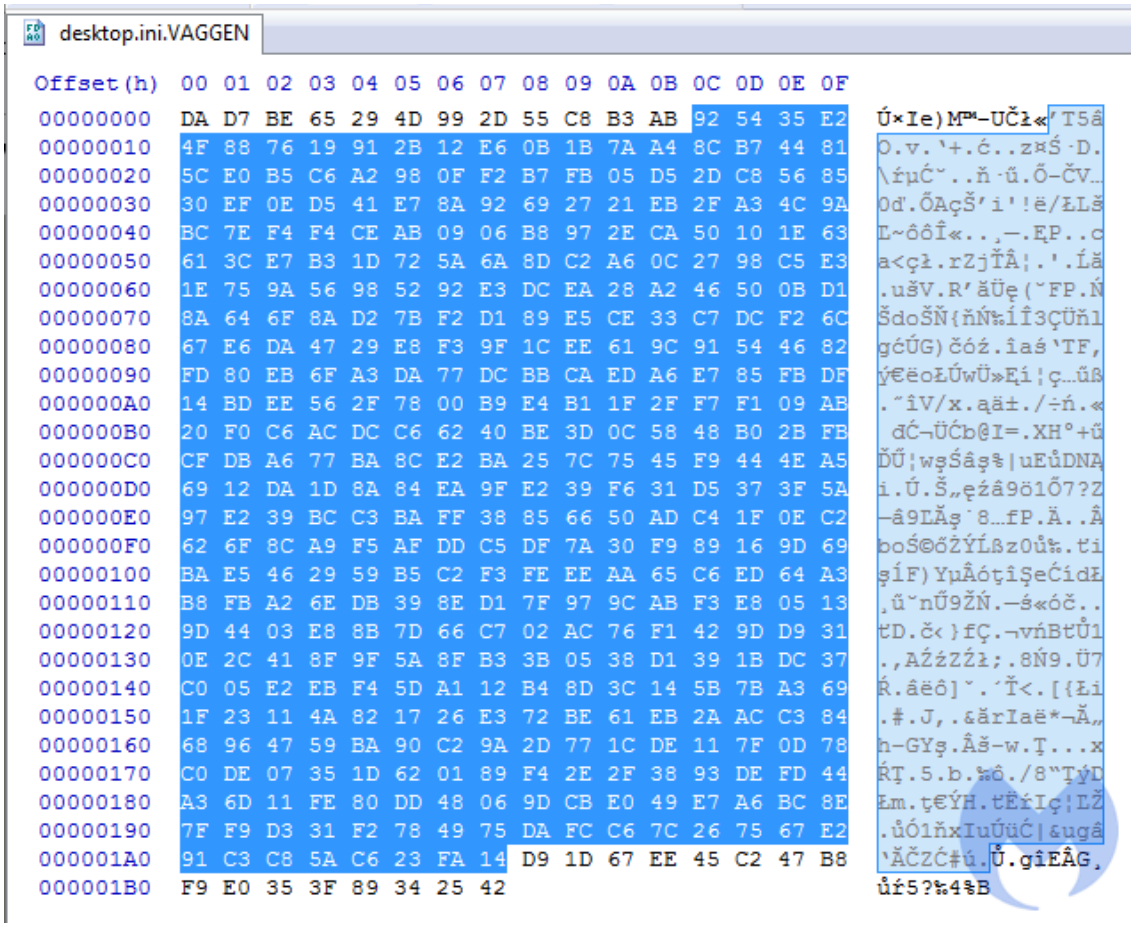
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

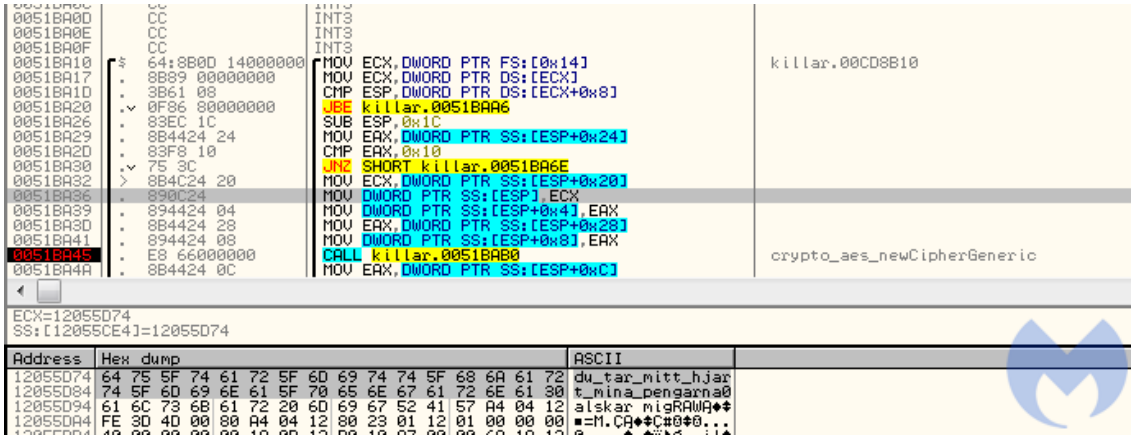


The content of the output file (with .VAGGEN extension) contains:

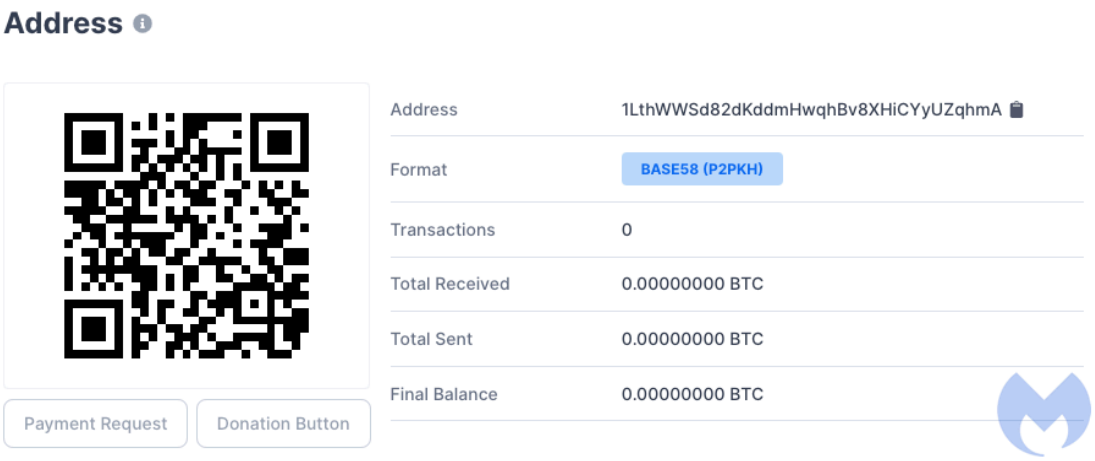
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

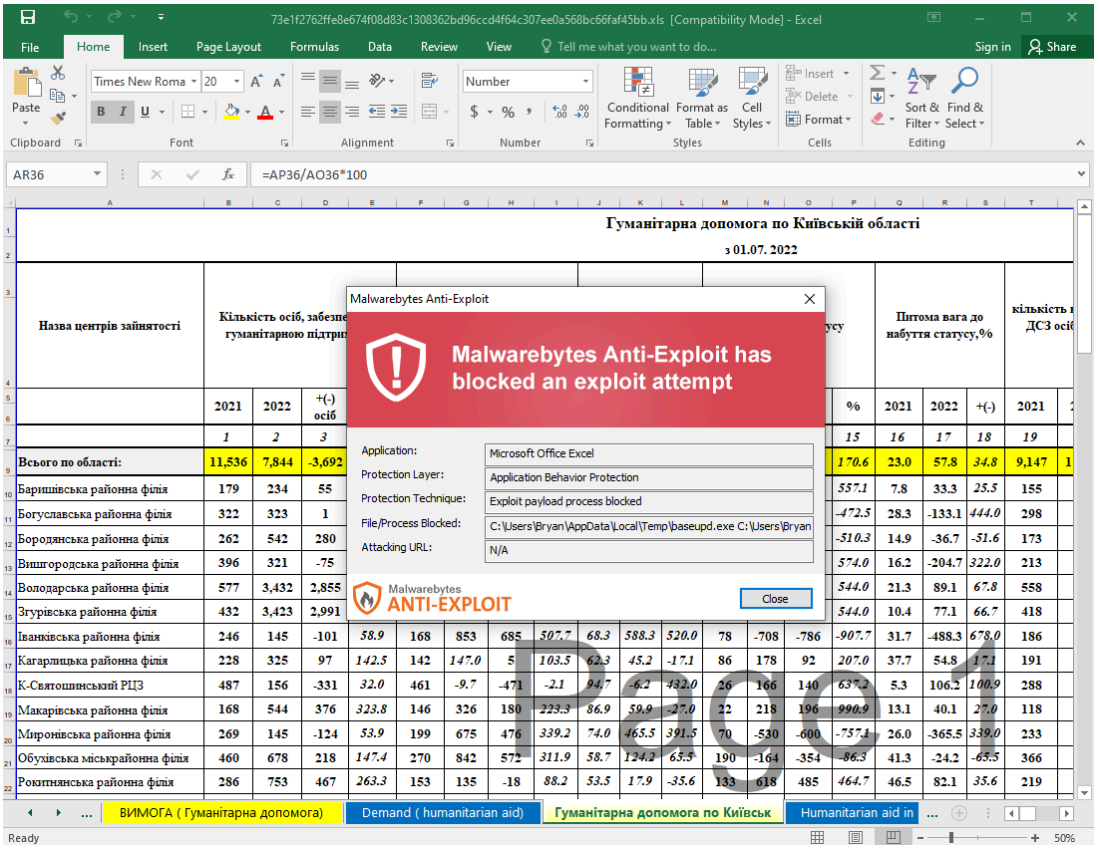
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

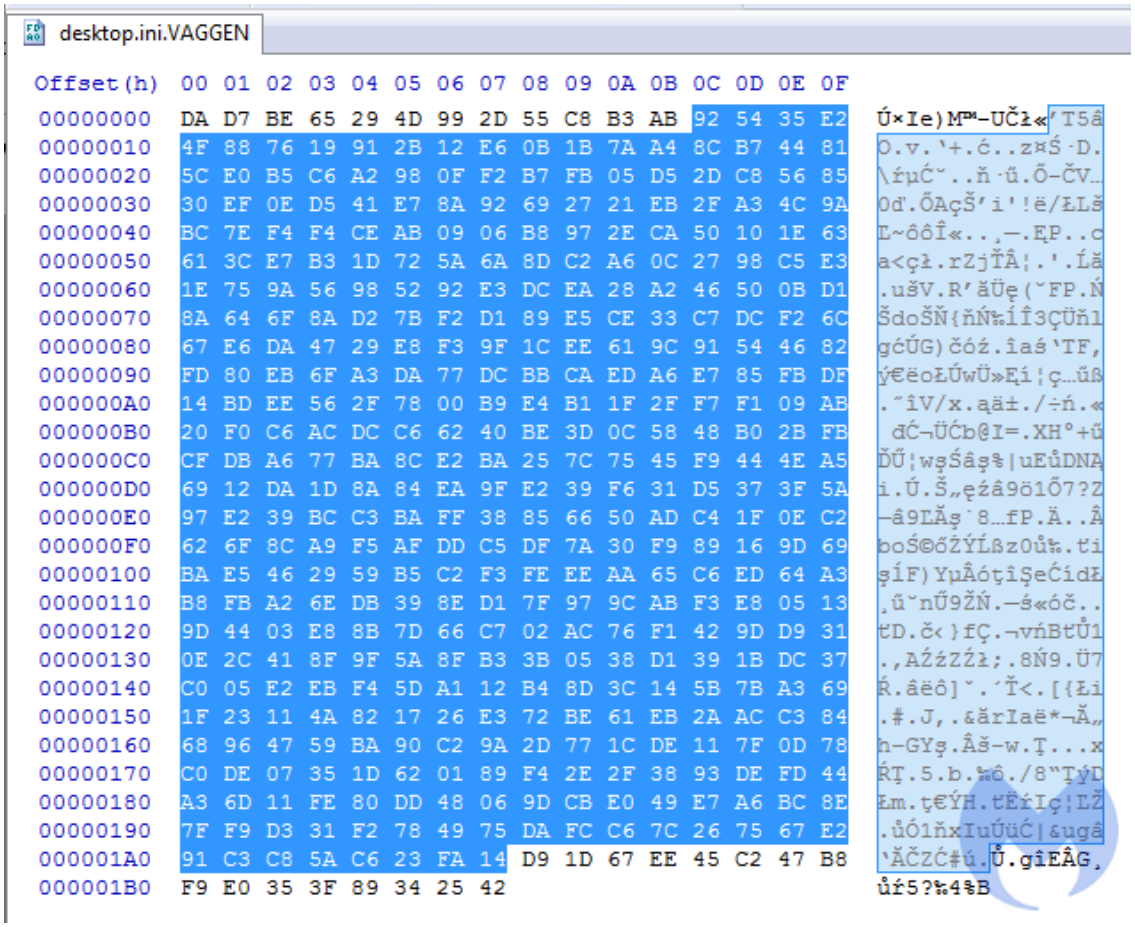
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

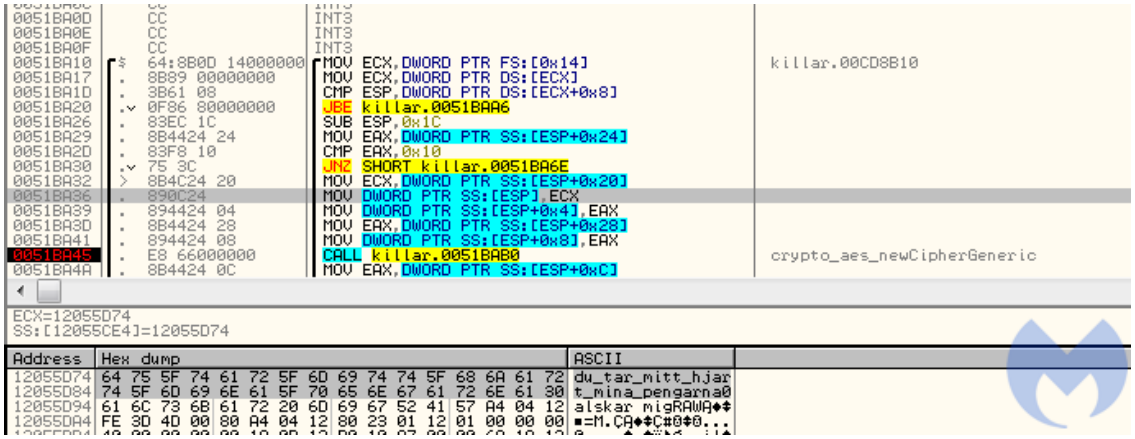
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

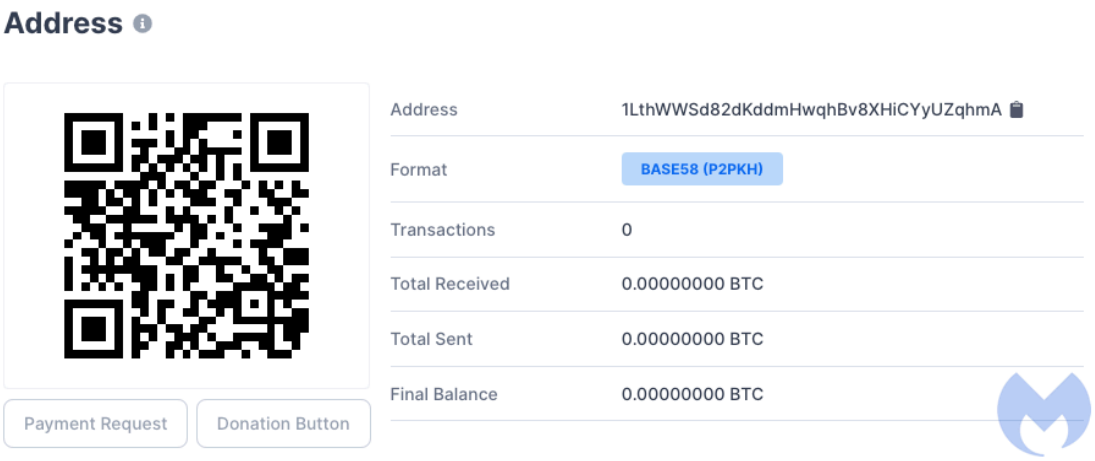
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

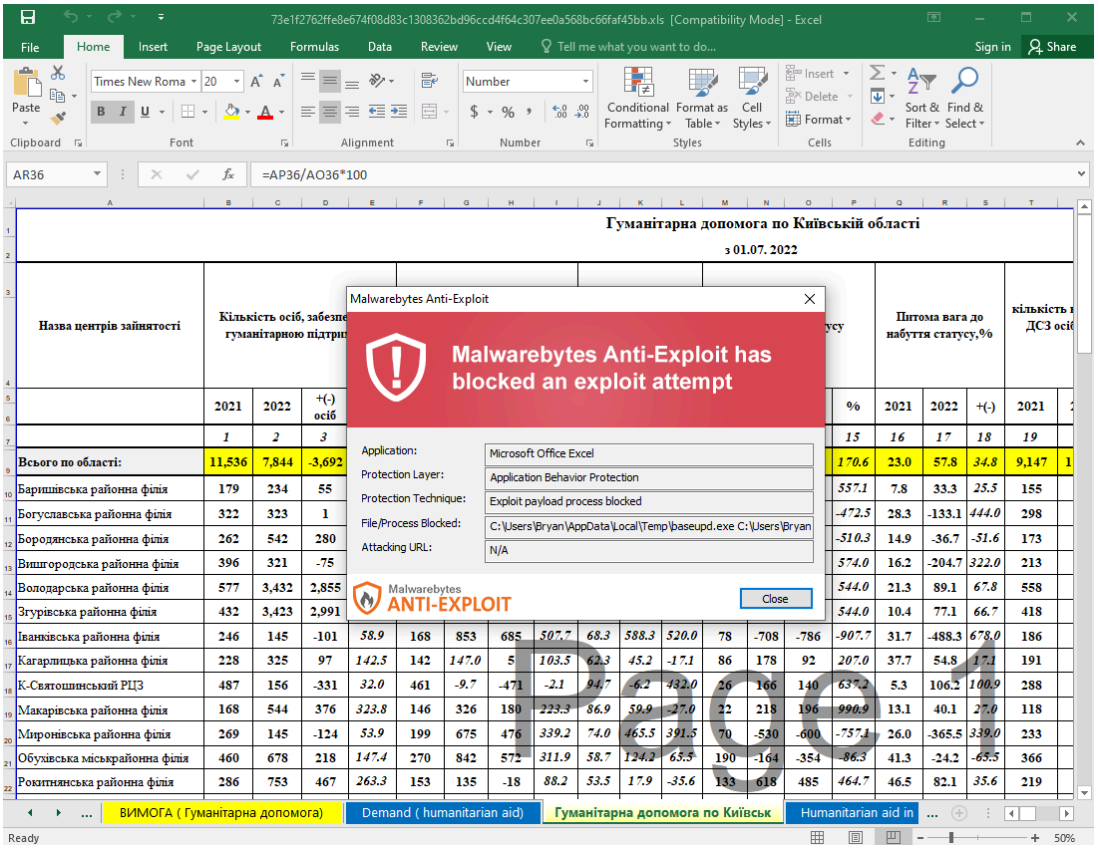
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

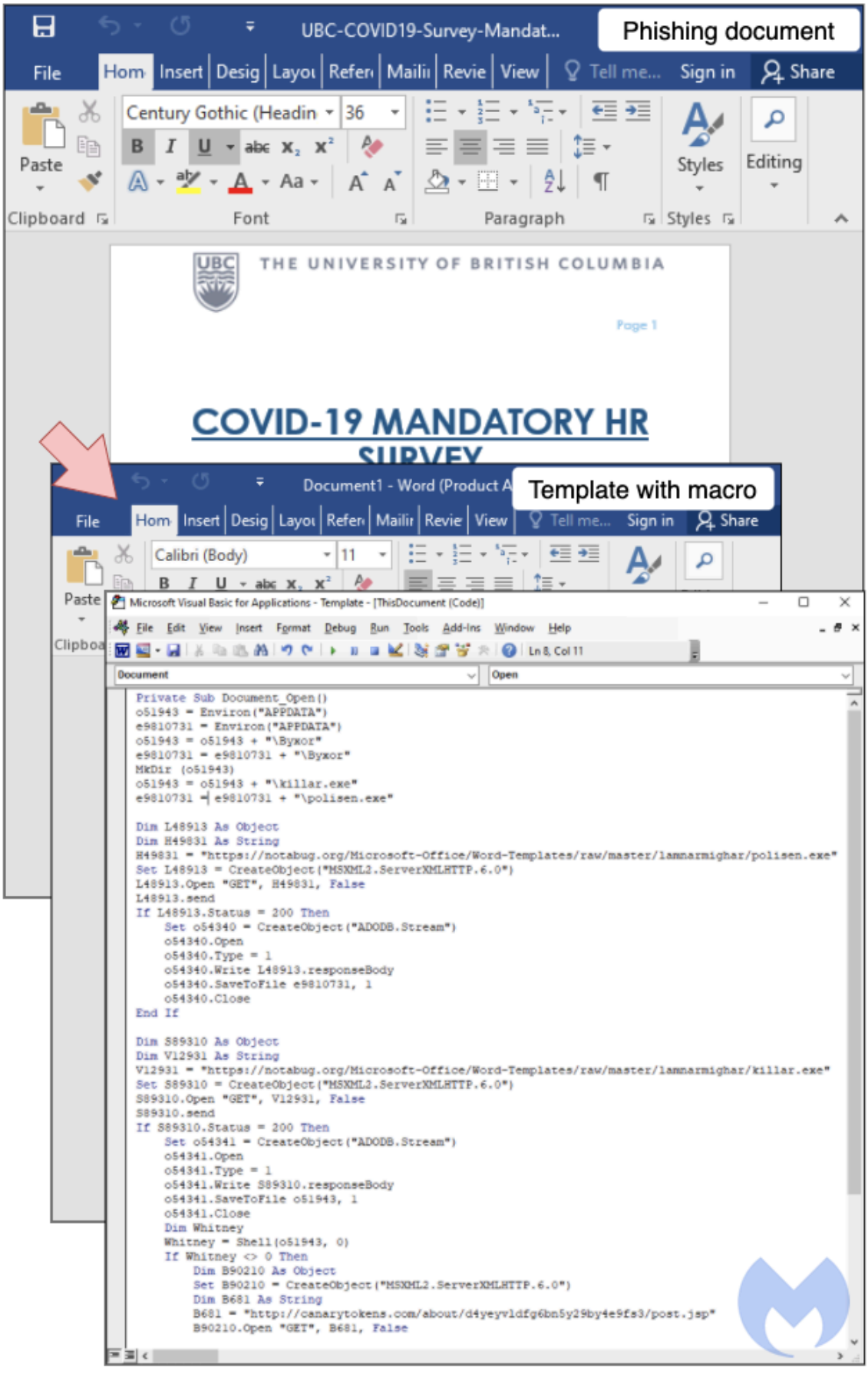
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

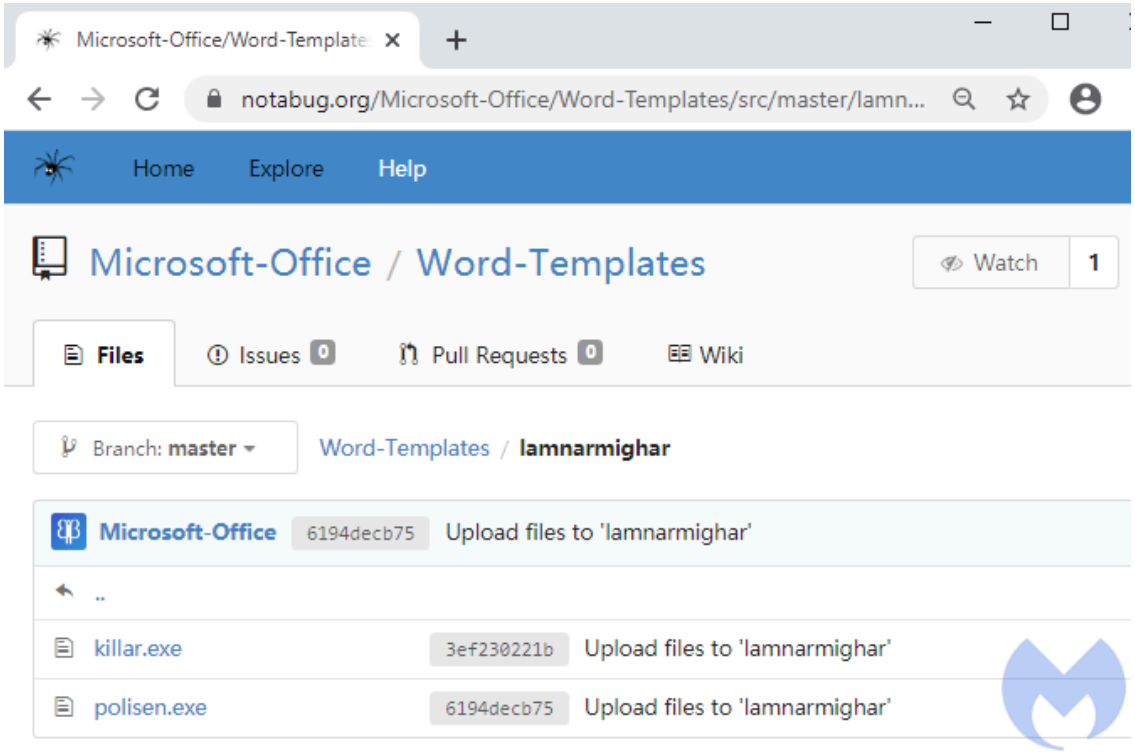
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



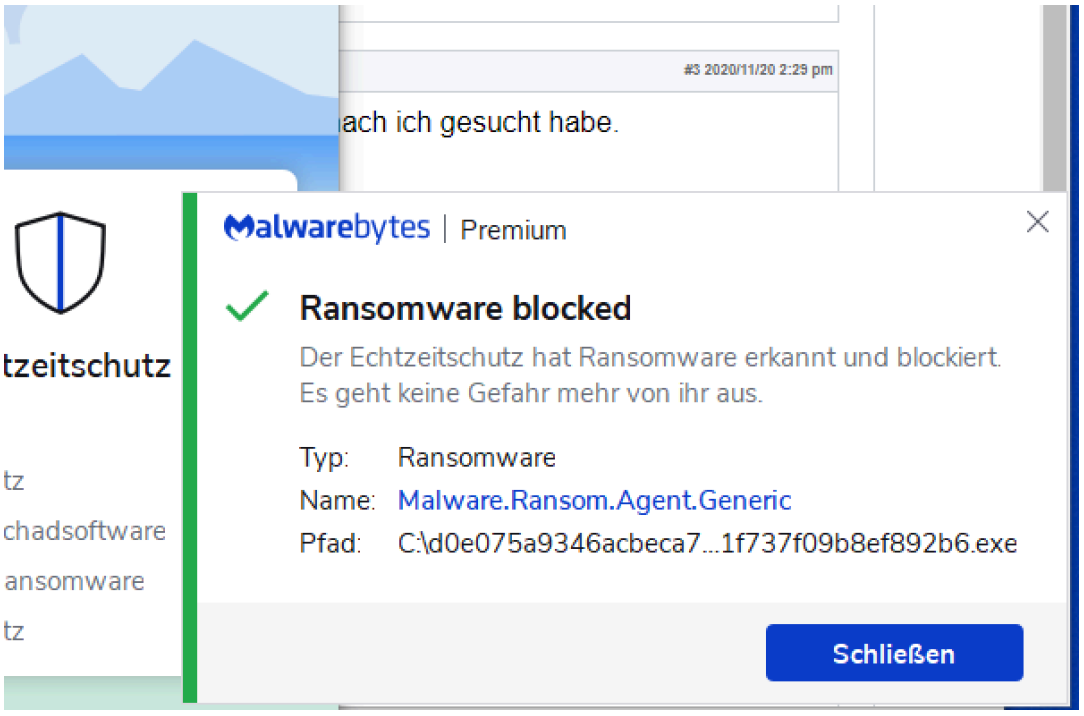
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

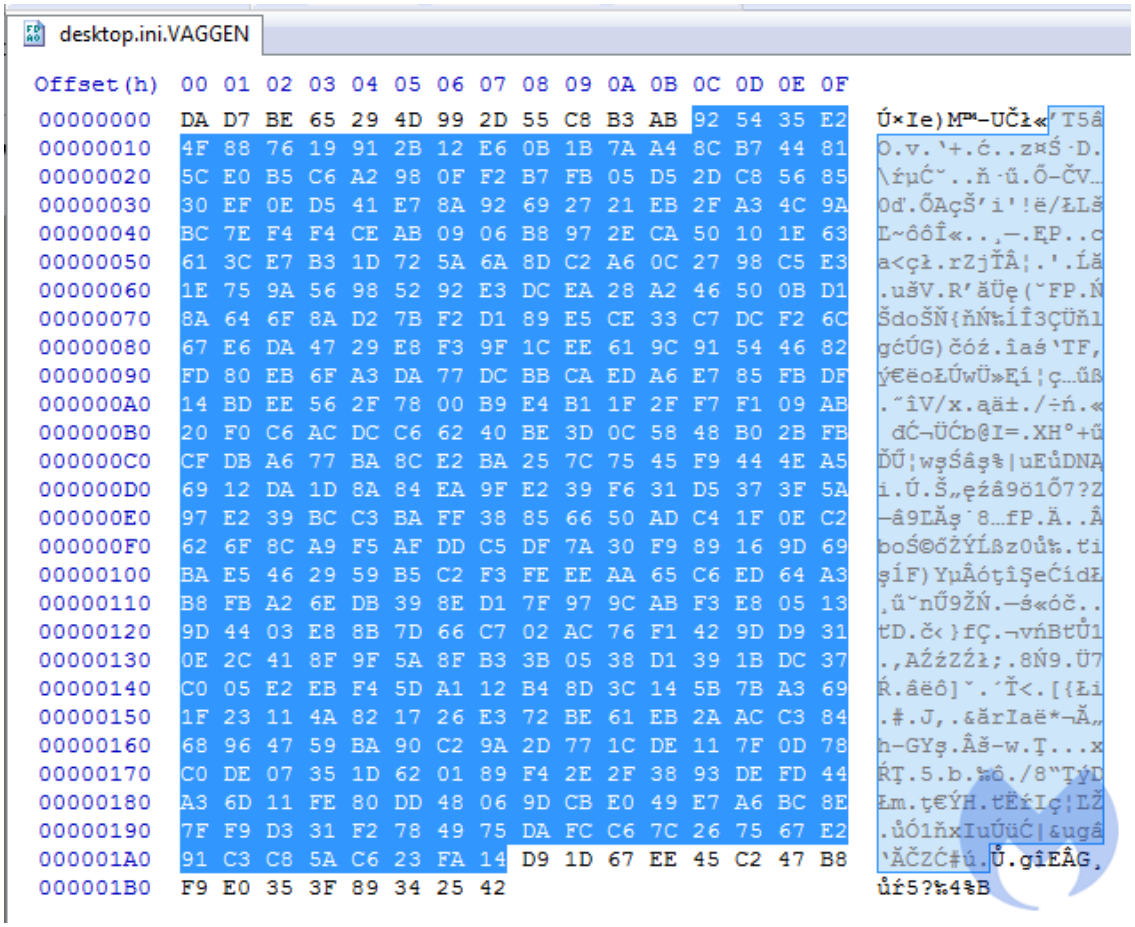
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

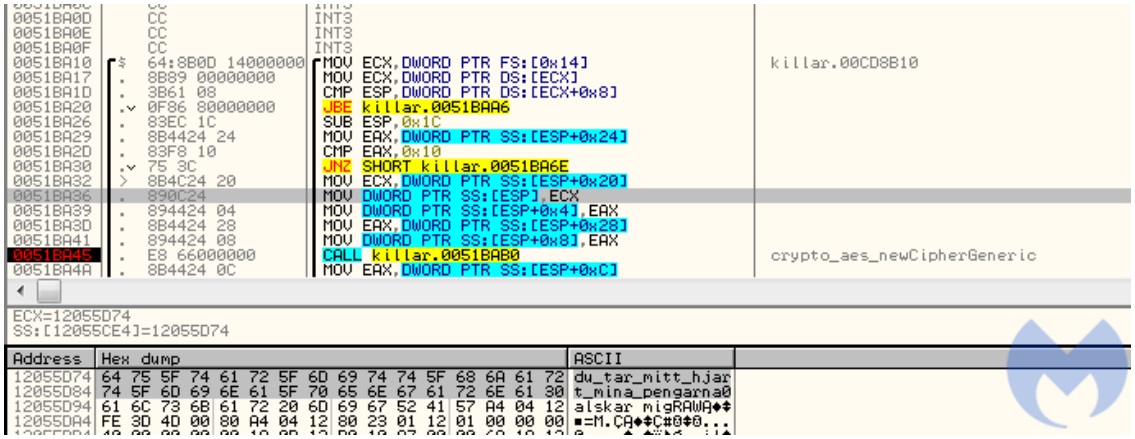


The content of the output file (with .VAGGEN extension) contains:

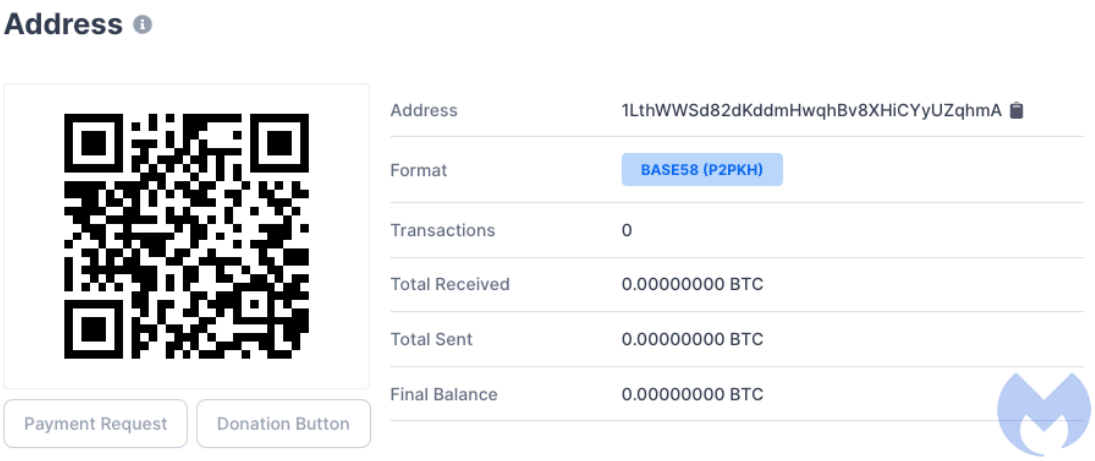
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

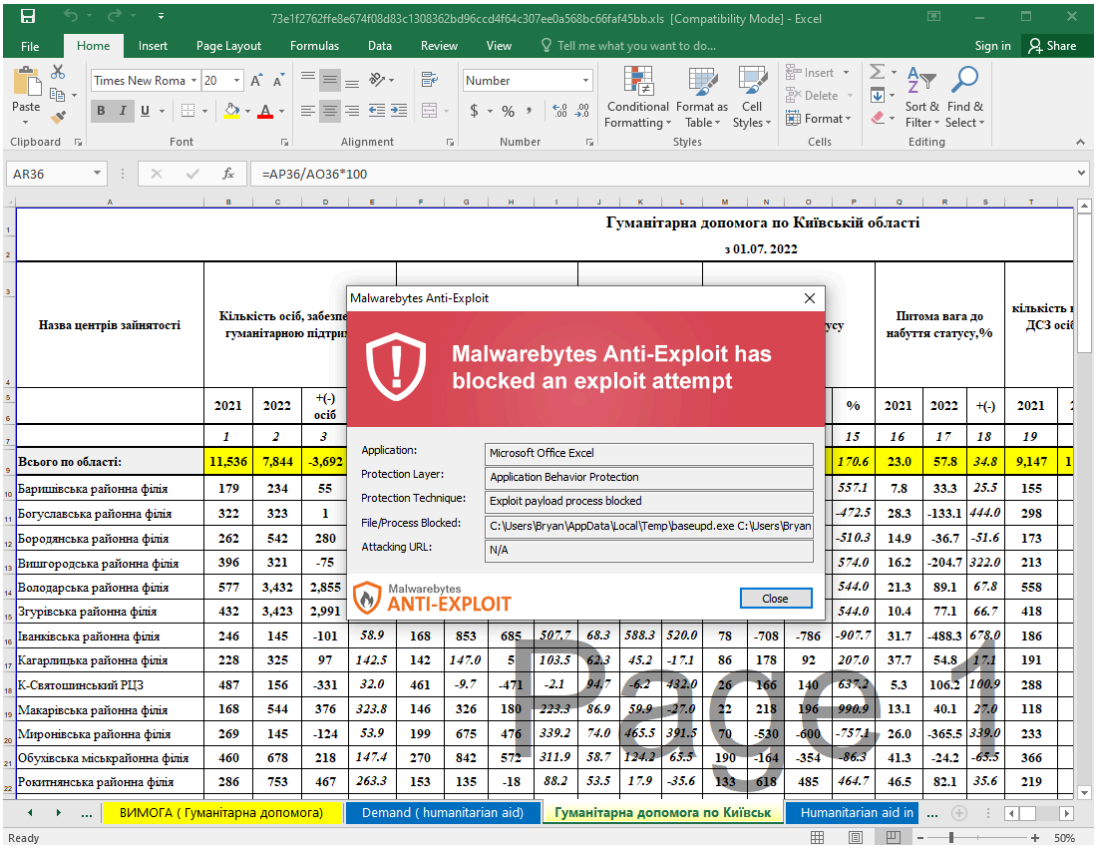
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

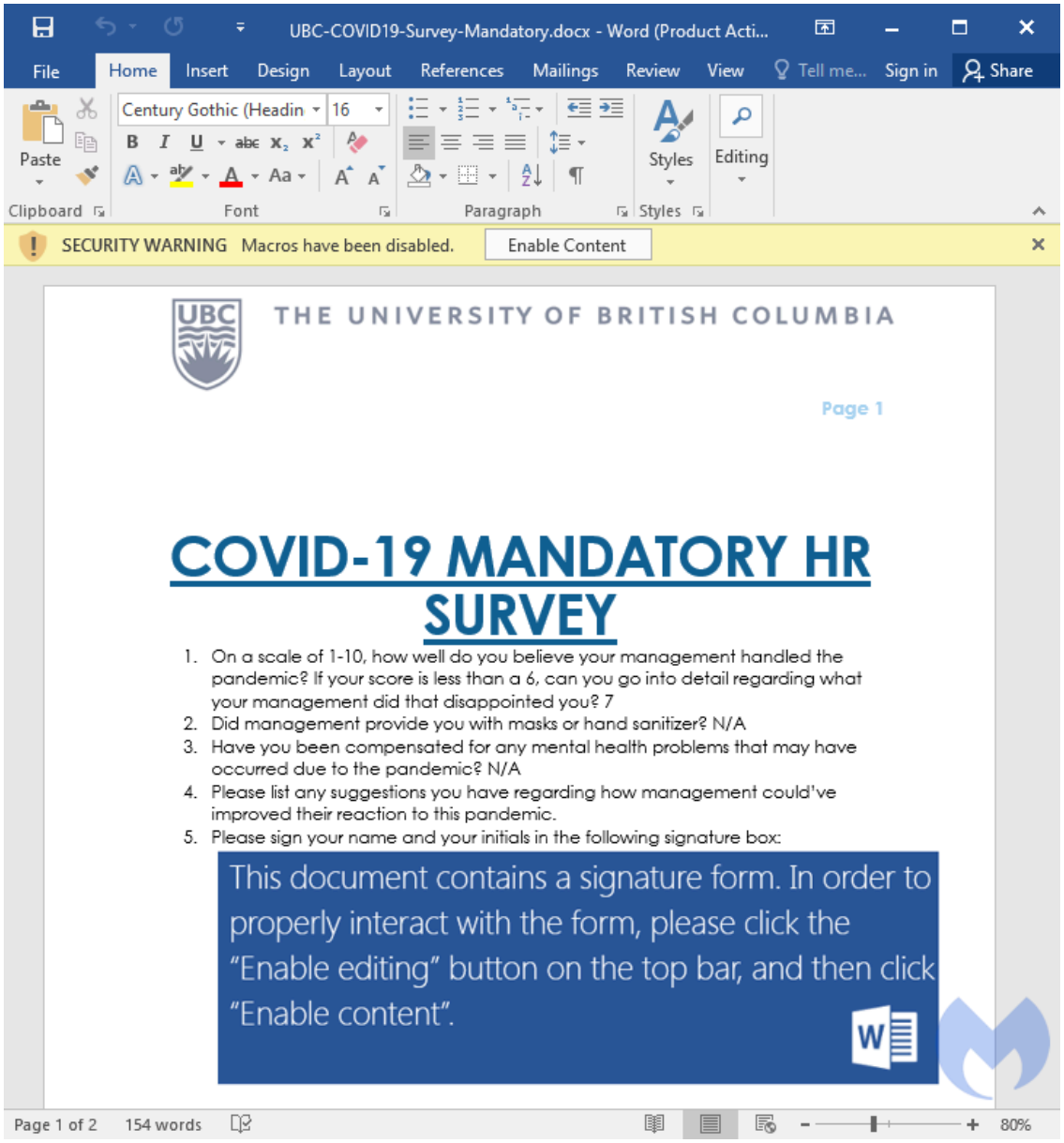
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

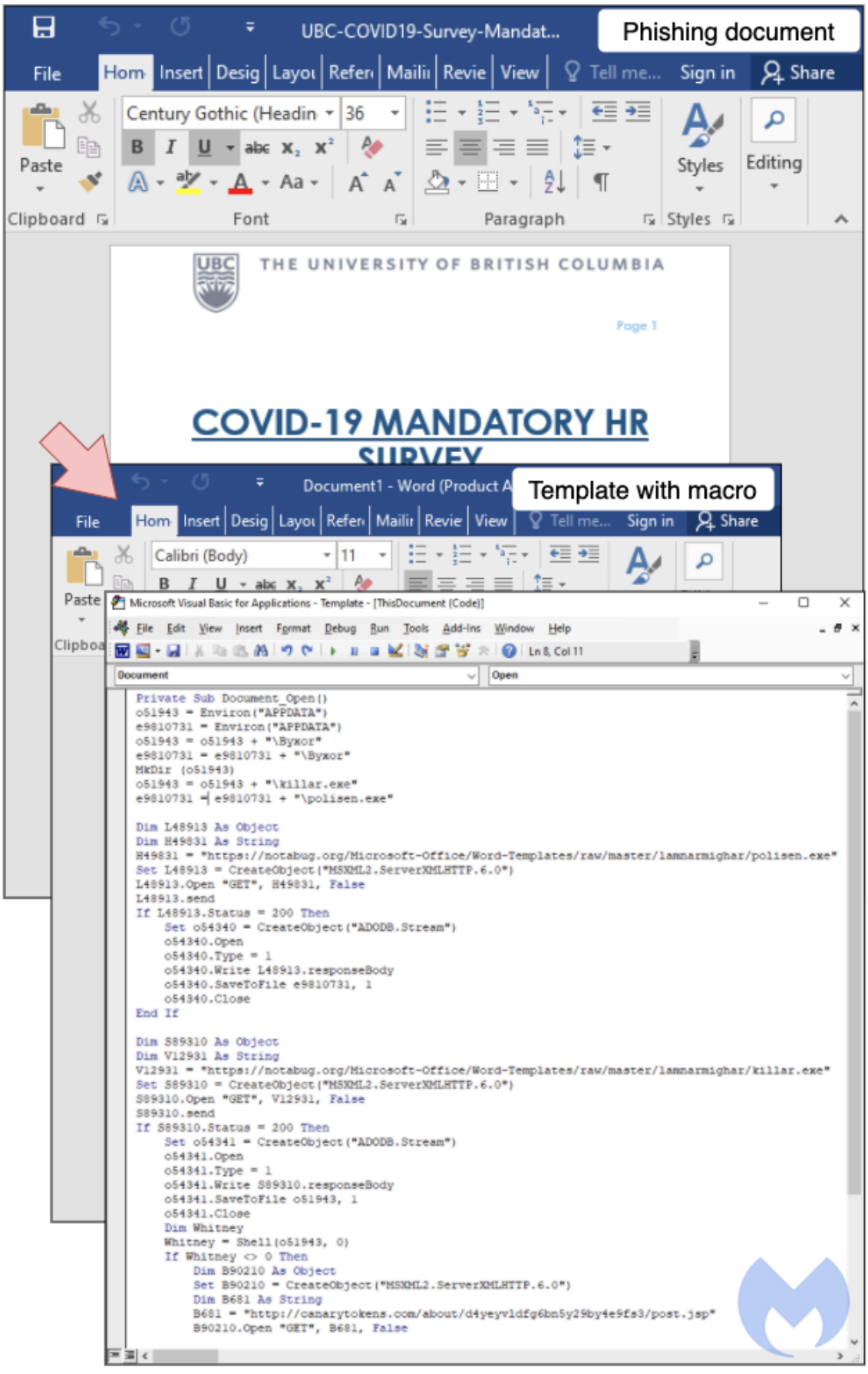
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

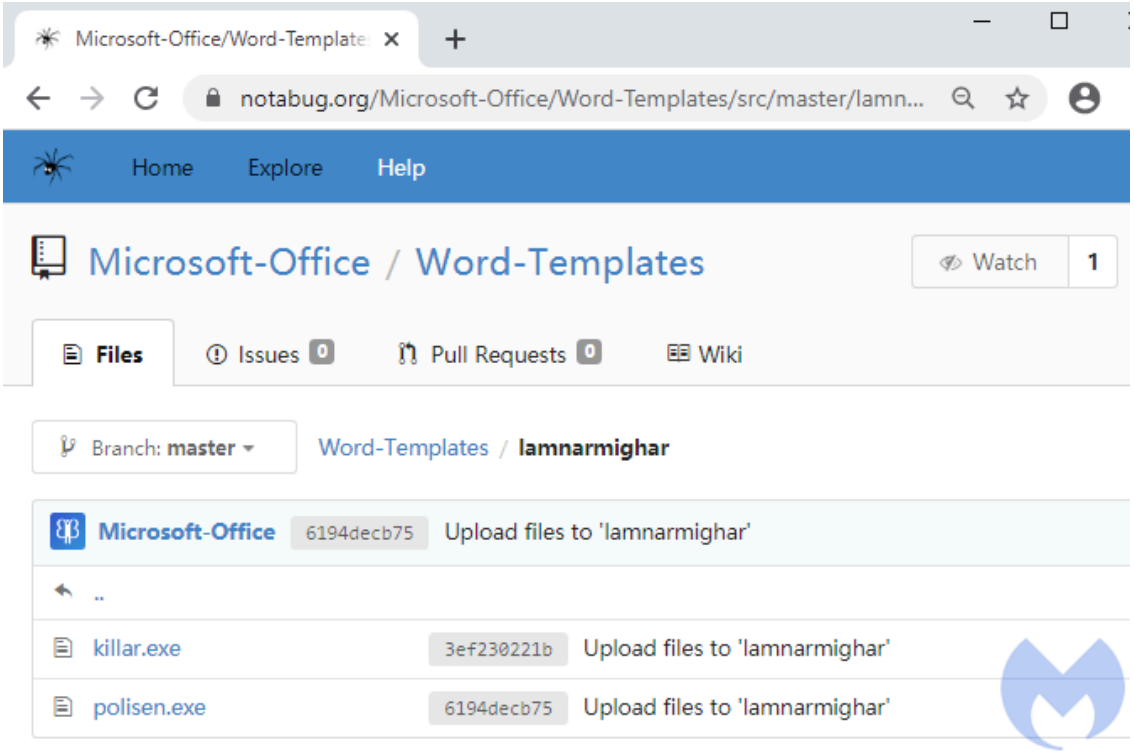
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



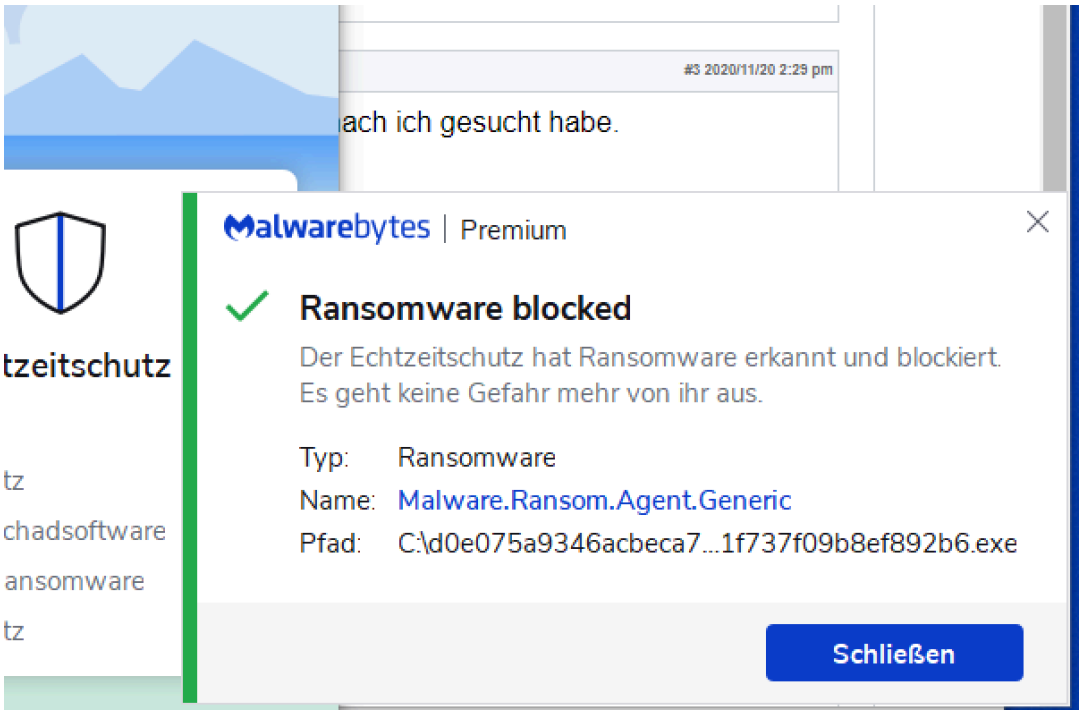
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

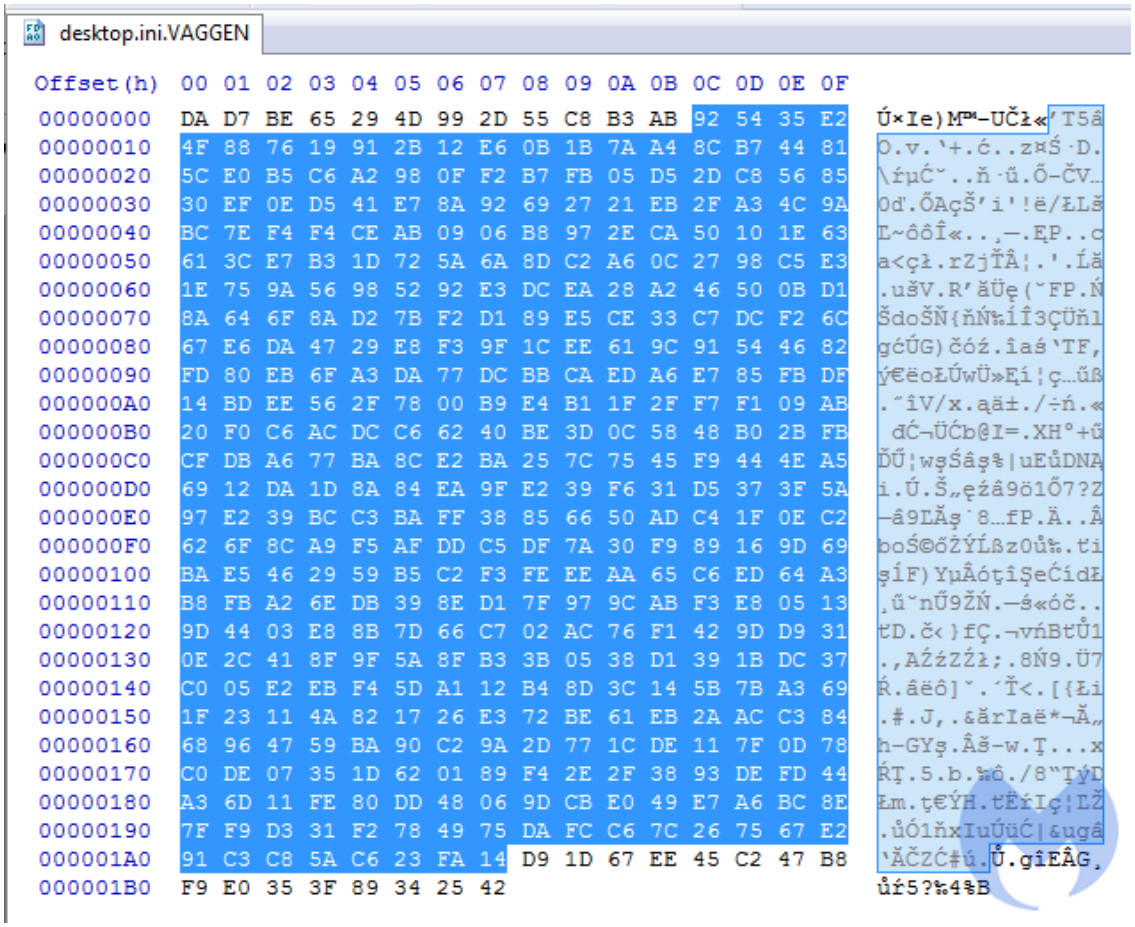
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



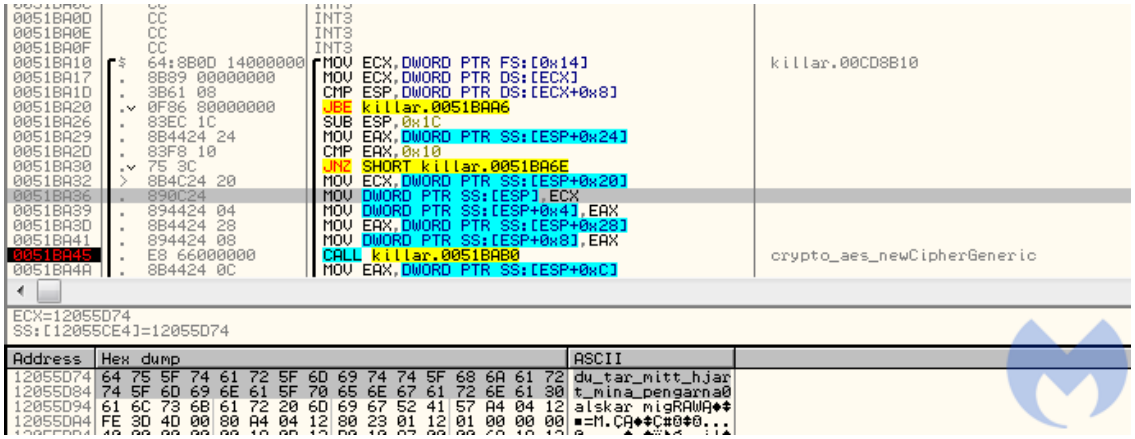
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

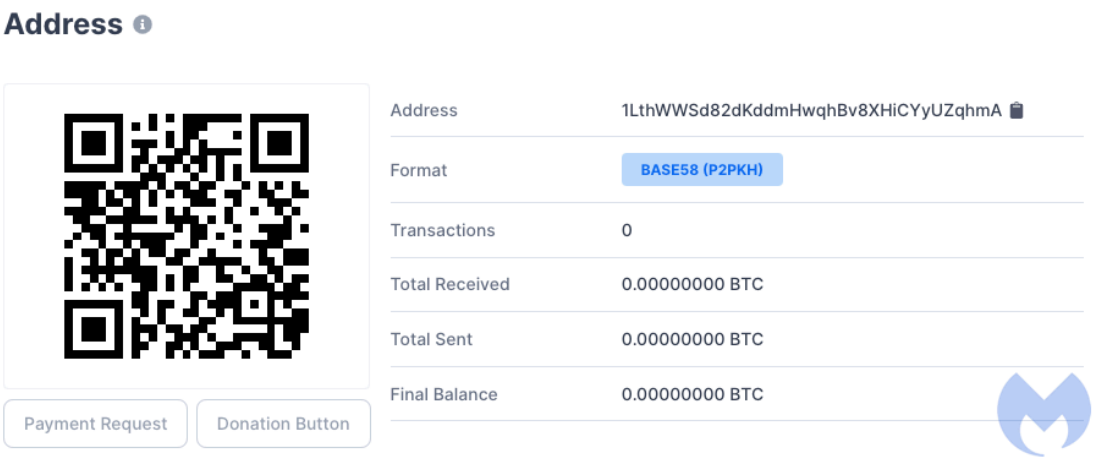
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

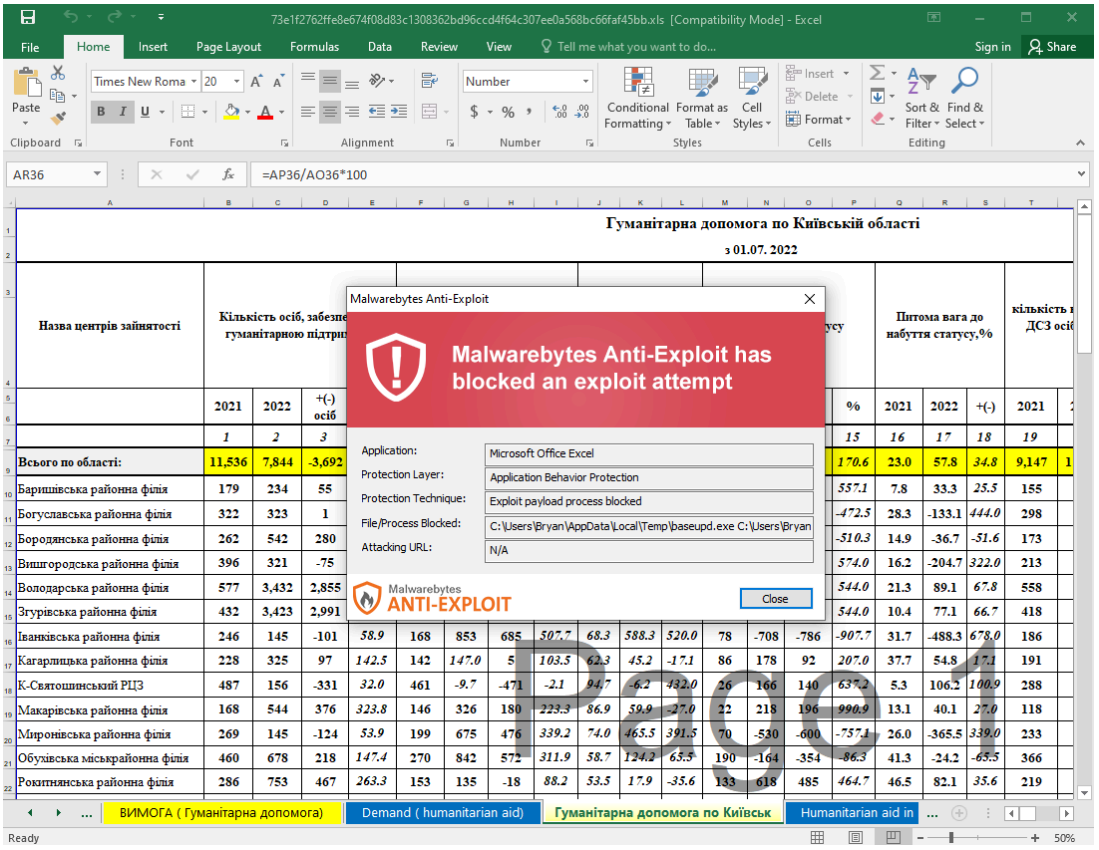
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

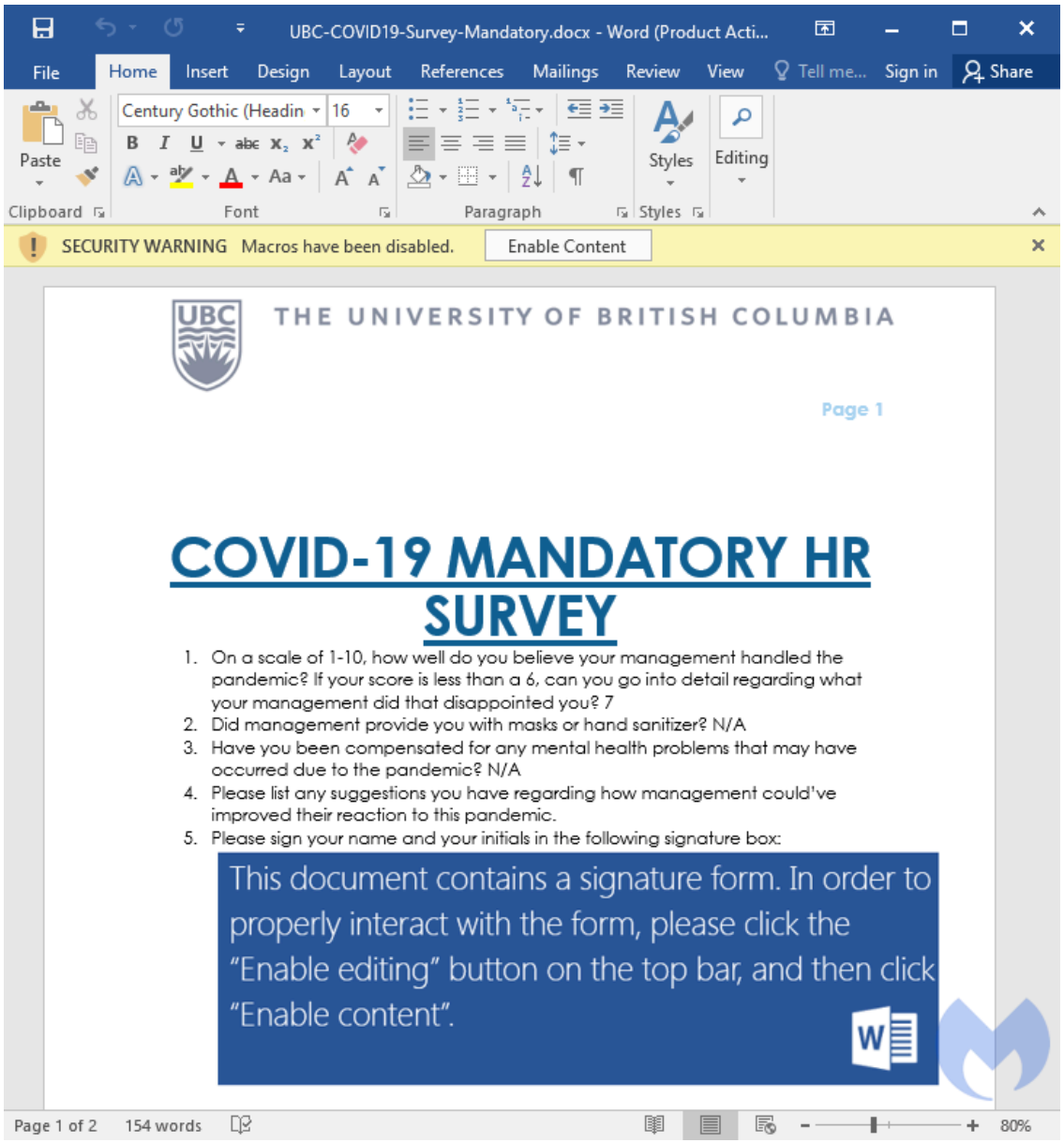
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

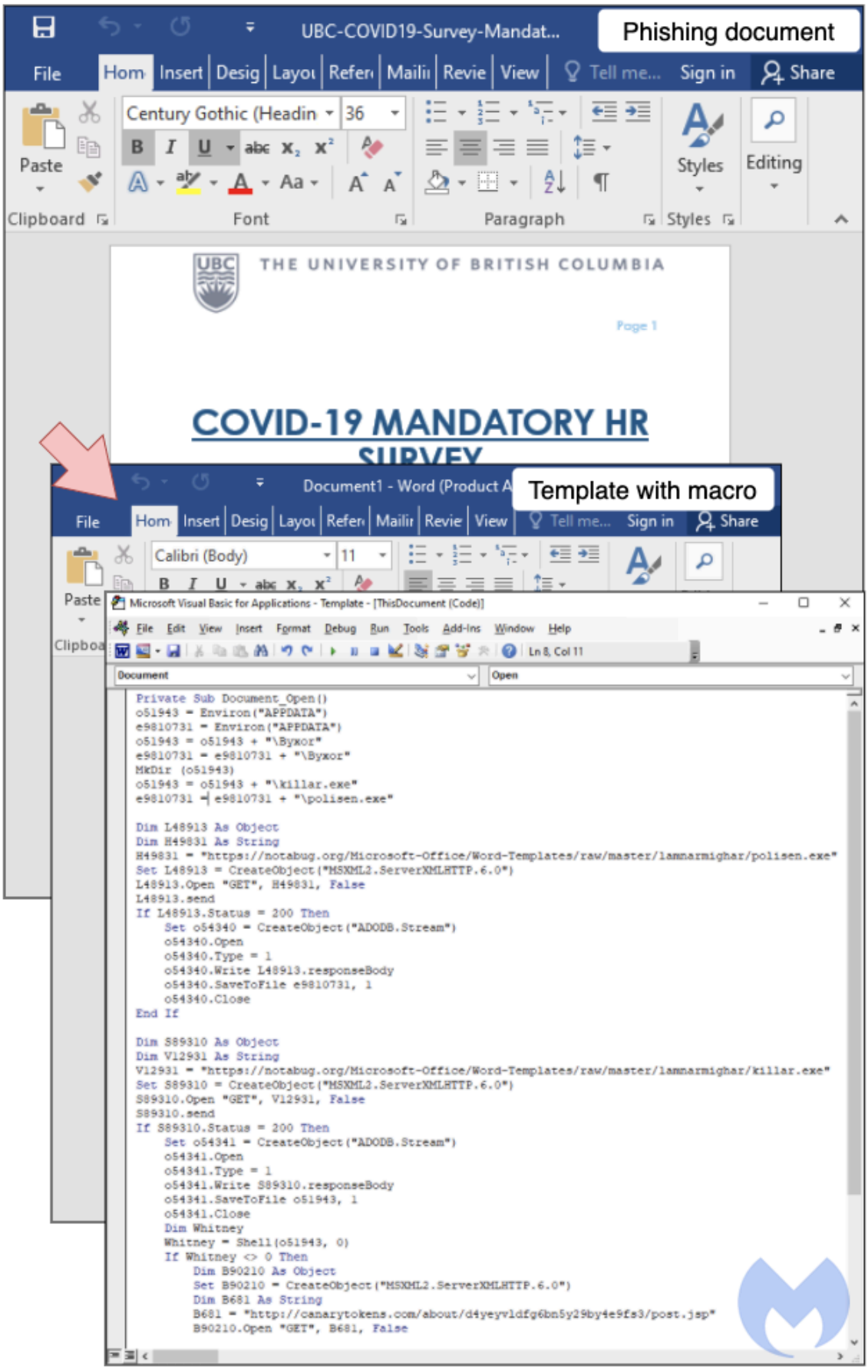
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

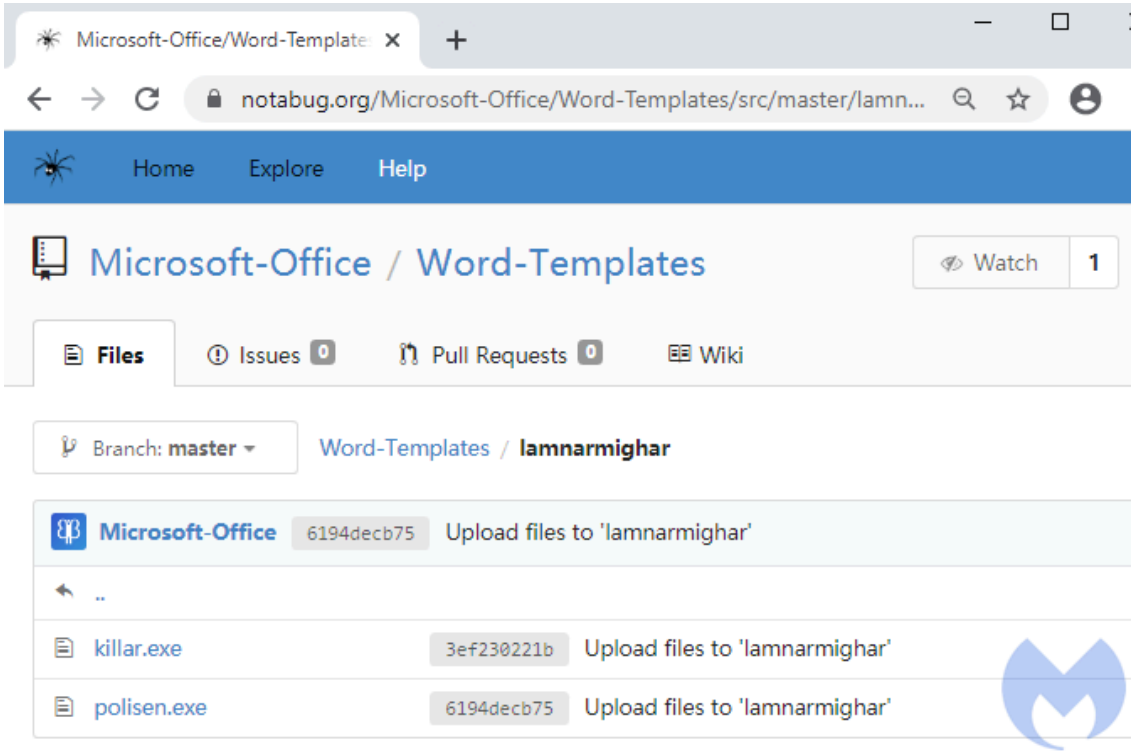
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



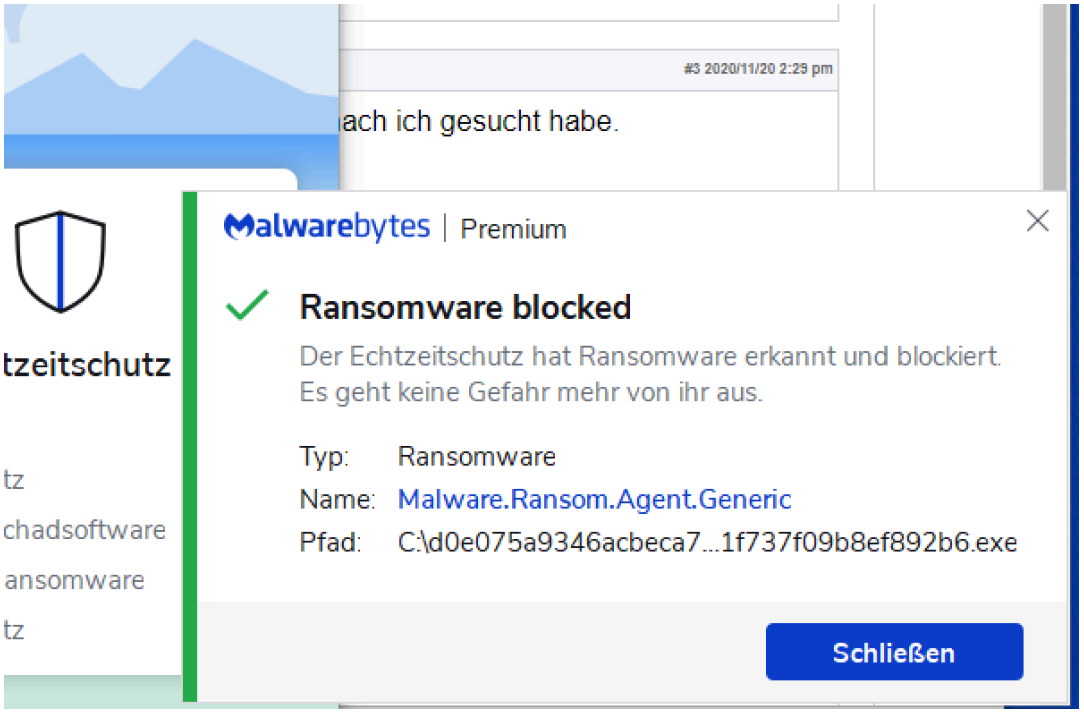
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

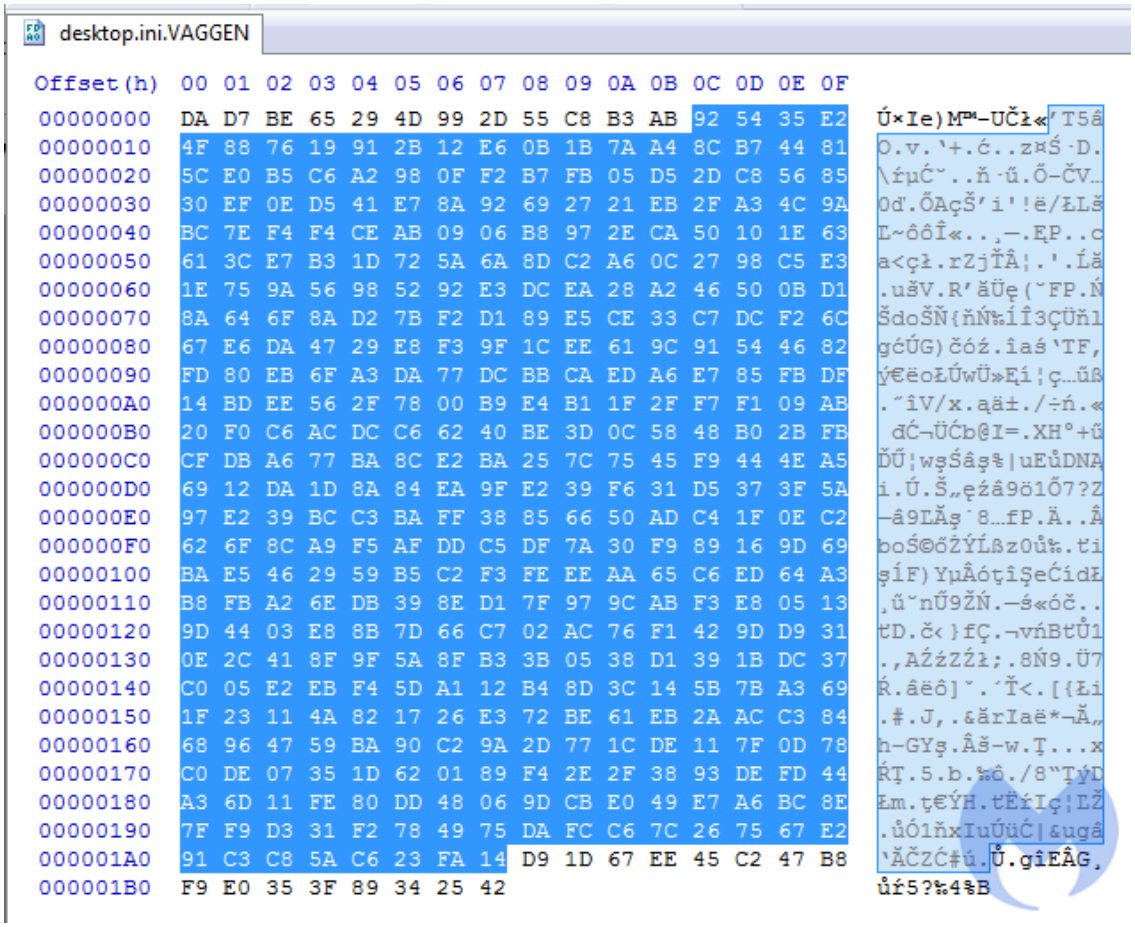
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

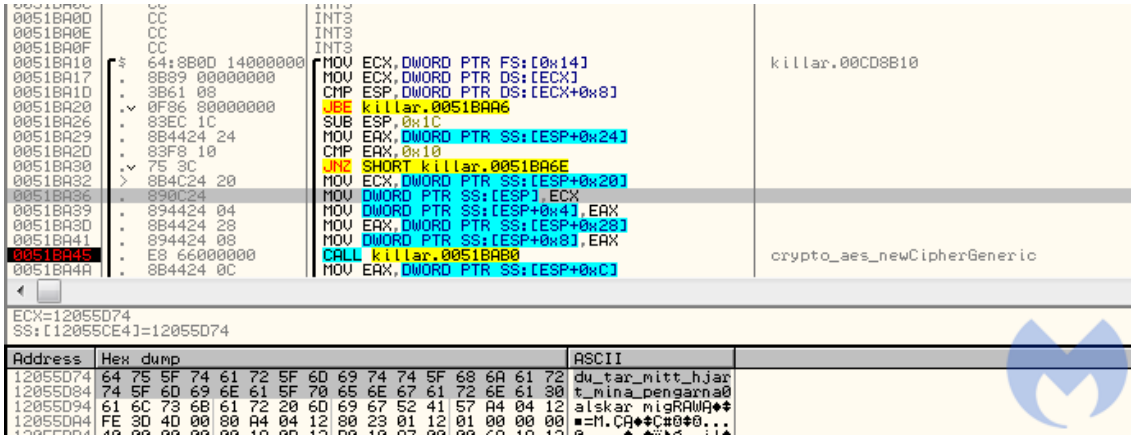


The content of the output file (with .VAGGEN extension) contains:

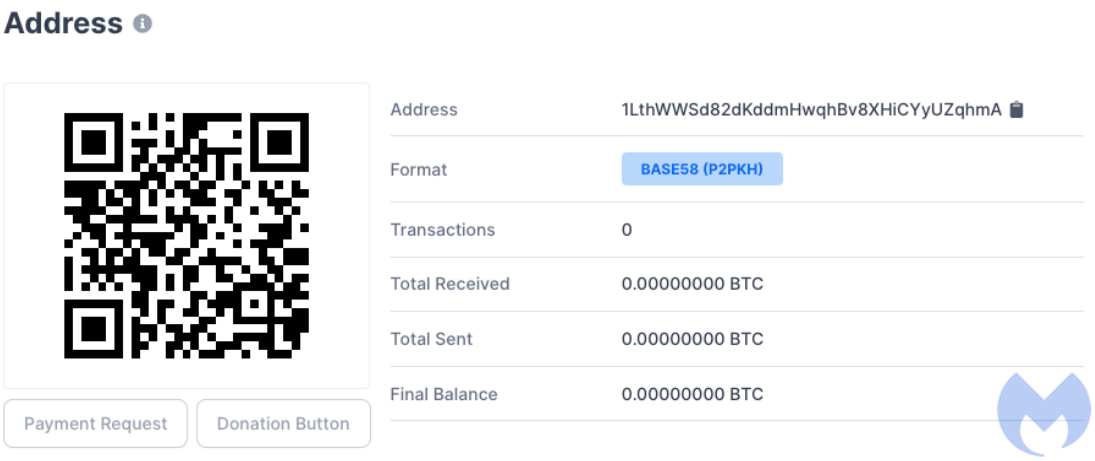
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

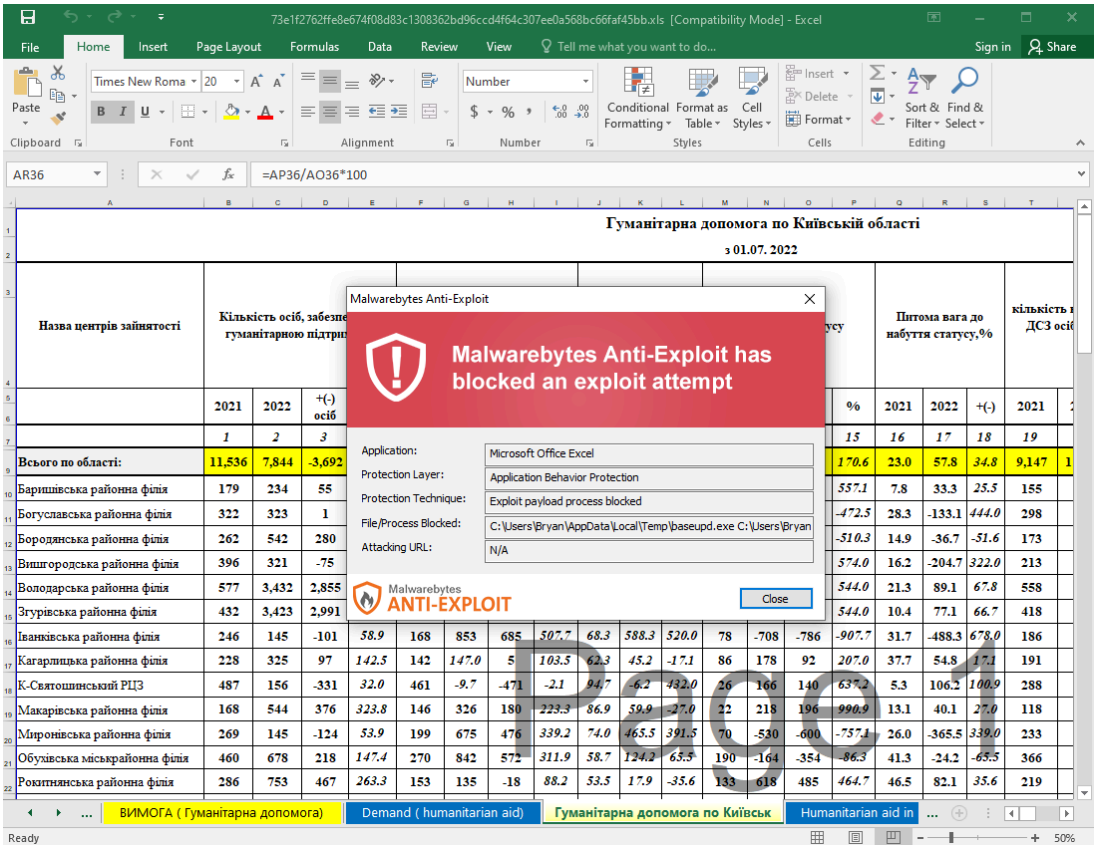
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

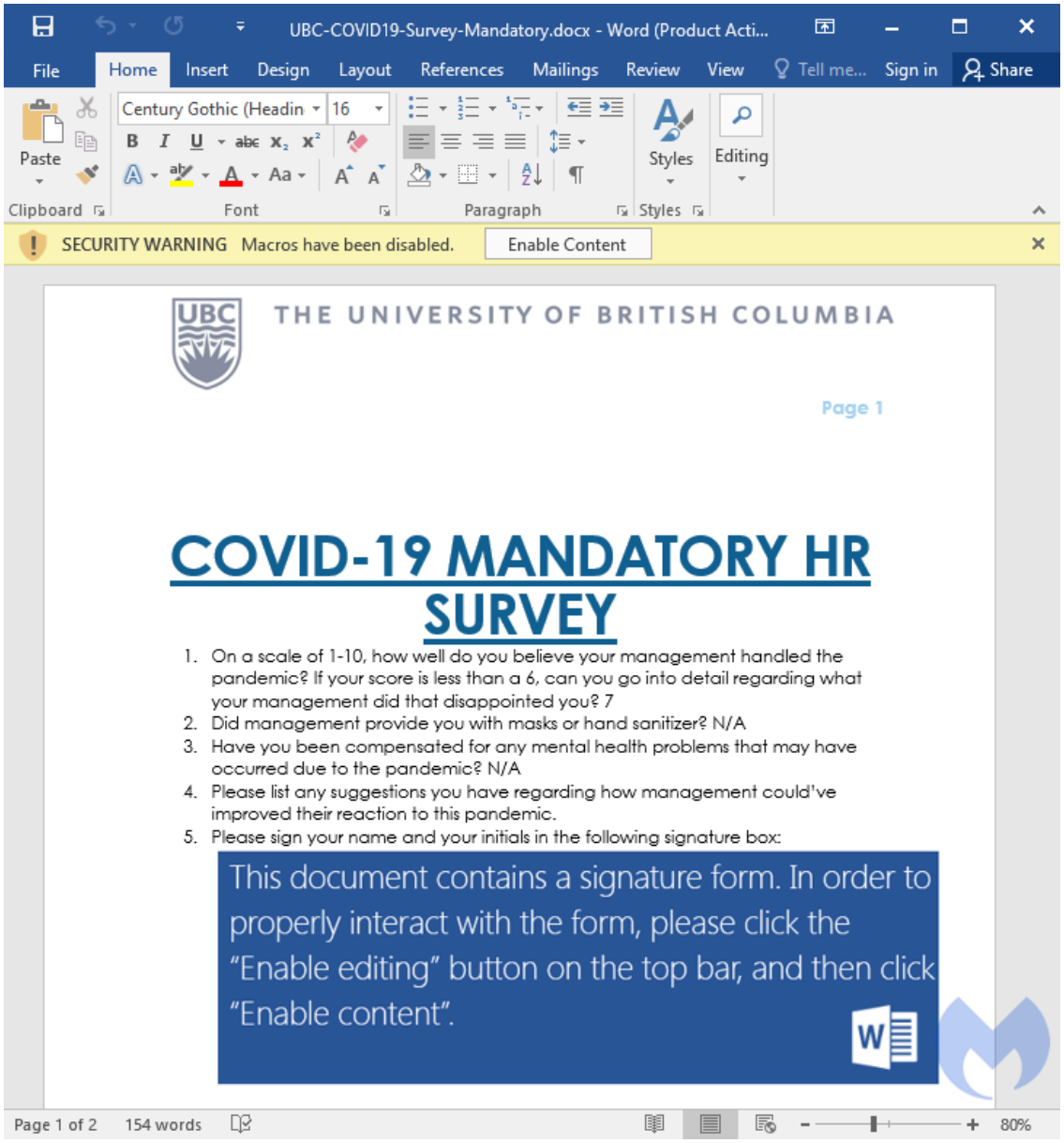
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

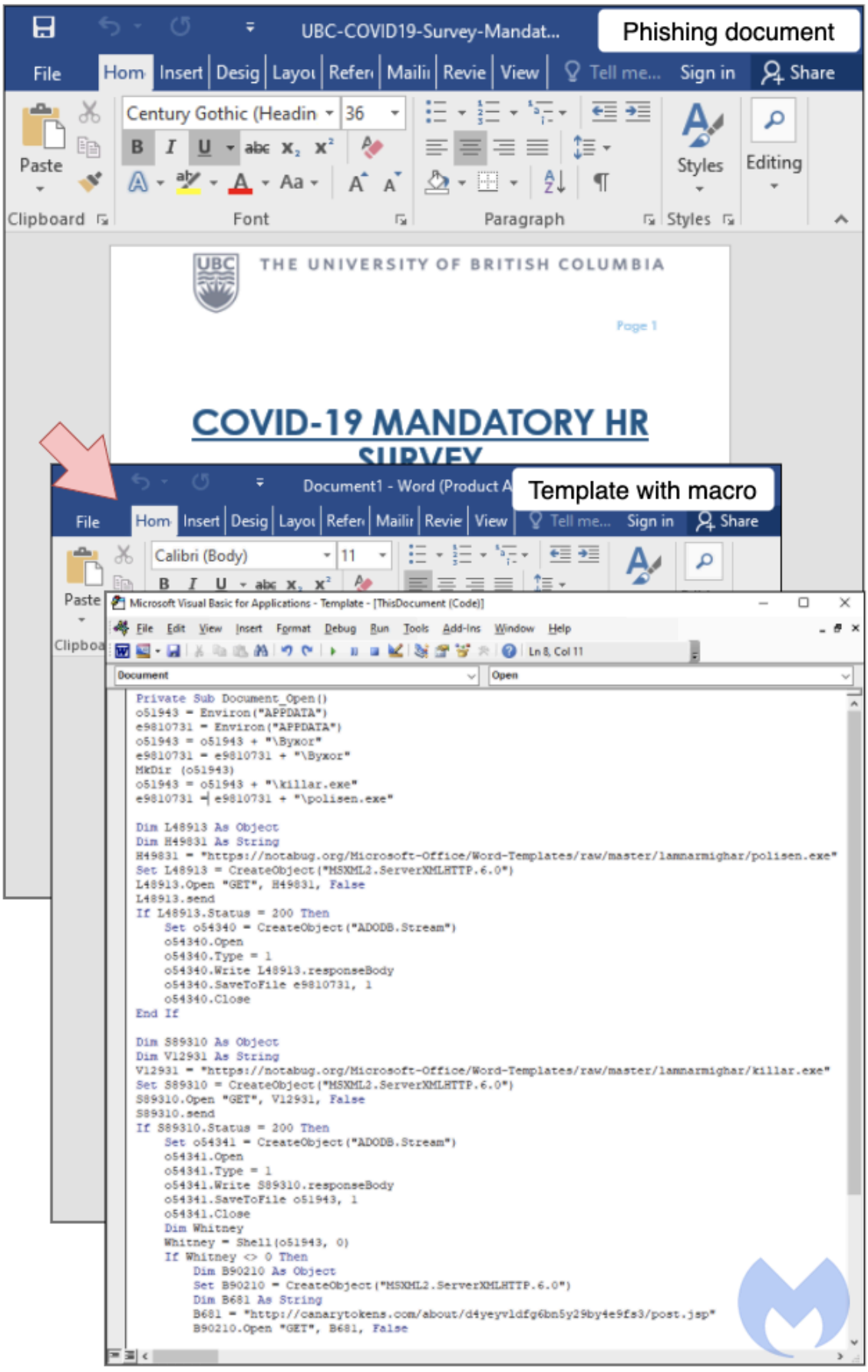
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

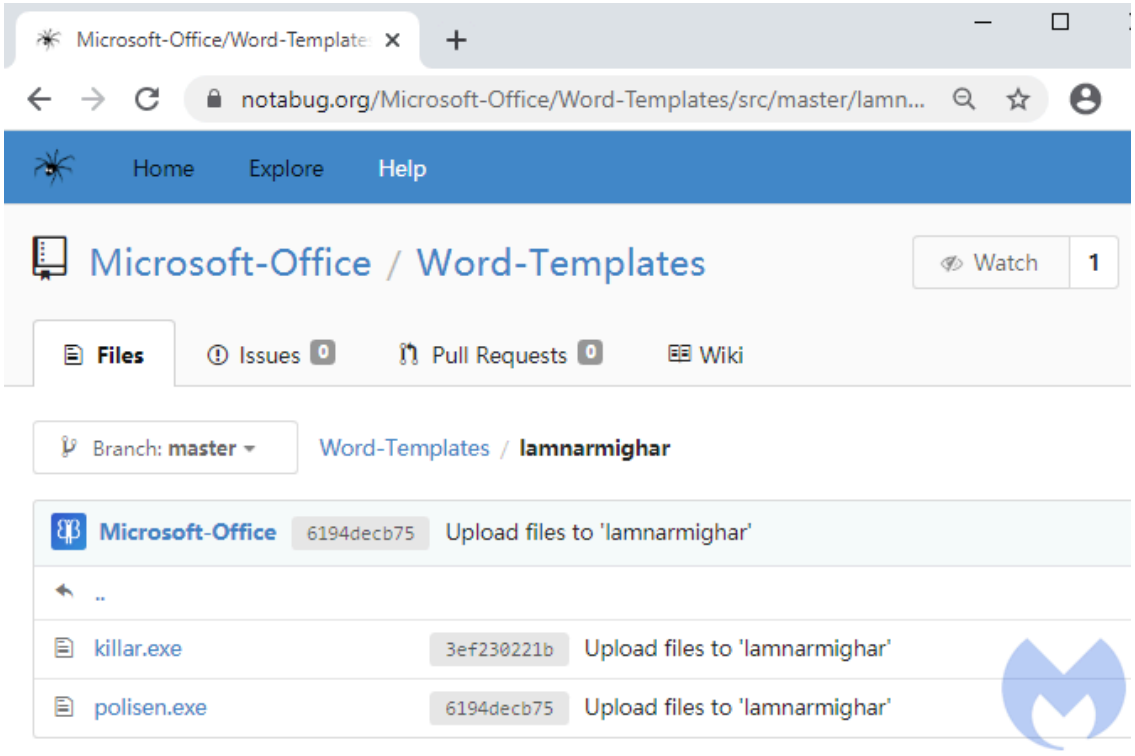
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



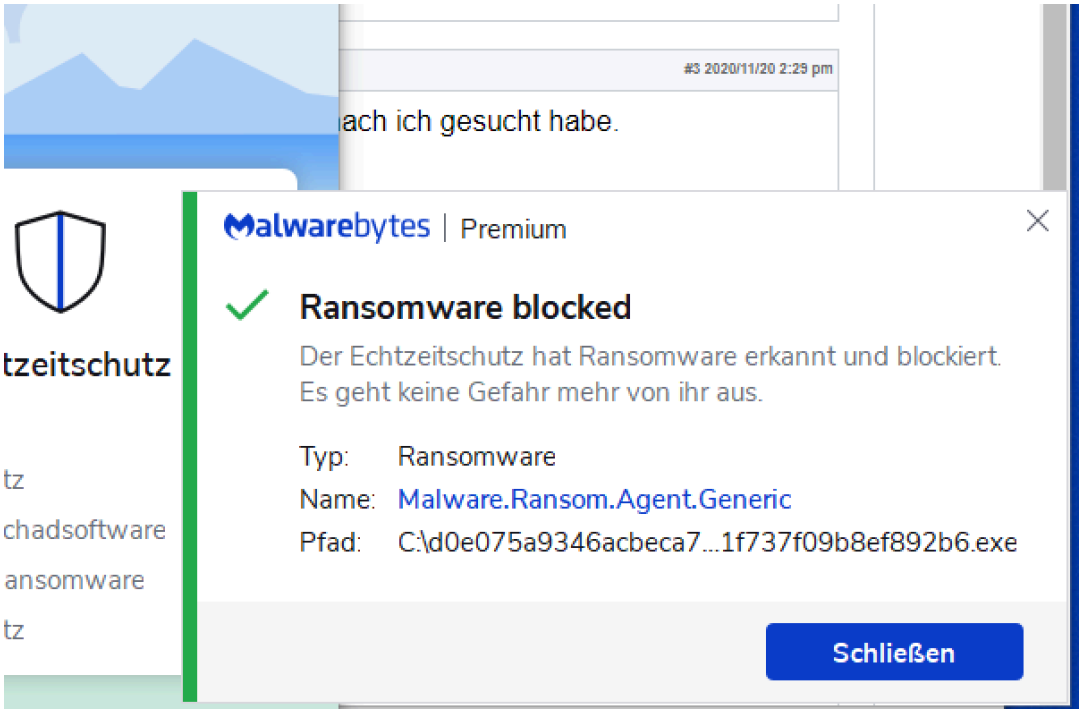
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

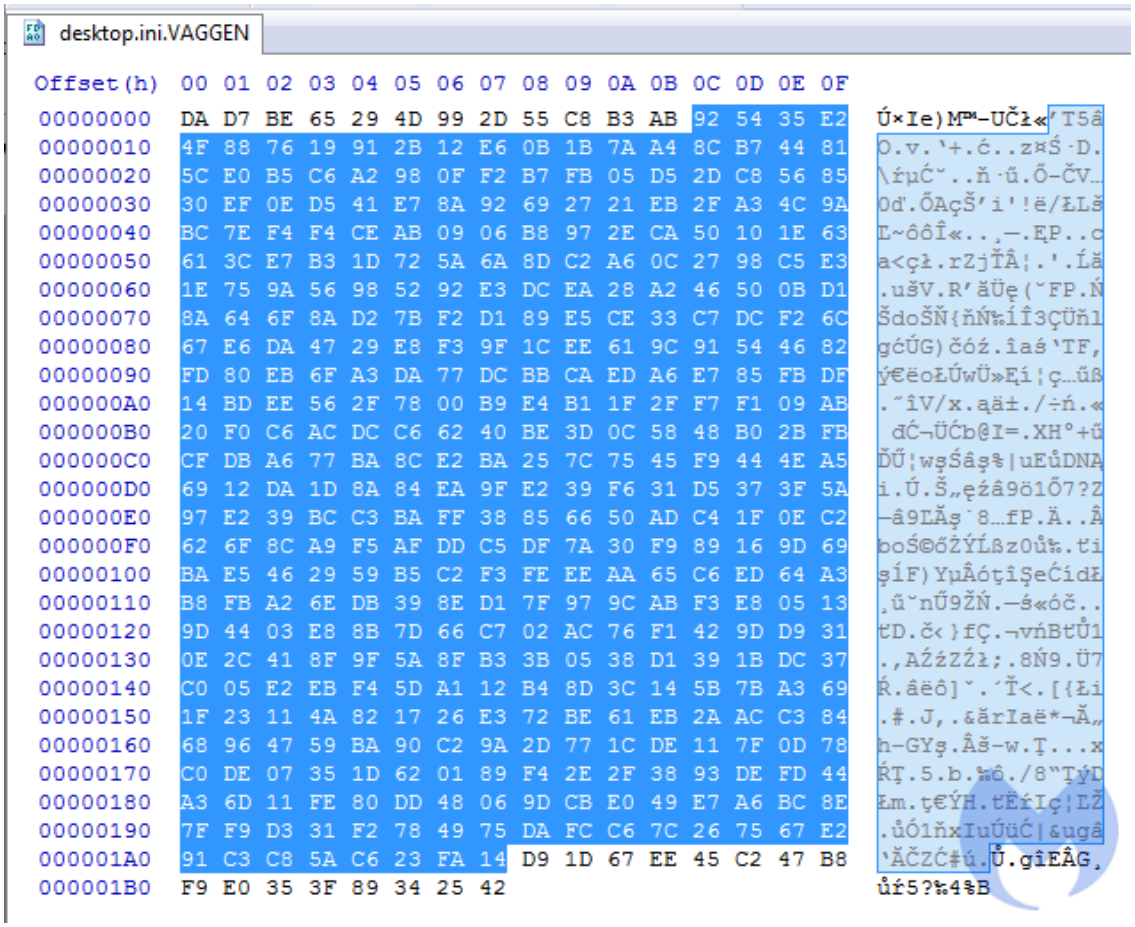
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

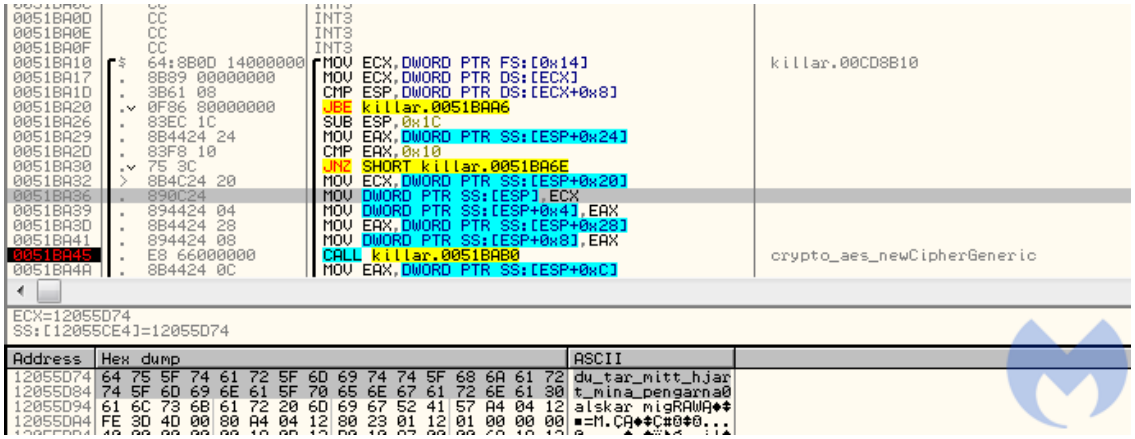


The content of the output file (with .VAGGEN extension) contains:

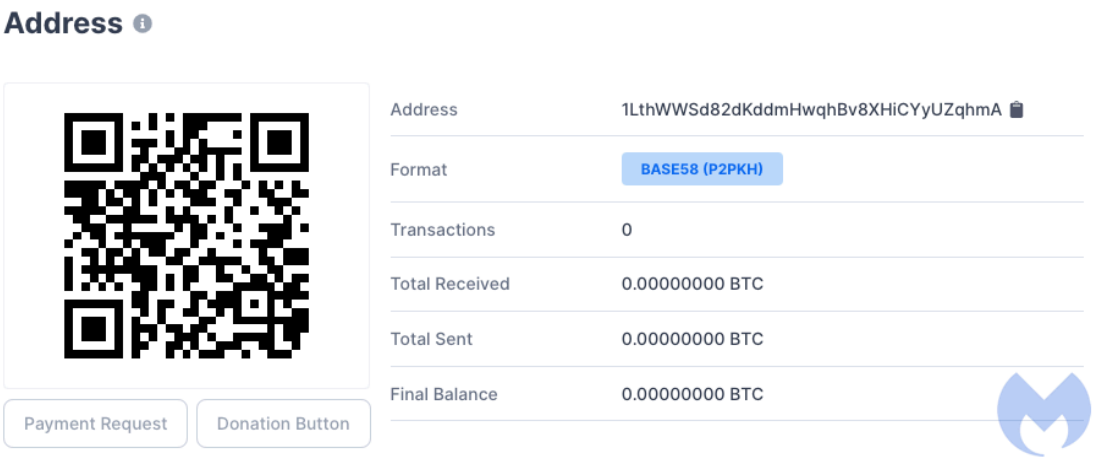
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

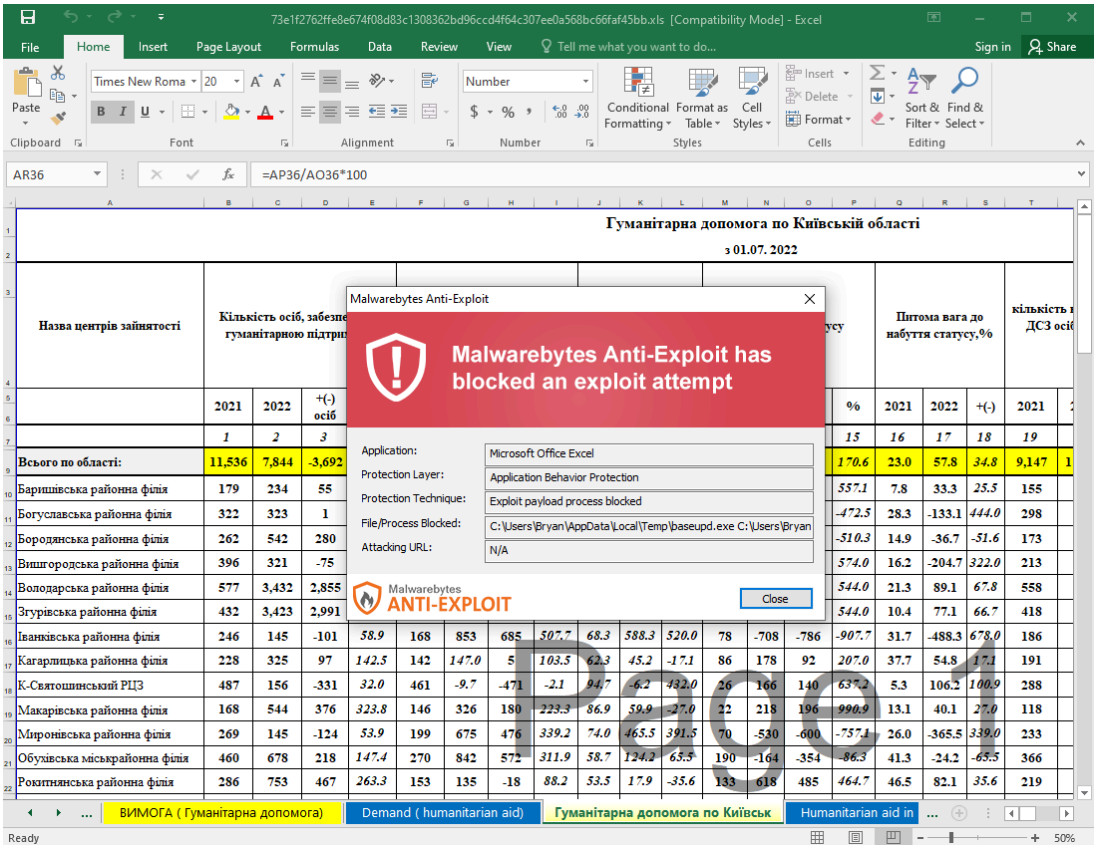
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

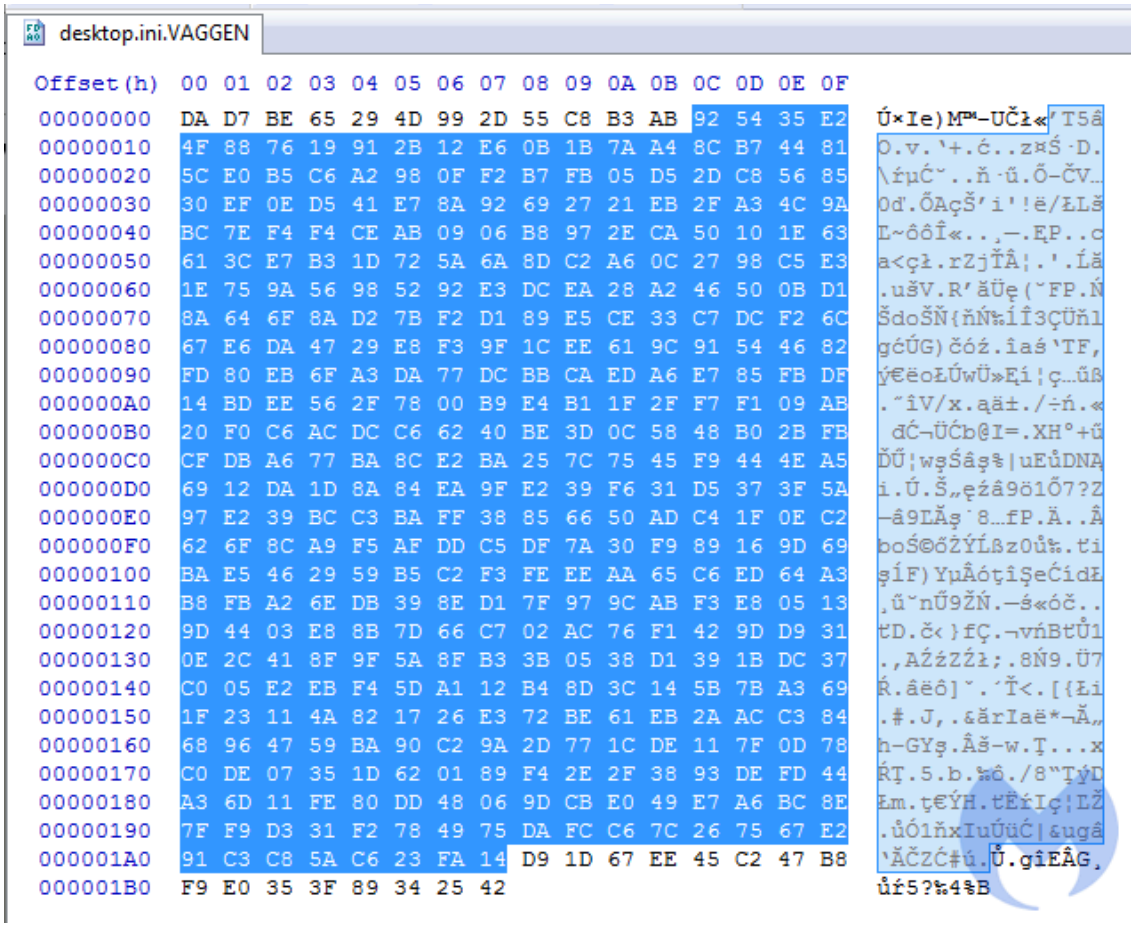
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

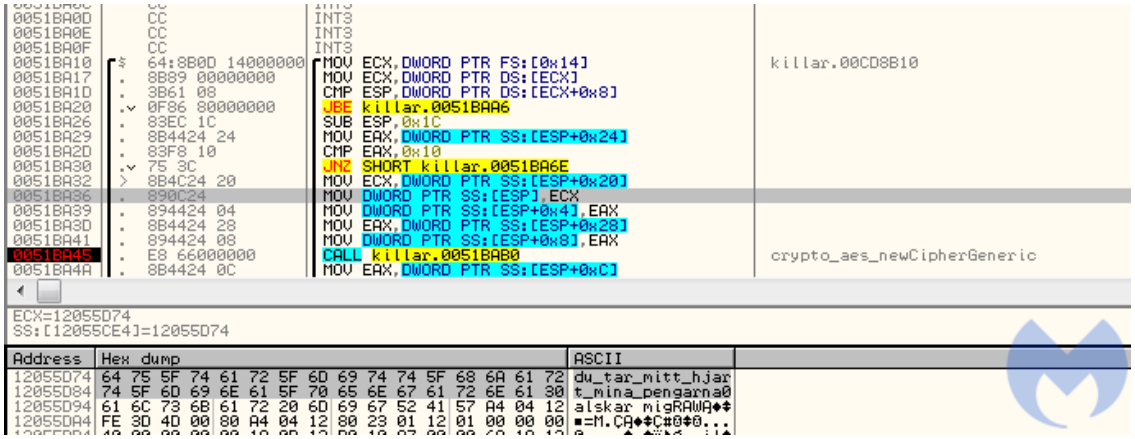


The content of the output file (with .VAGGEN extension) contains:

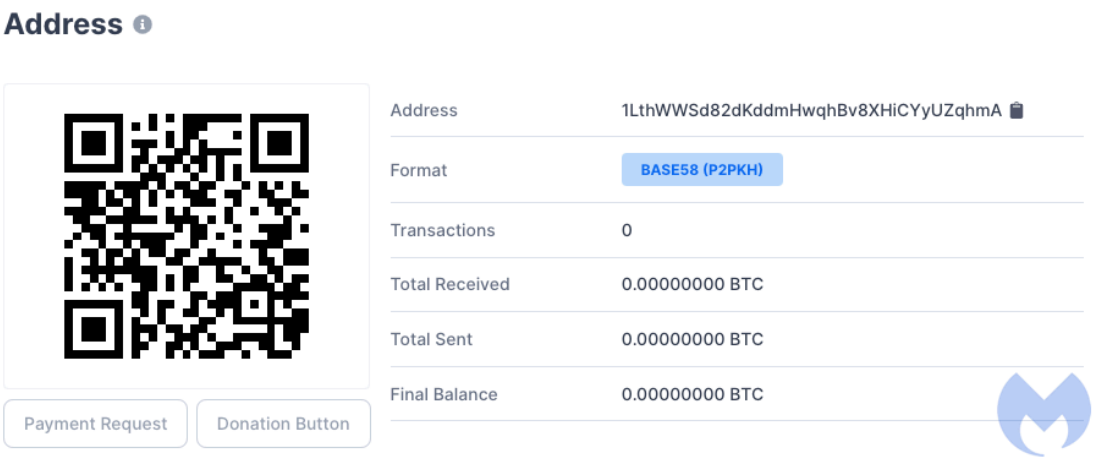
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

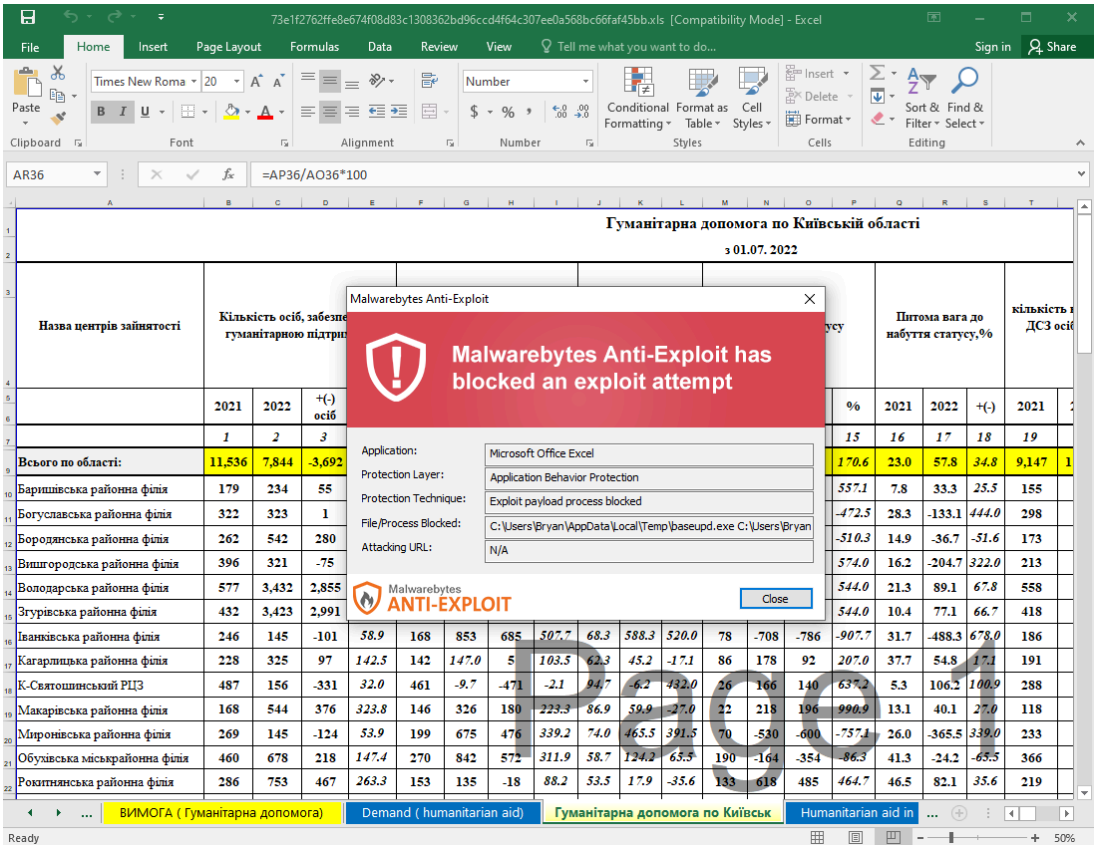
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

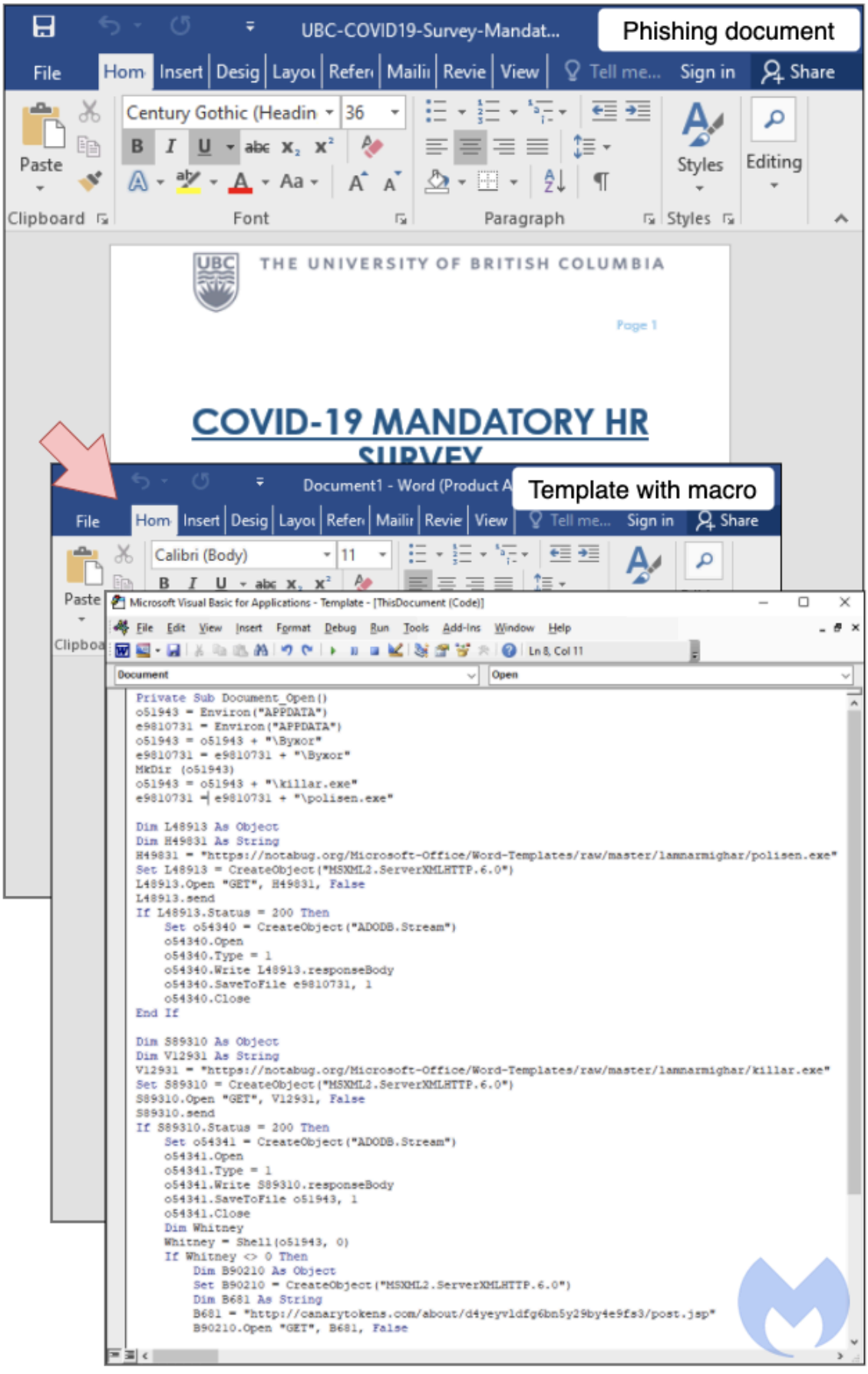
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

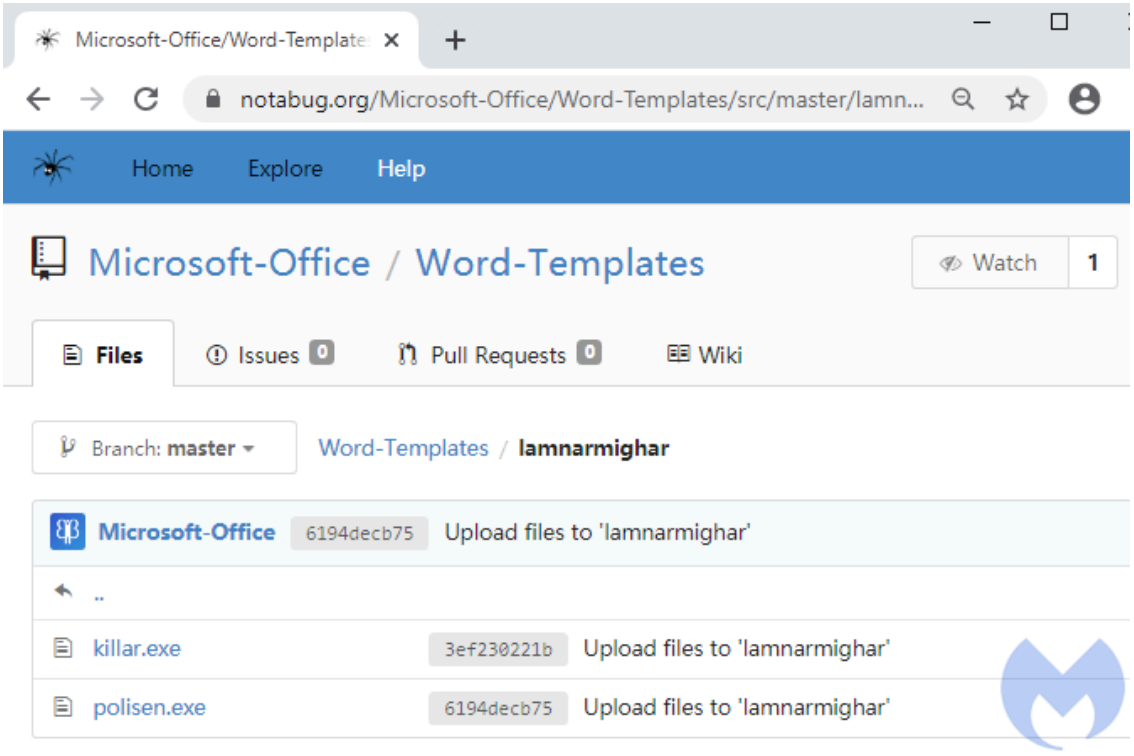
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



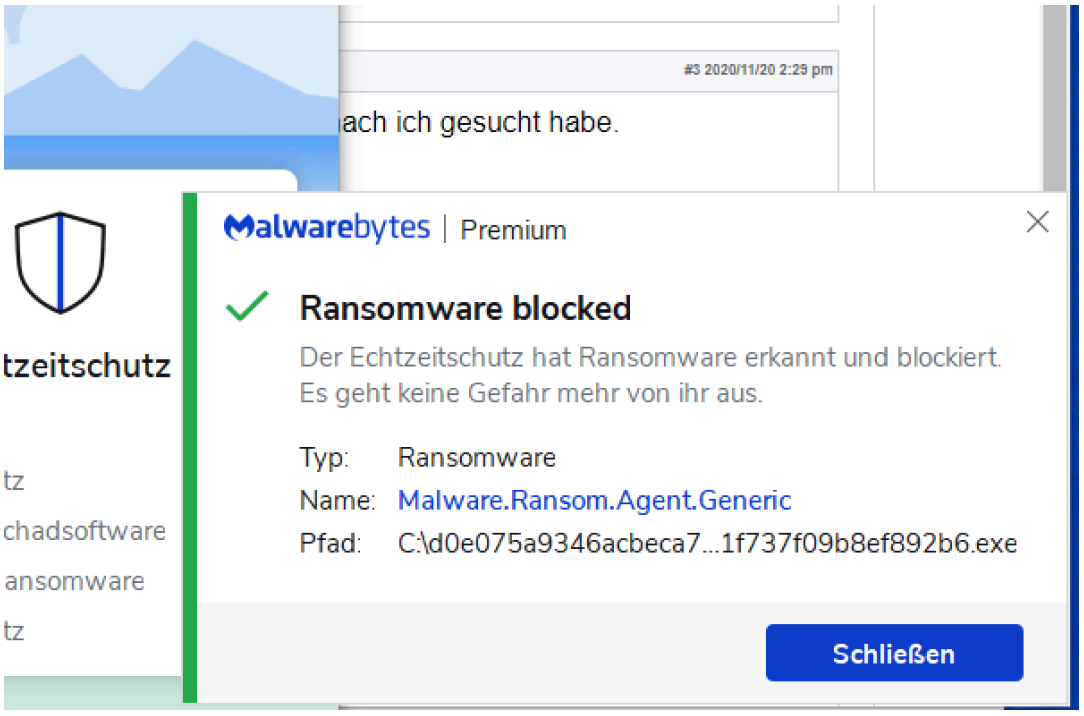
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

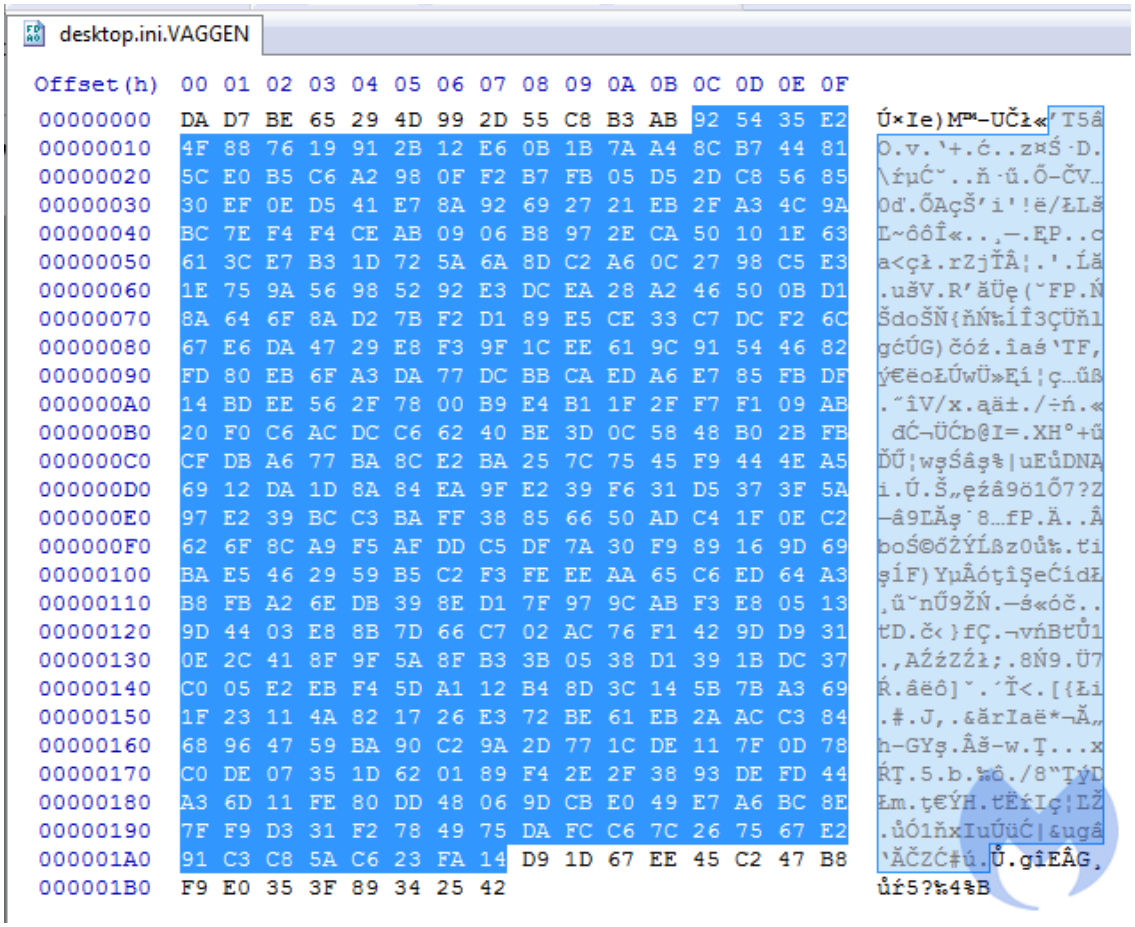
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

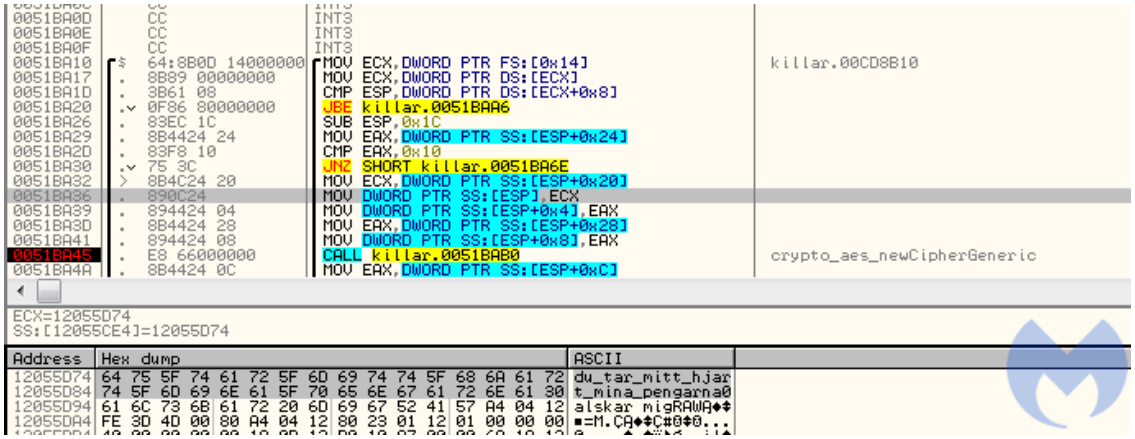


The content of the output file (with .VAGGEN extension) contains:

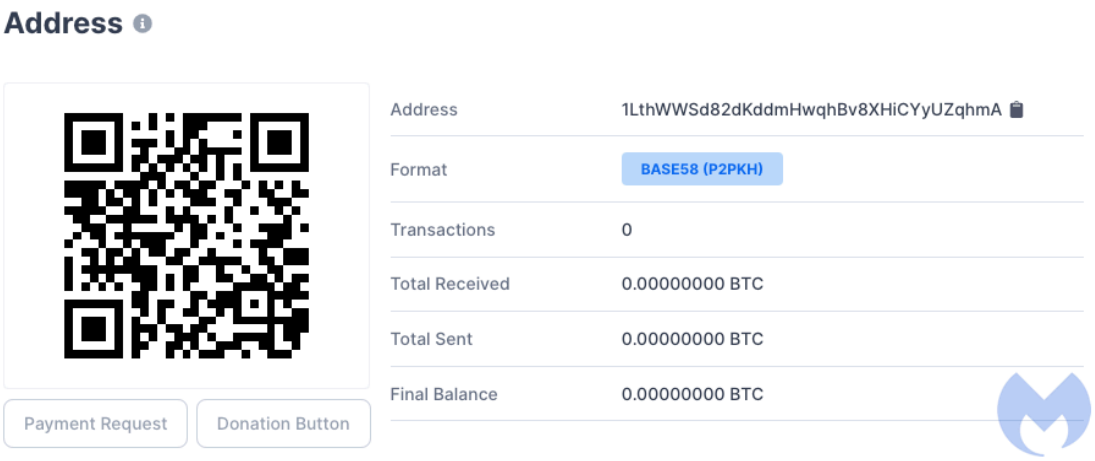
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

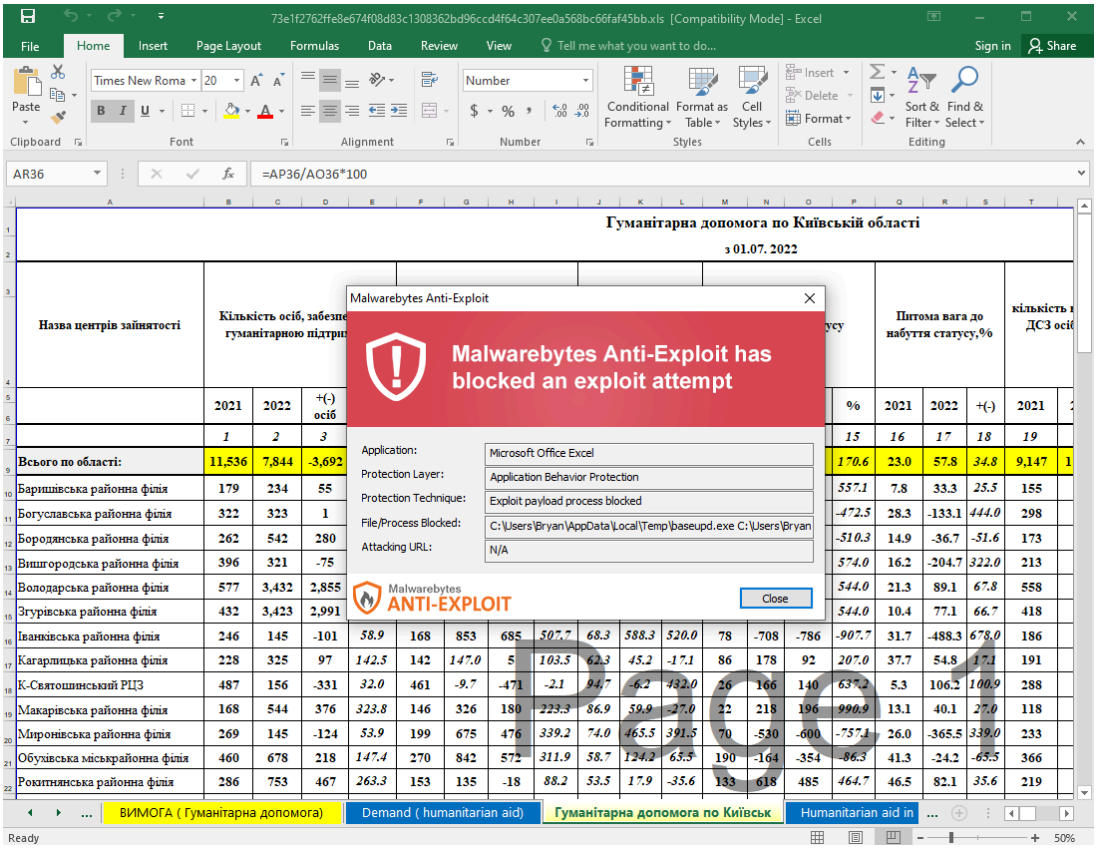
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

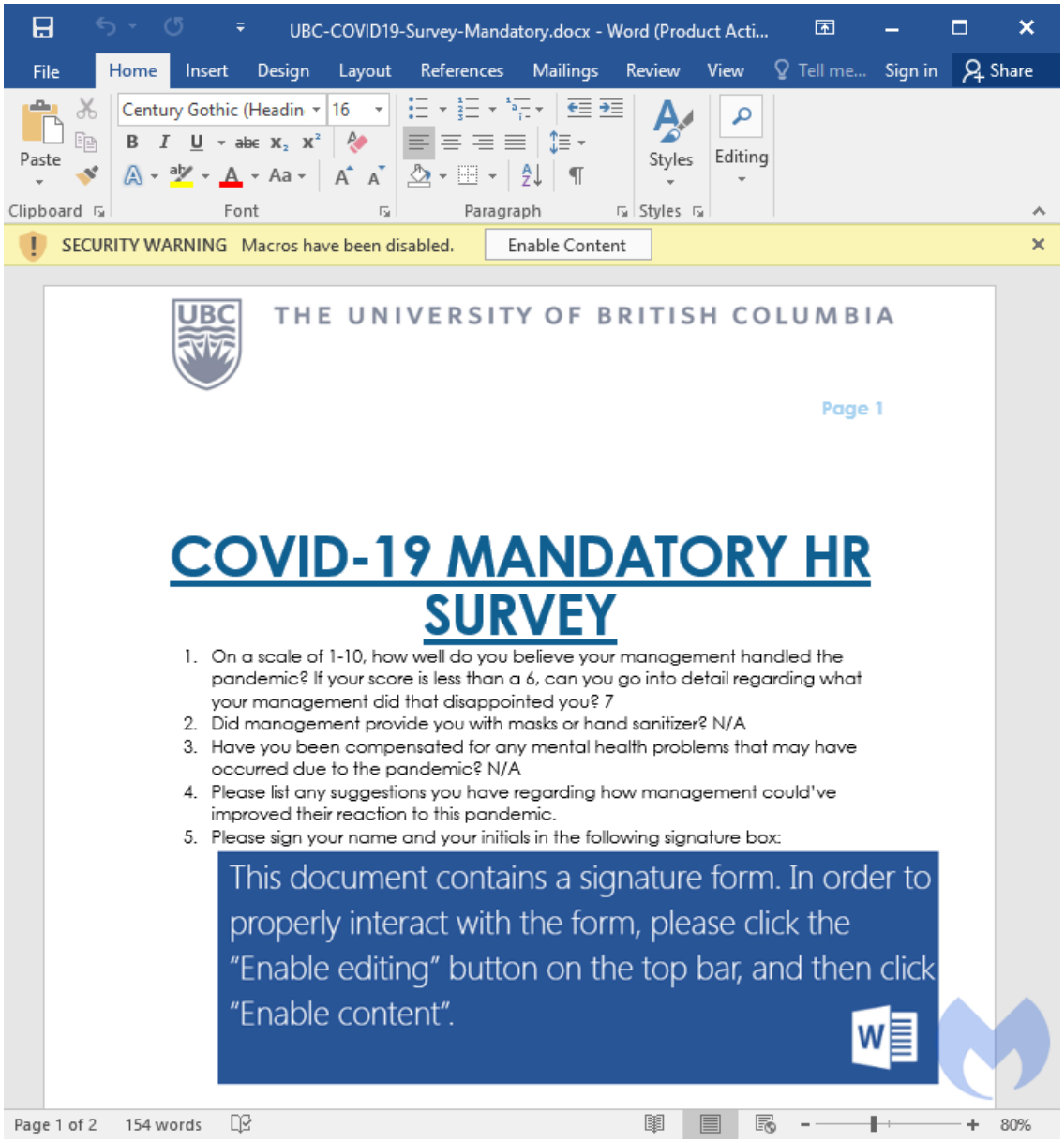
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

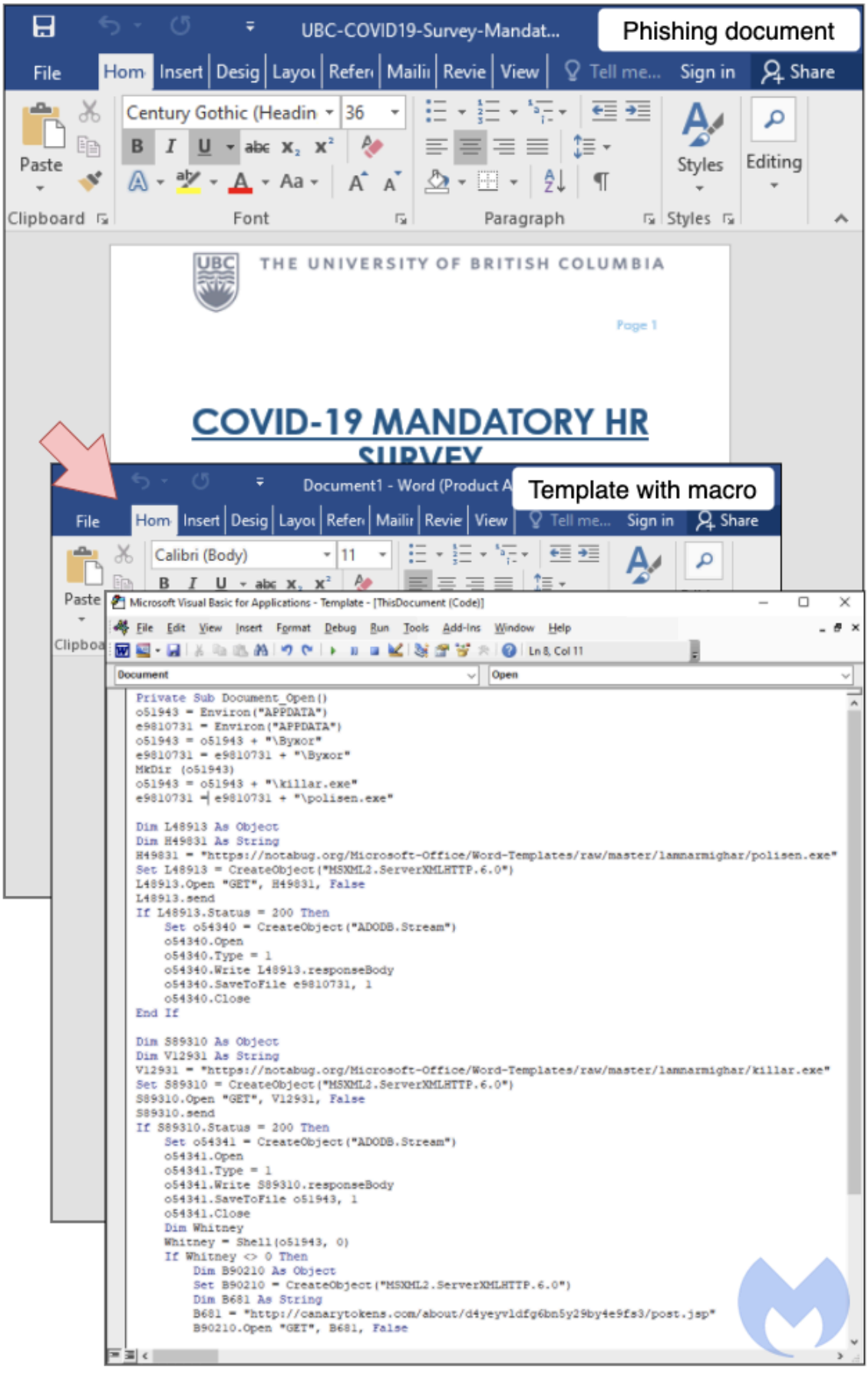
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

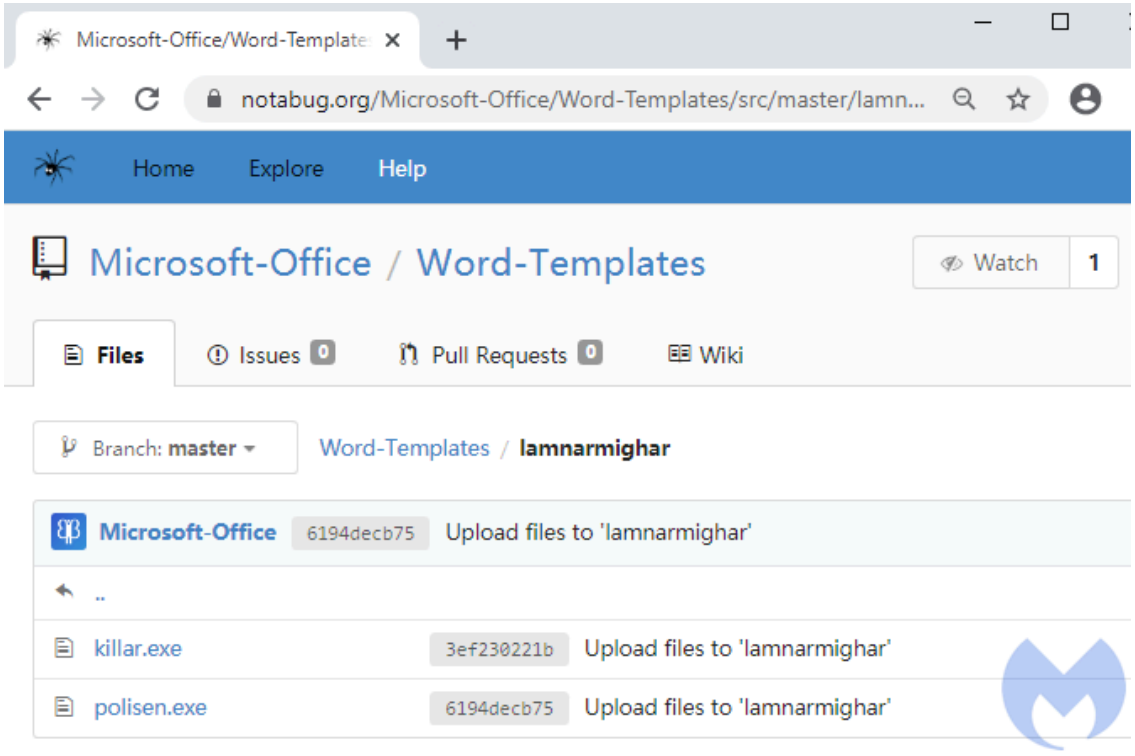
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



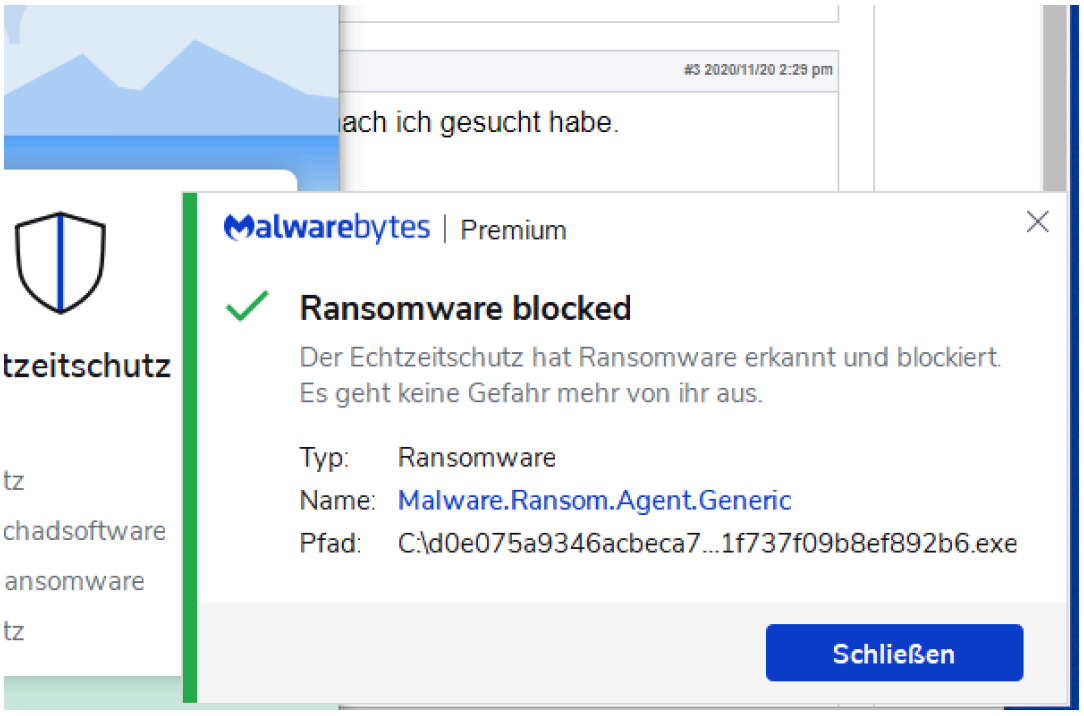
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

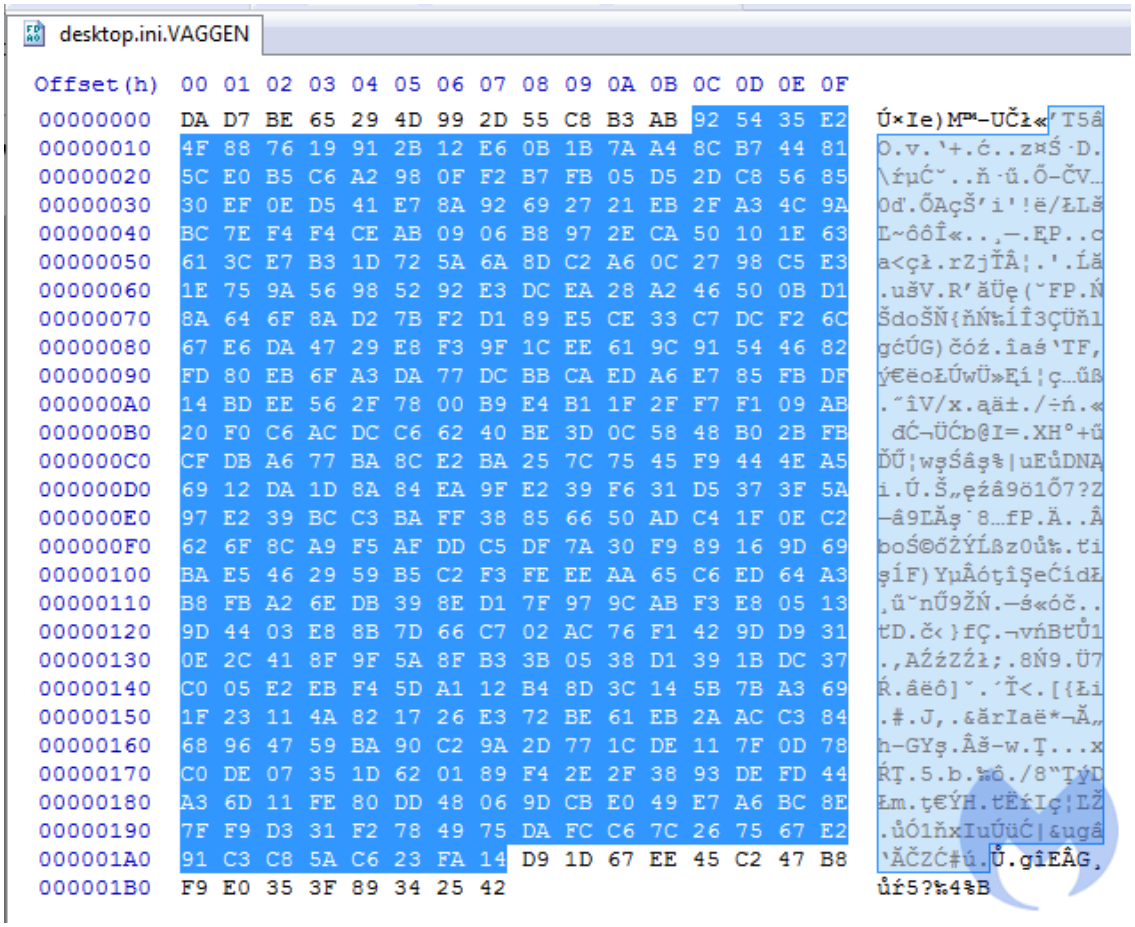
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

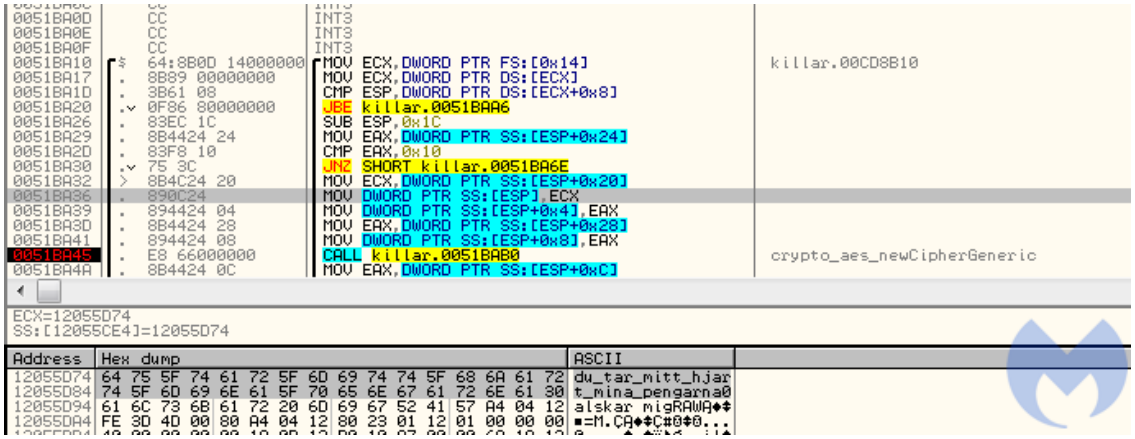


The content of the output file (with .VAGGEN extension) contains:

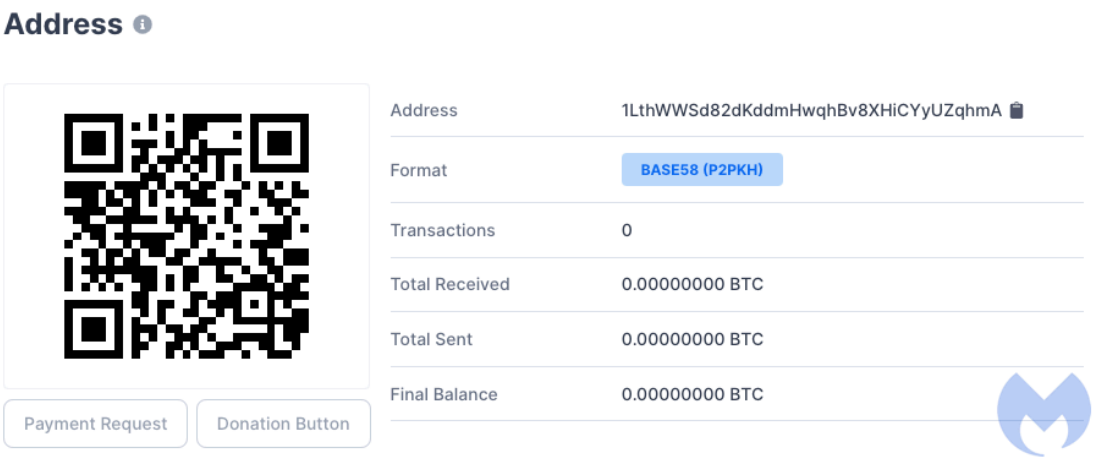
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

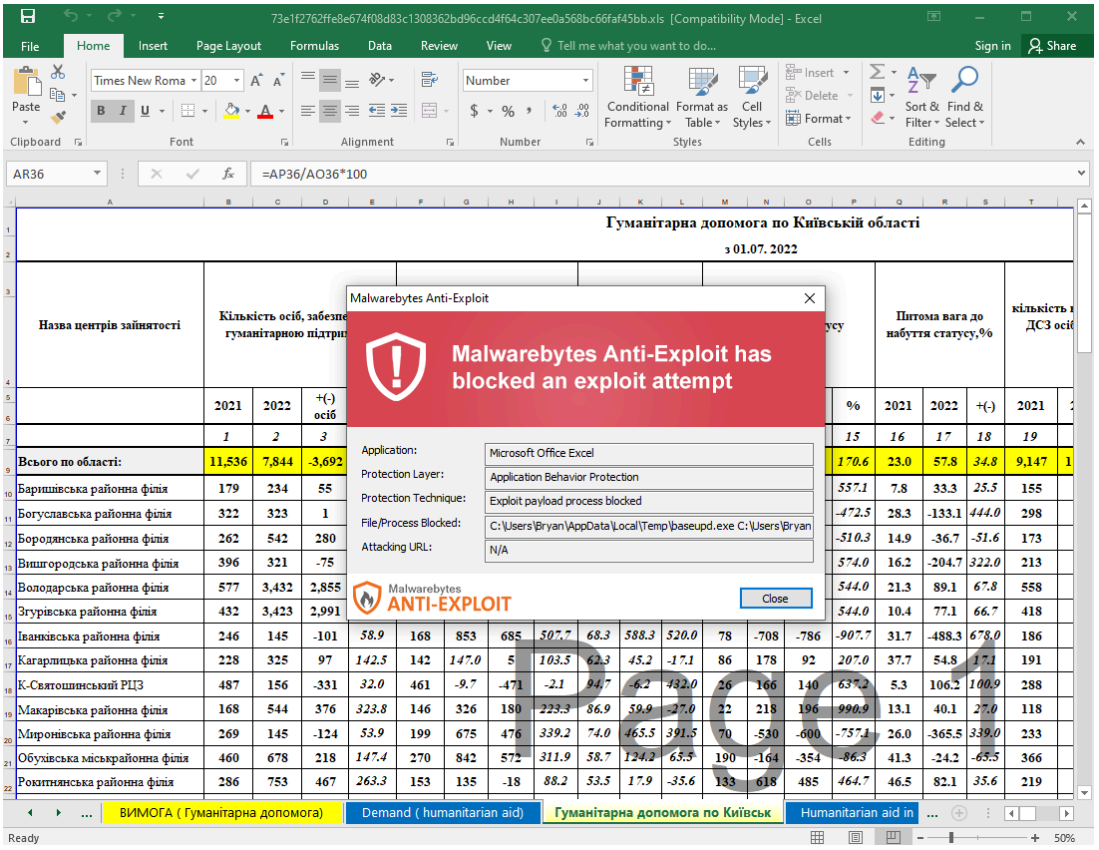
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

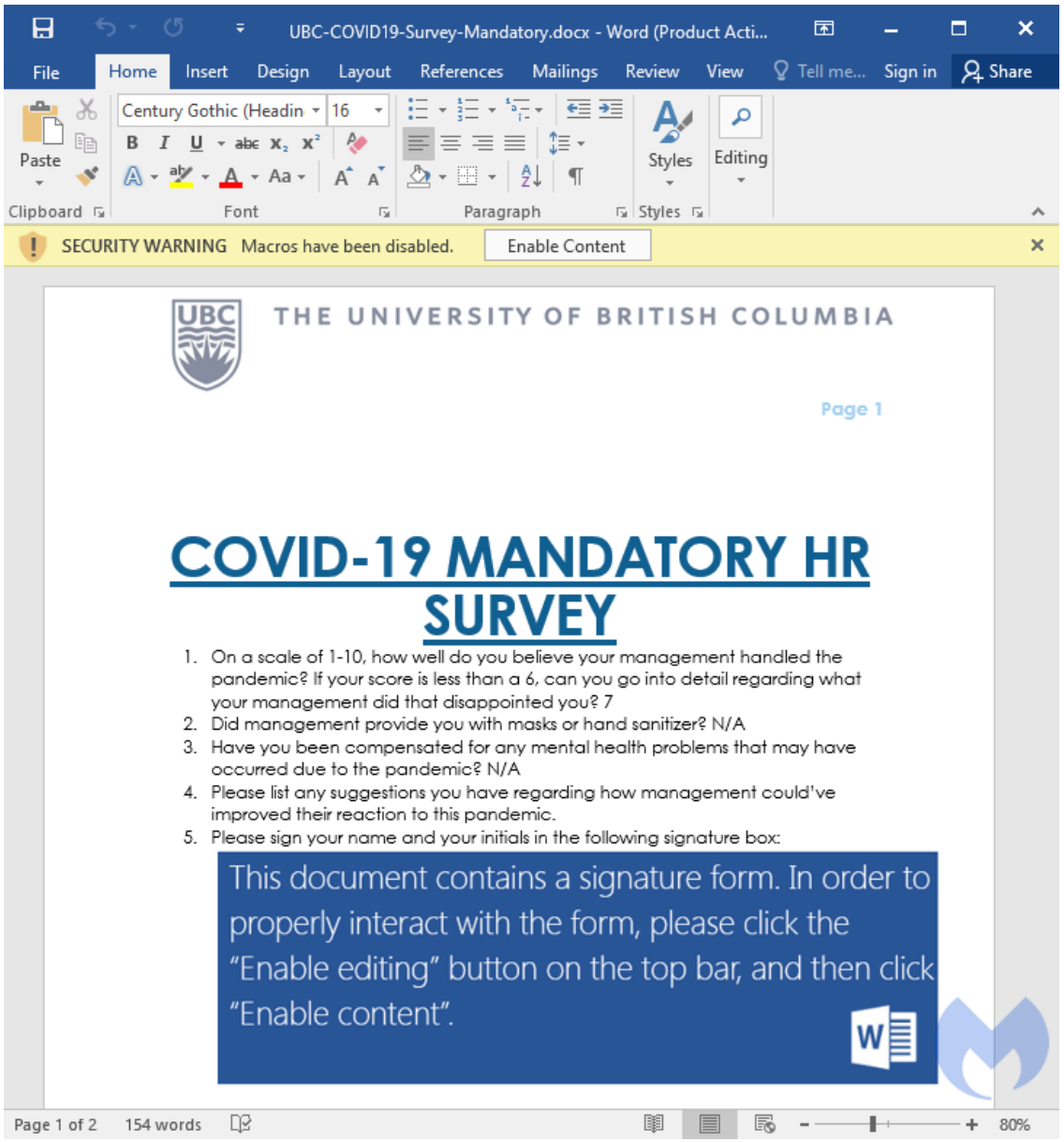
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

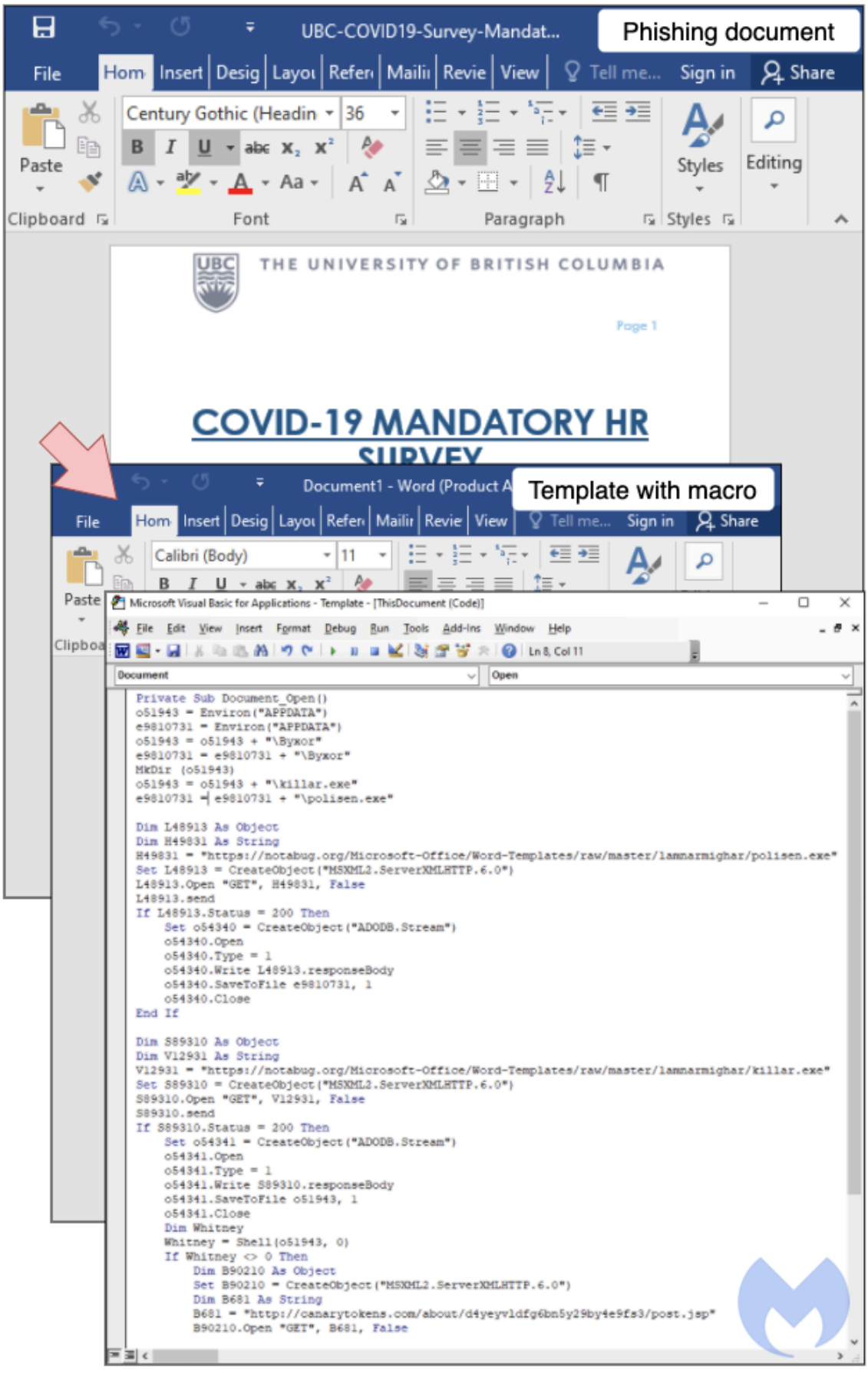
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

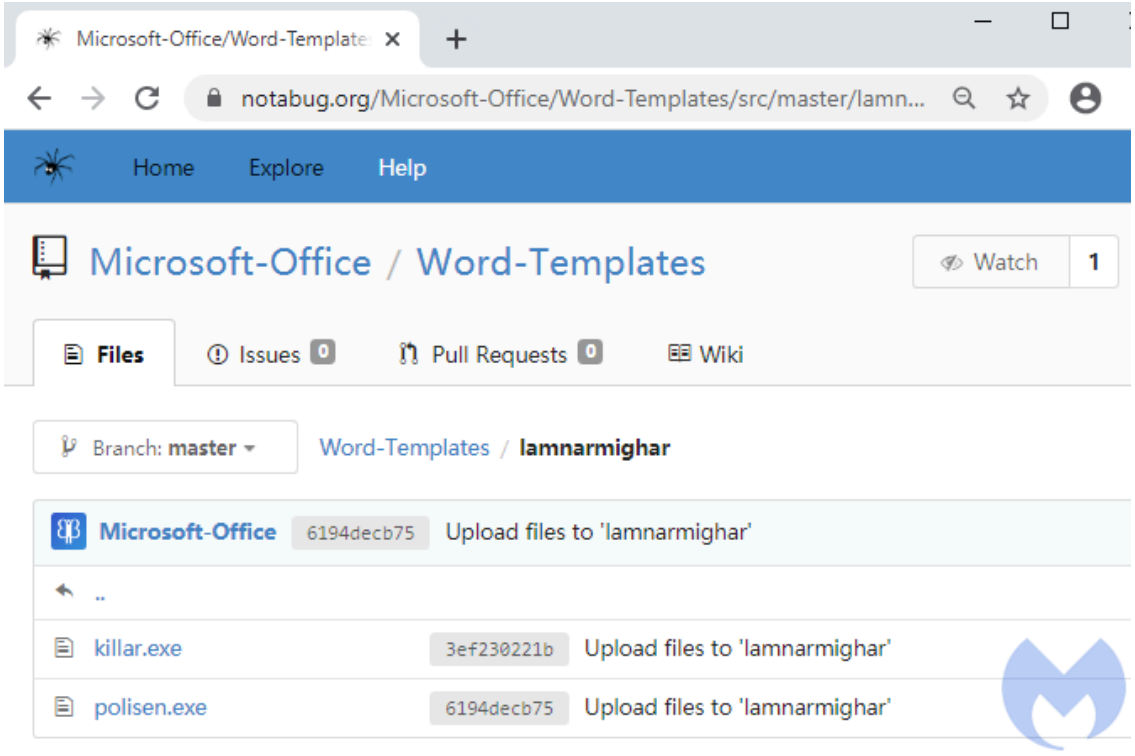
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



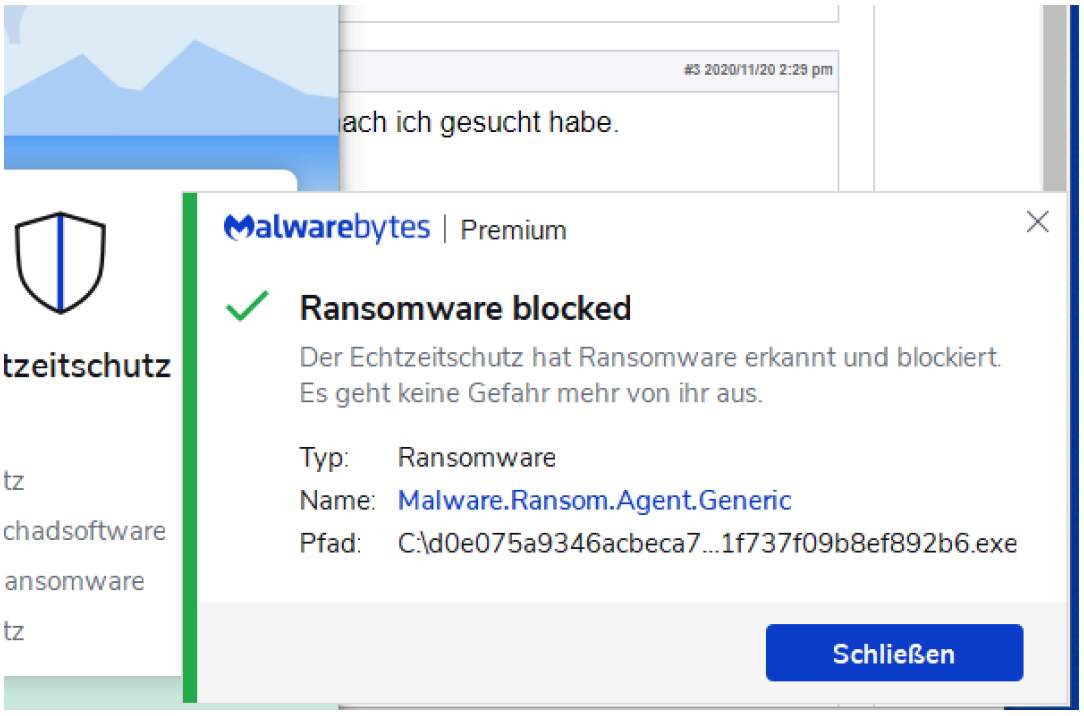
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

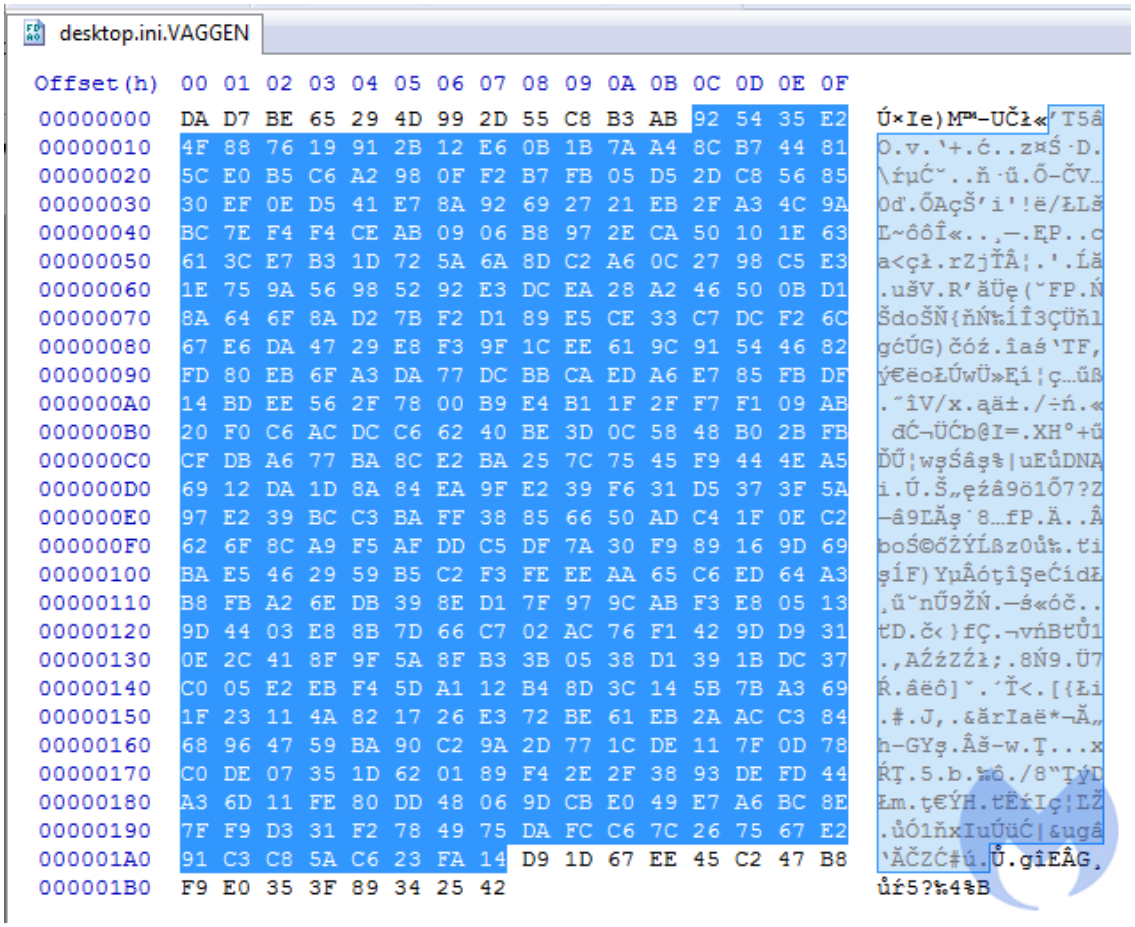
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



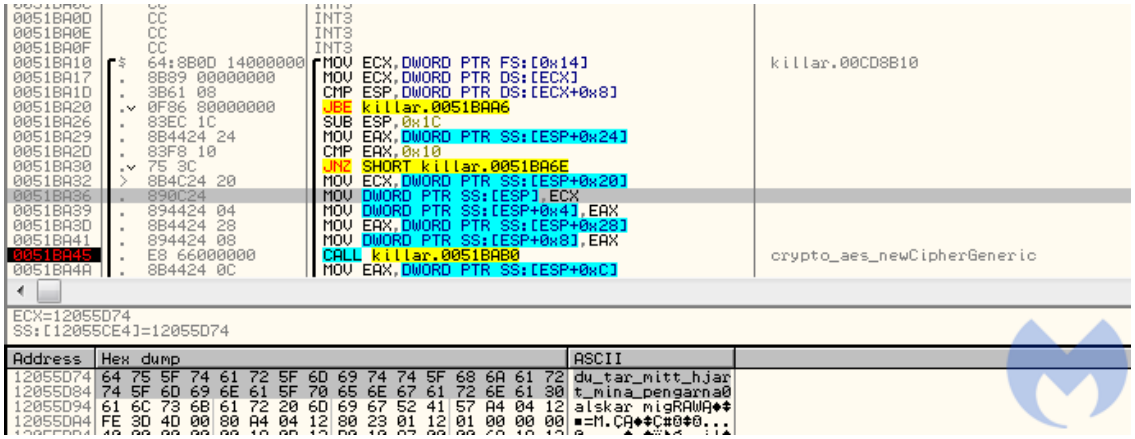
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

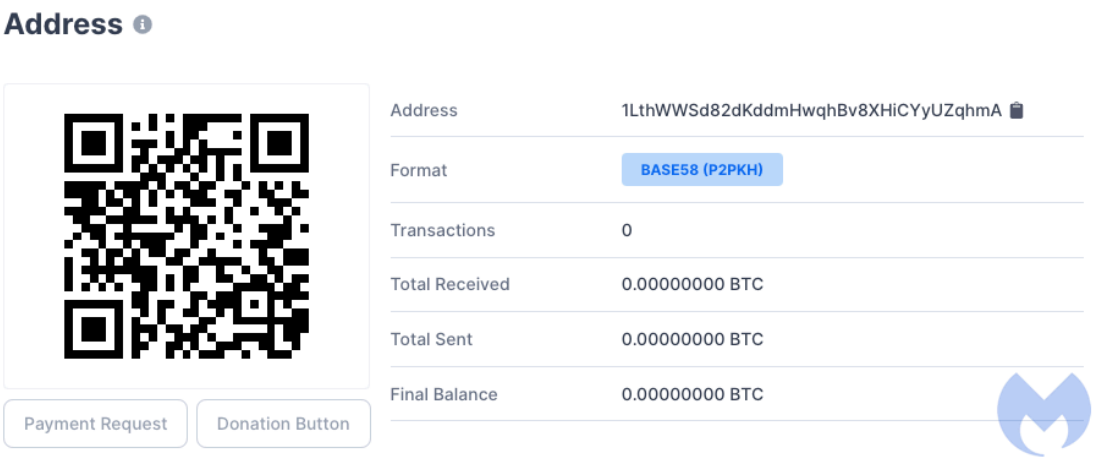
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

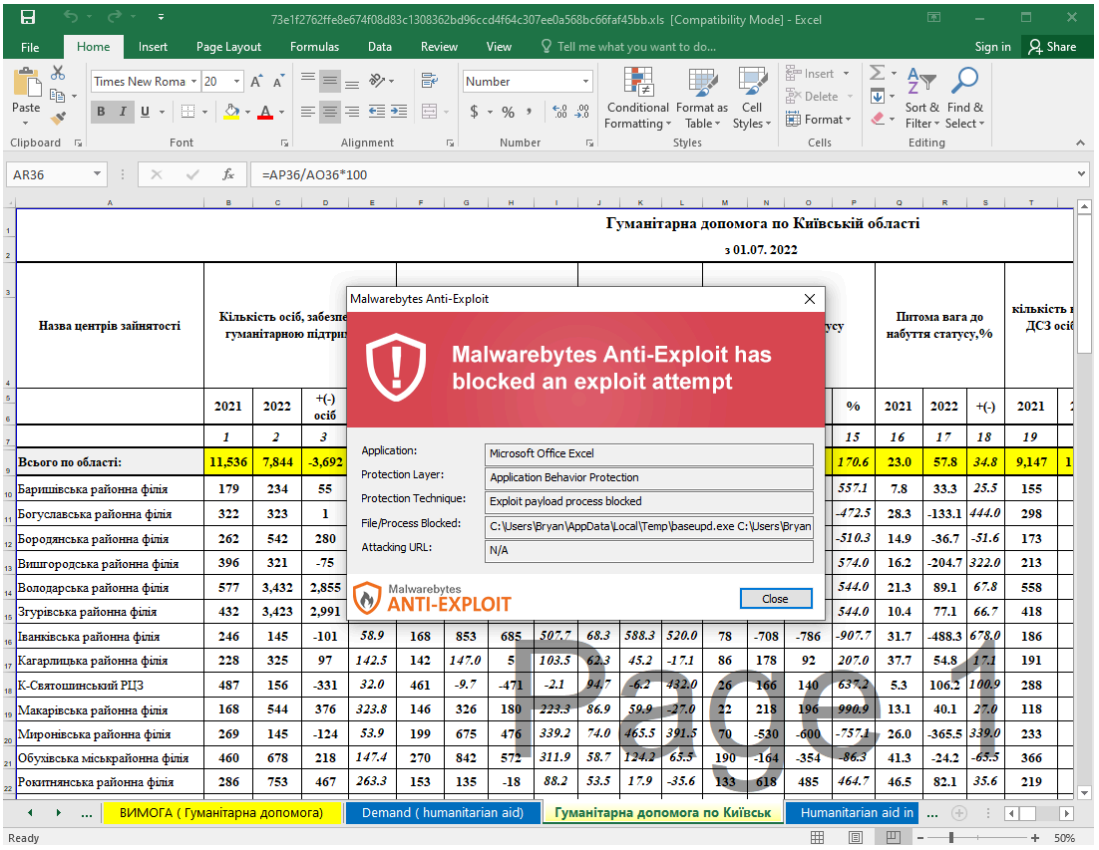
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

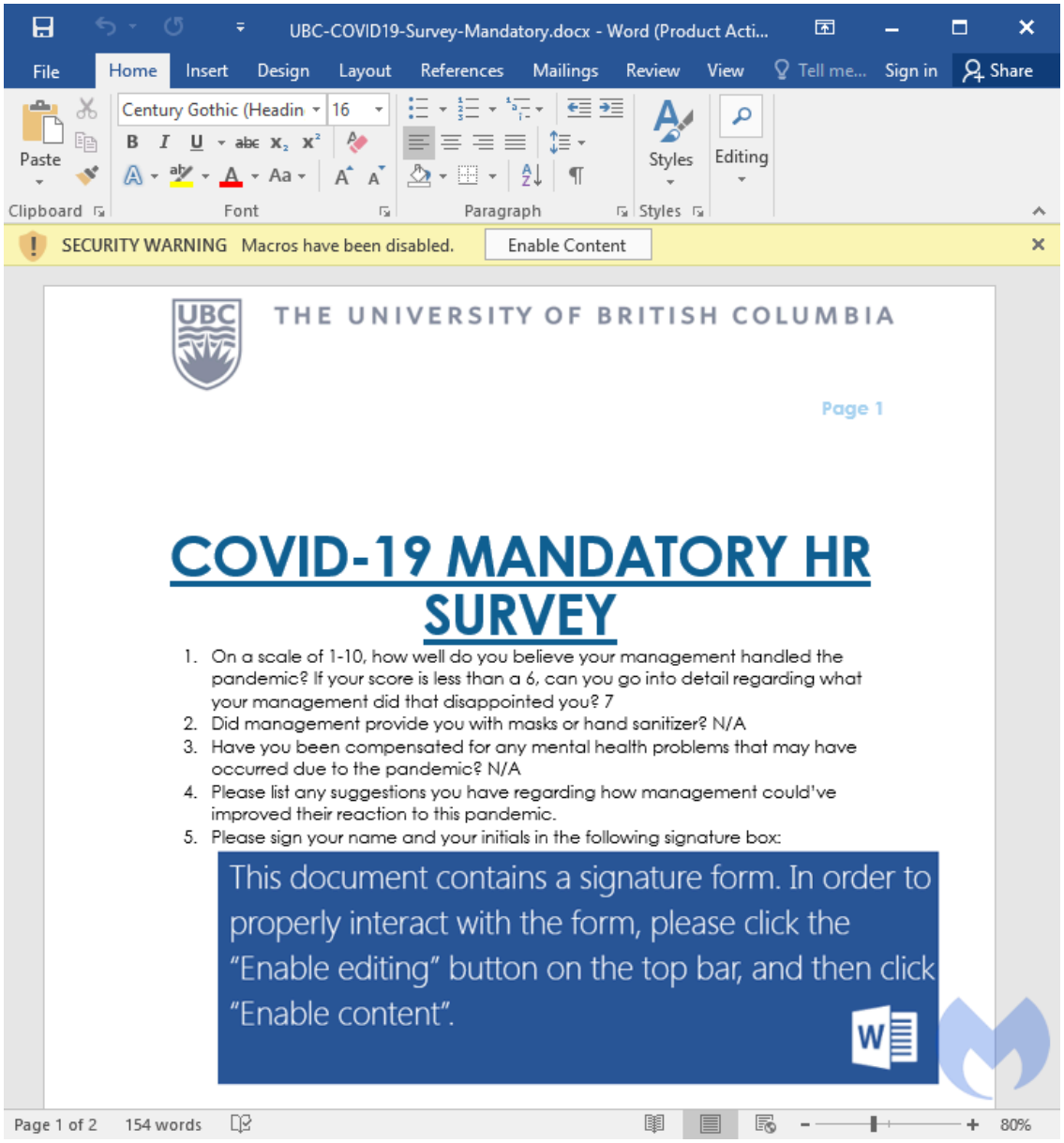
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

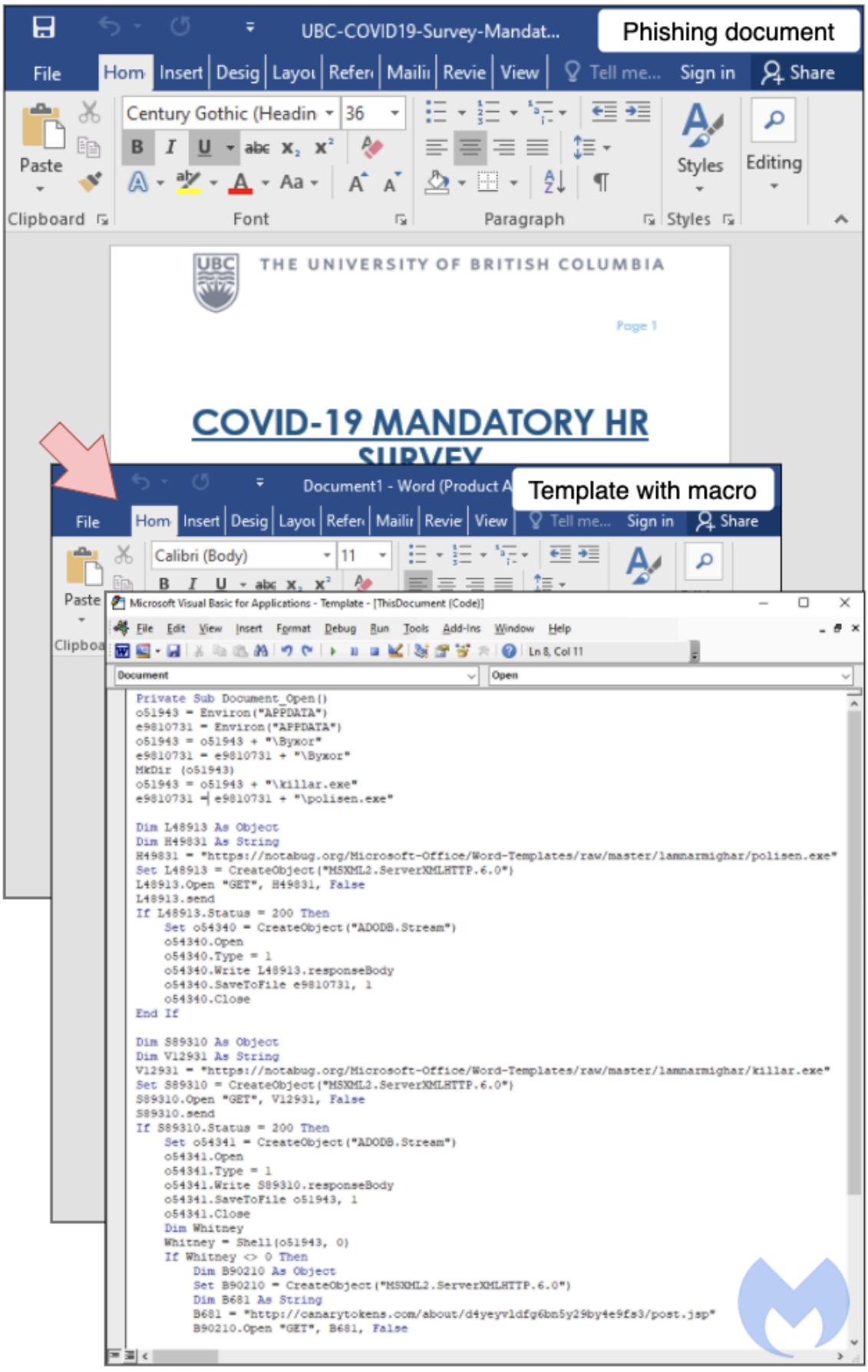
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

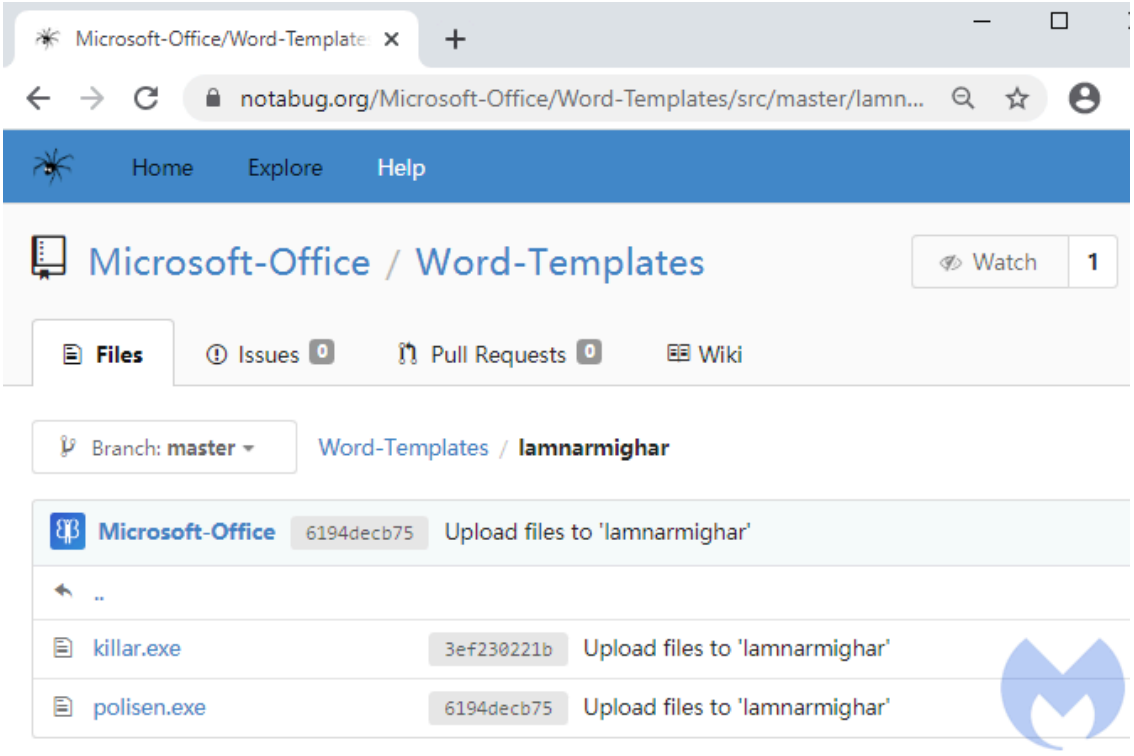
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



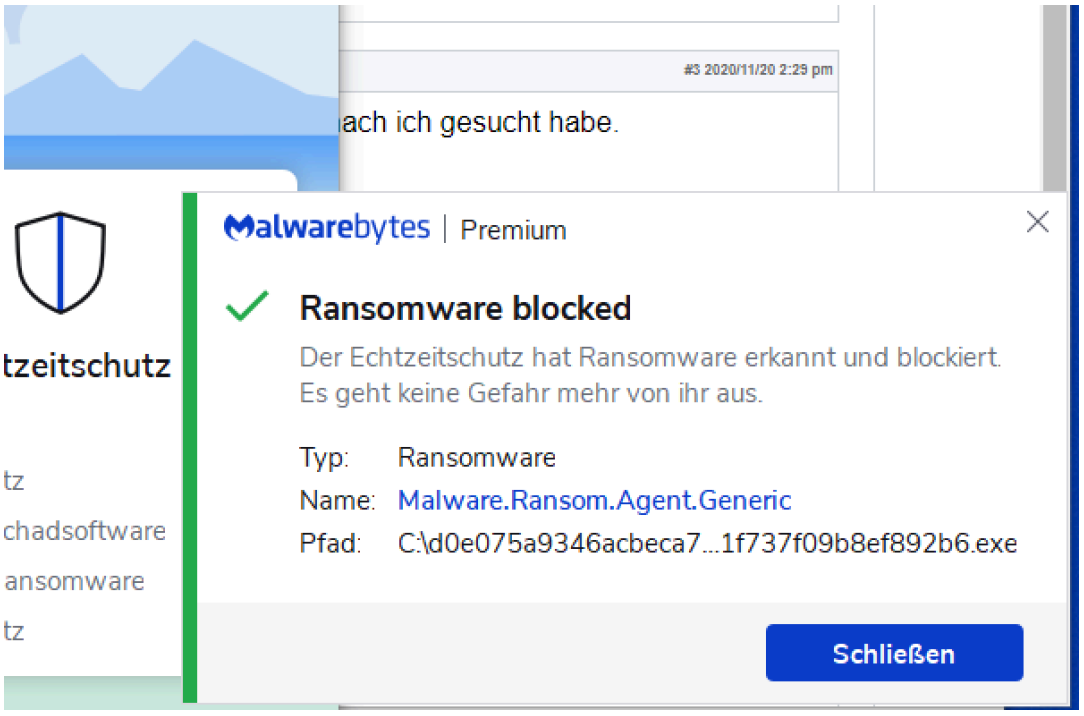
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

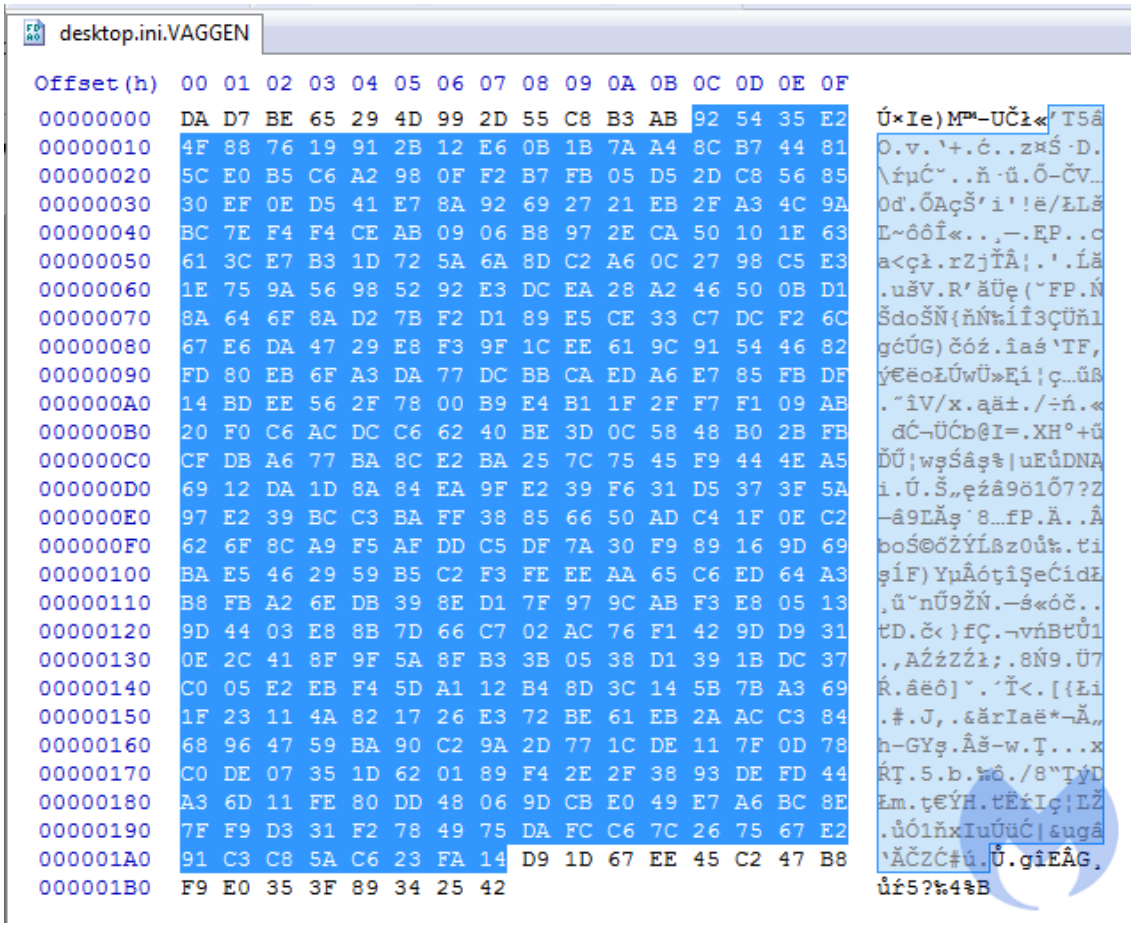
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

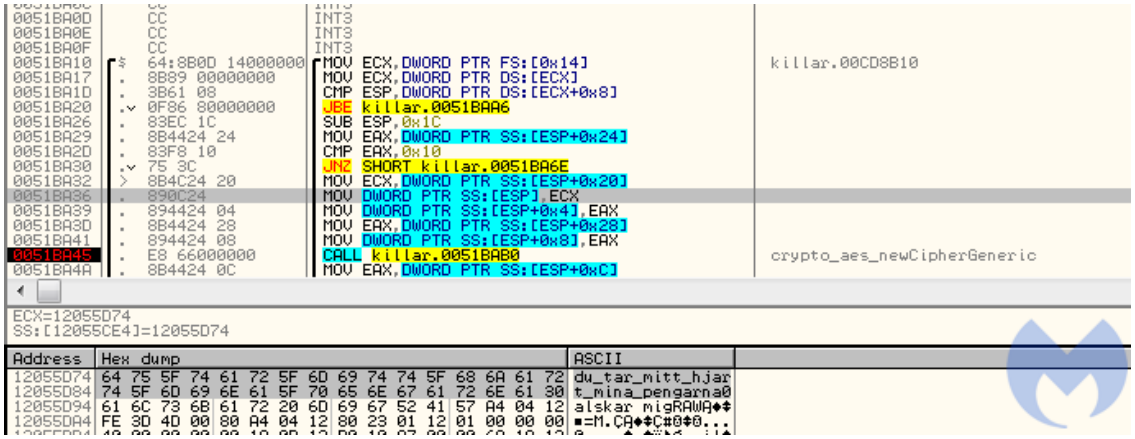


The content of the output file (with .VAGGEN extension) contains:

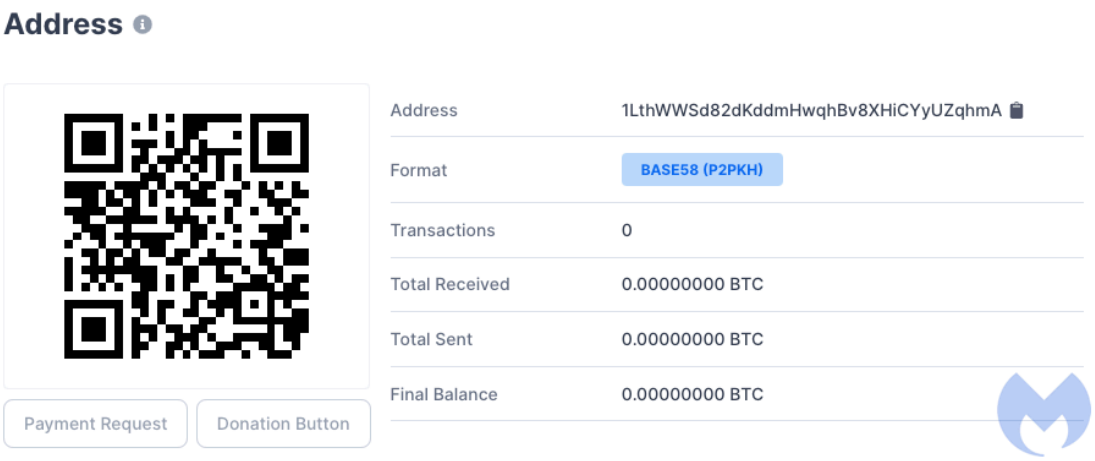
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

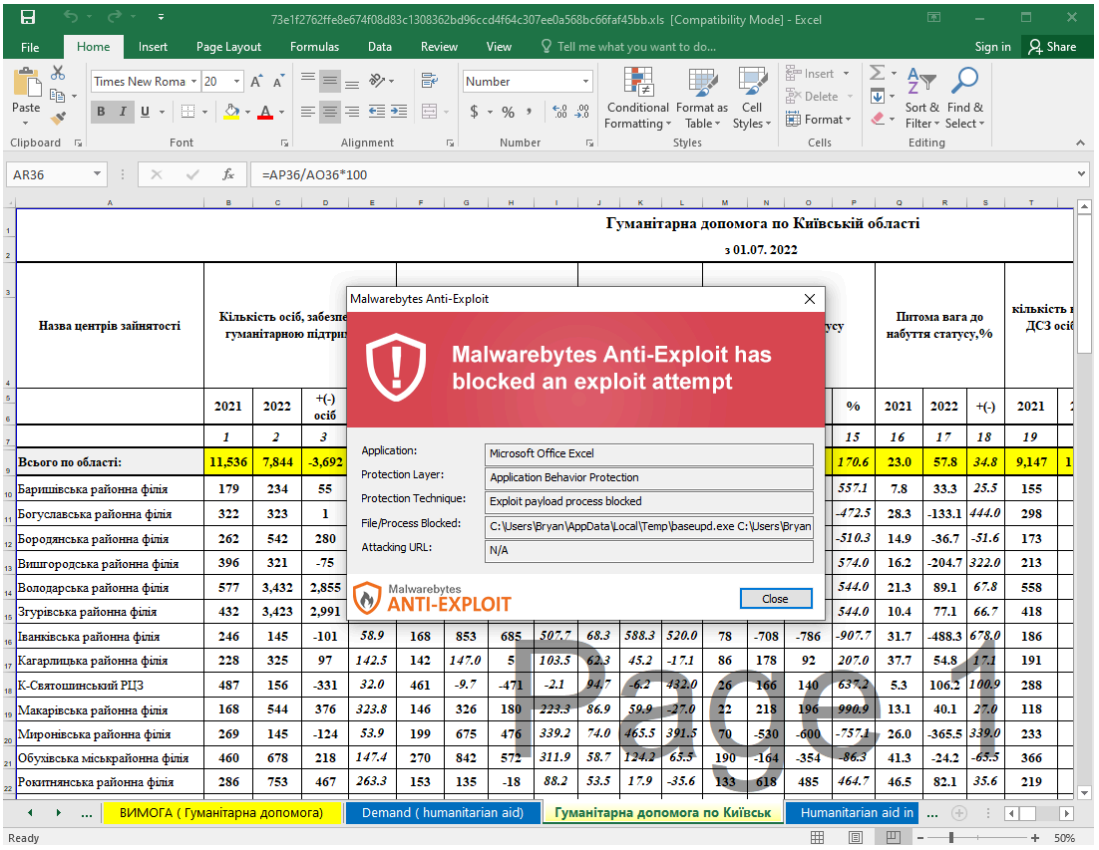
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
46  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
47    runtime_morestack_noctxt();
48  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
49  ((void (*)(void))loc_45CDA0)();
50  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
51  v38 = v4;
52  v30 = v5;
53  v31 = v6;
54  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
55  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
56  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
57  v32 = v10;
58  v11 = *(void (__cdecl **)(int))(v10 + 16);
59  v39 = v12;
60  v11(v12);
61  v29 = v13;
62  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
63  v40 = v15;
64  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
65  v17 = v27;
66  if ( v27 )
67  {
68    if ( v27 )
69      v17 = *(_DWORD *)(v27 + 4);
70    v43 = v17;
71    v44 = v28;
72    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
73  }
```

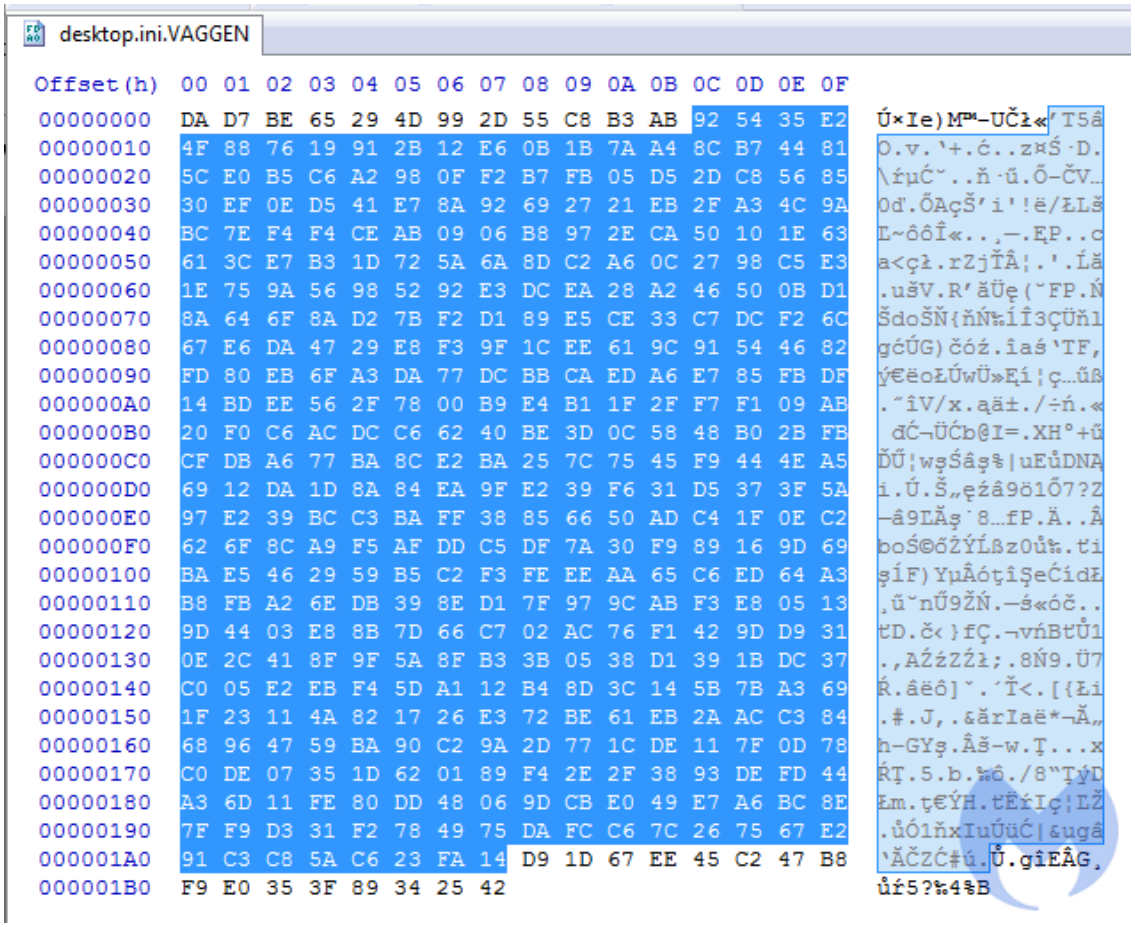


The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

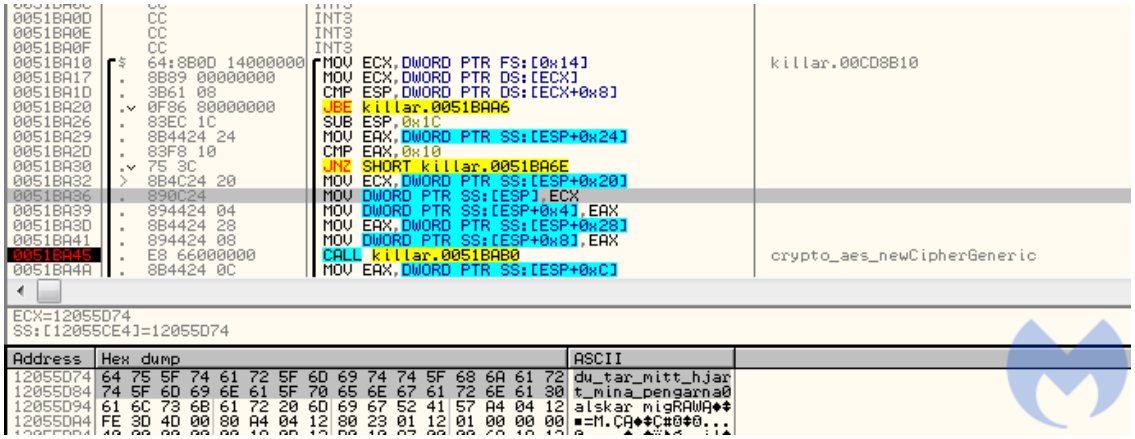


The content of the output file (with .VAGGEN extension) contains:

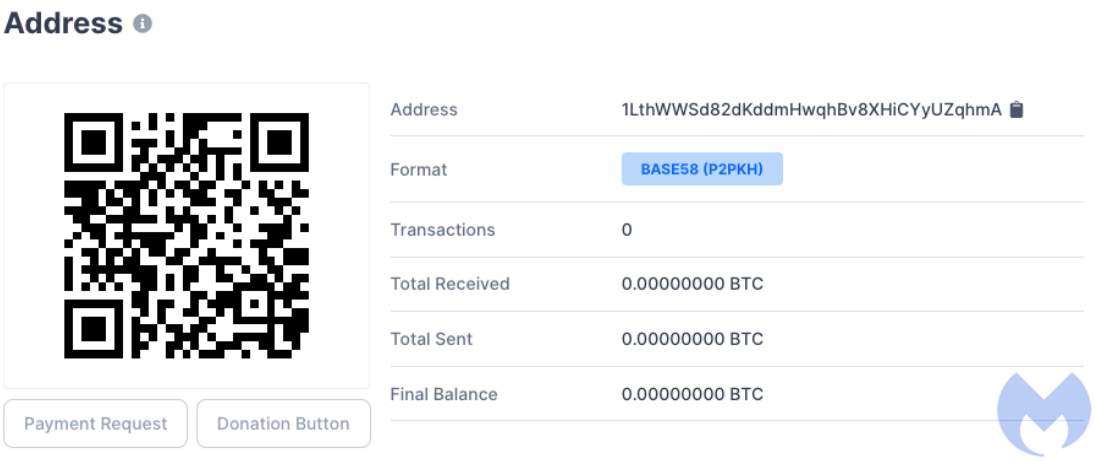
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

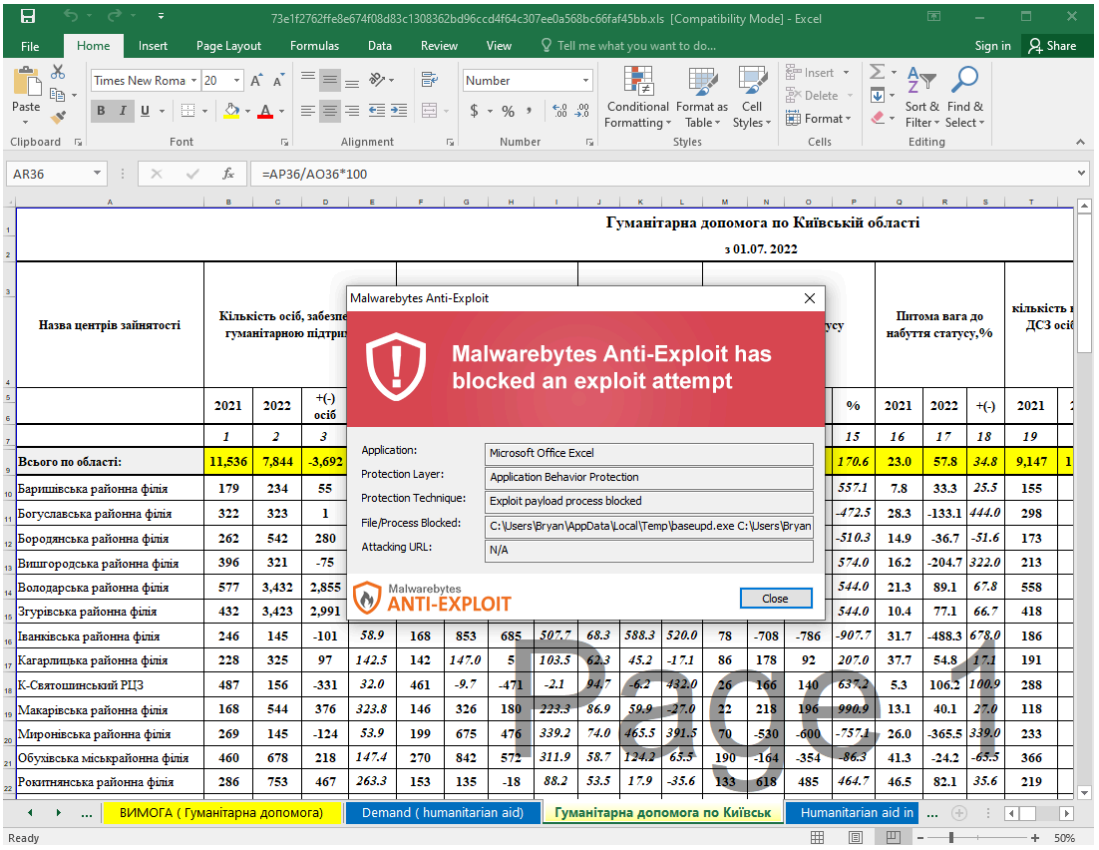
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

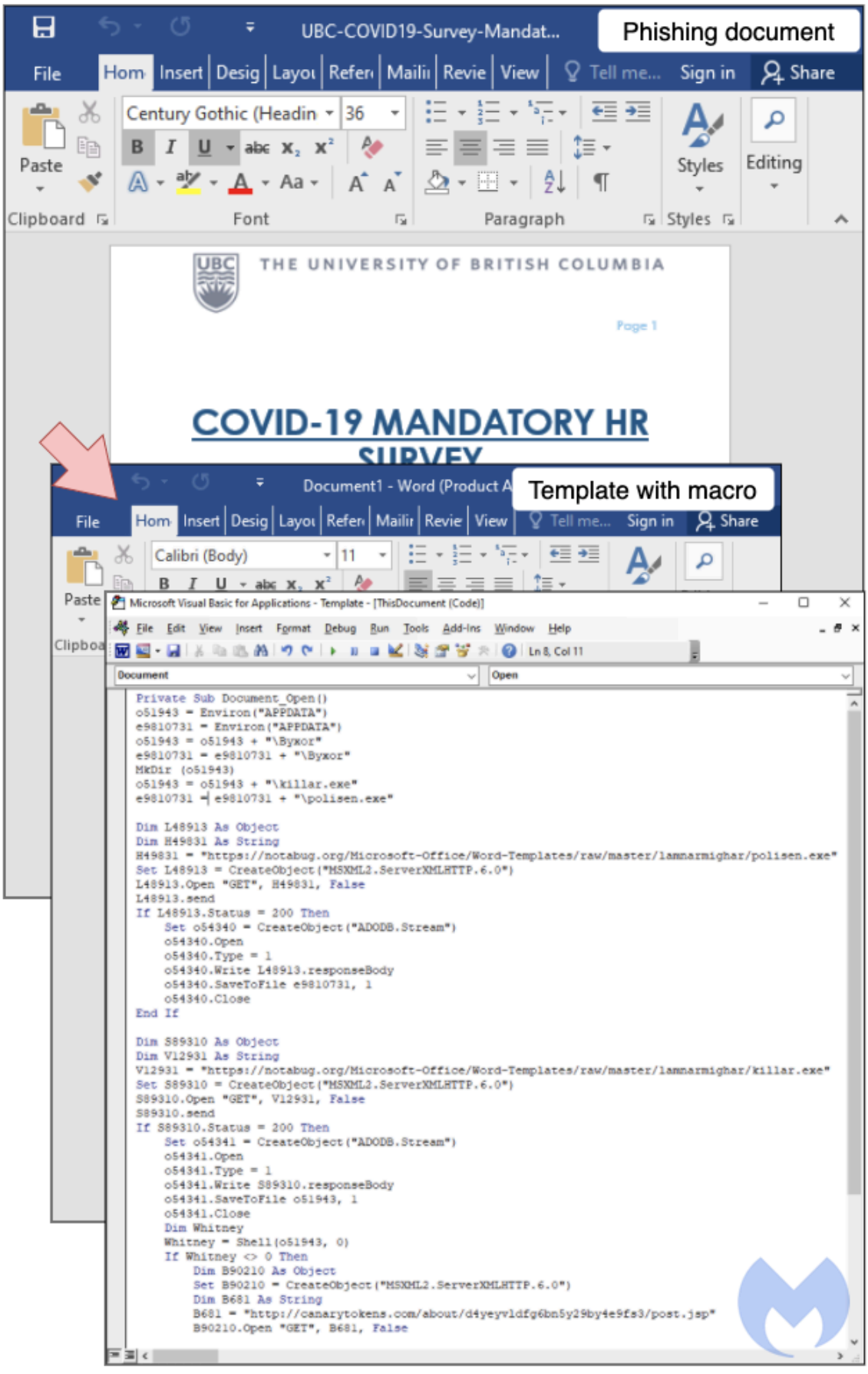
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

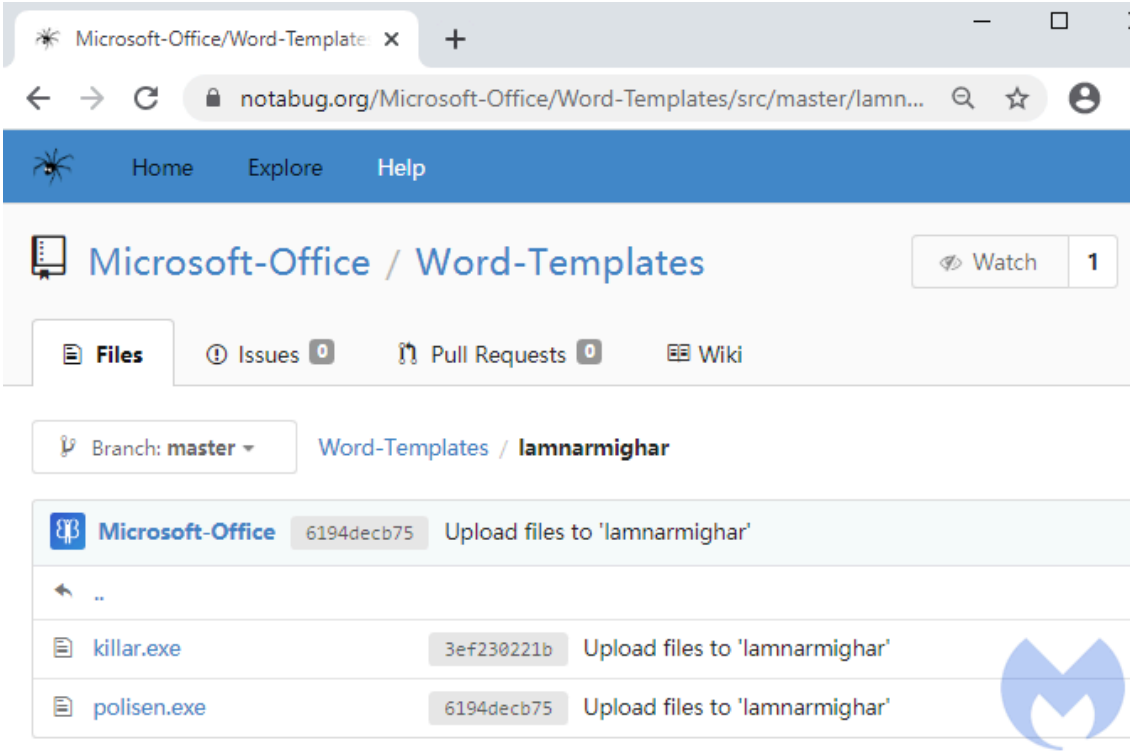
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



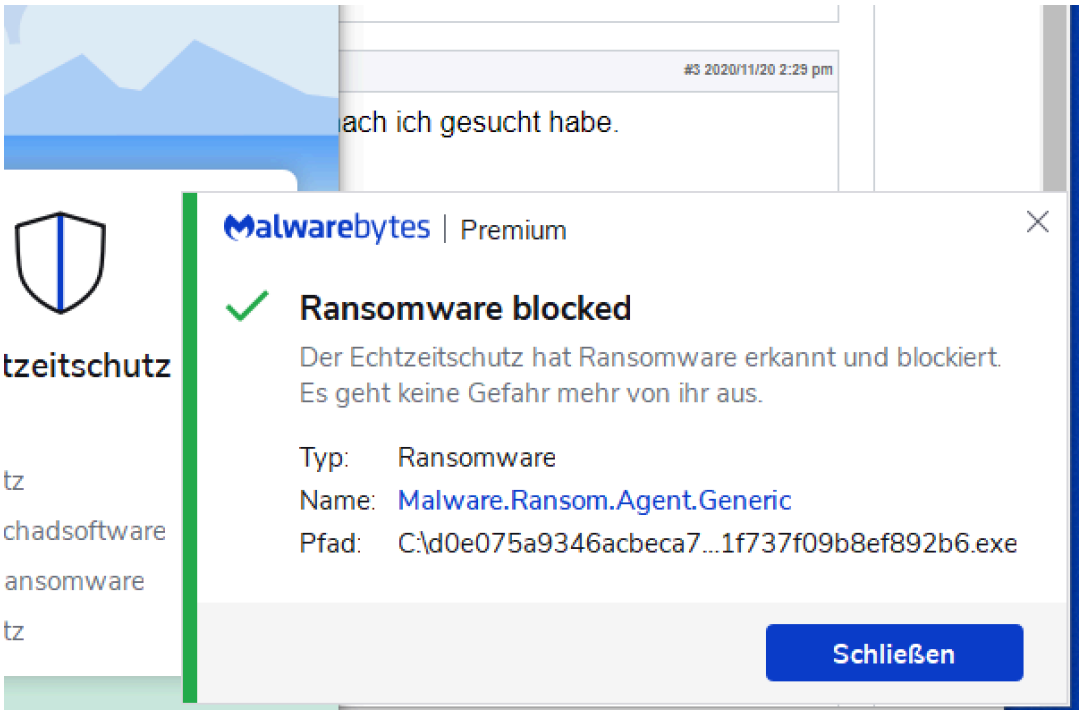
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

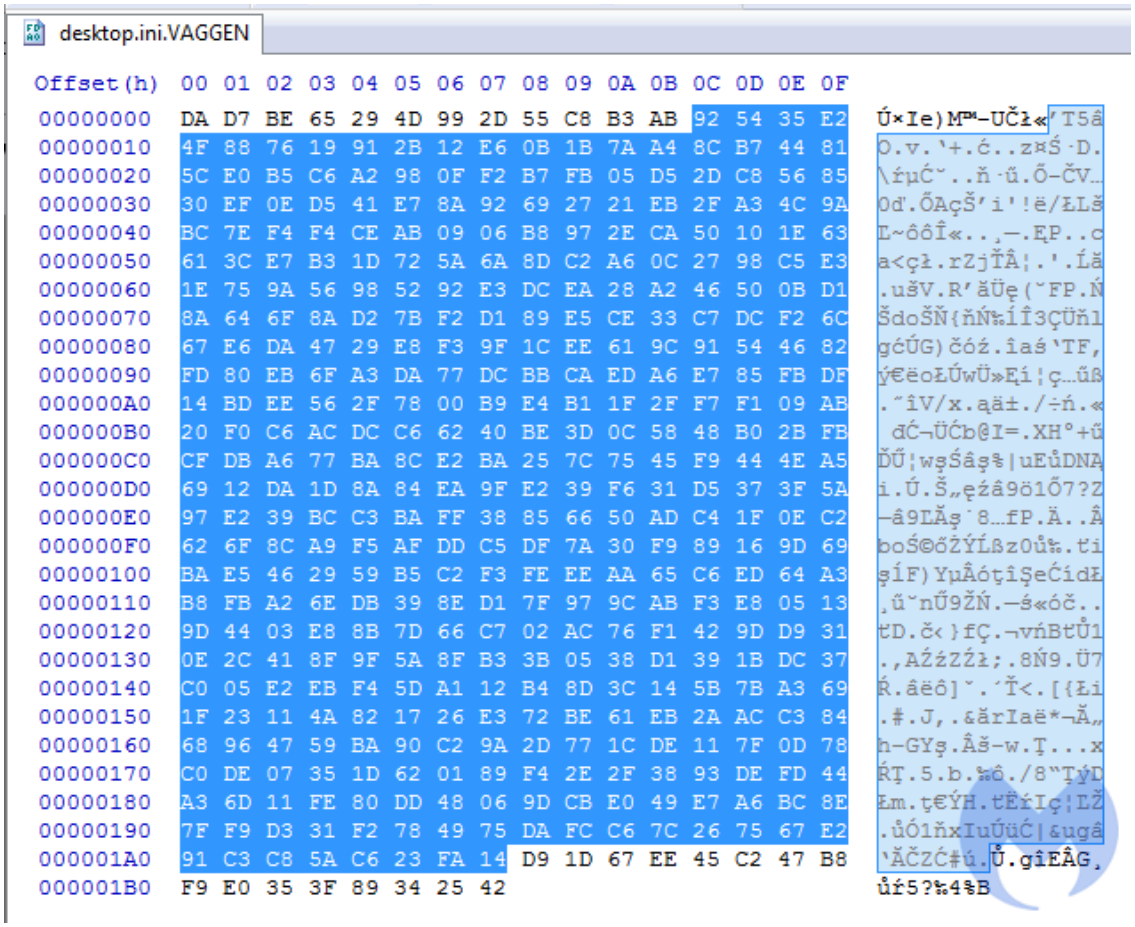
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

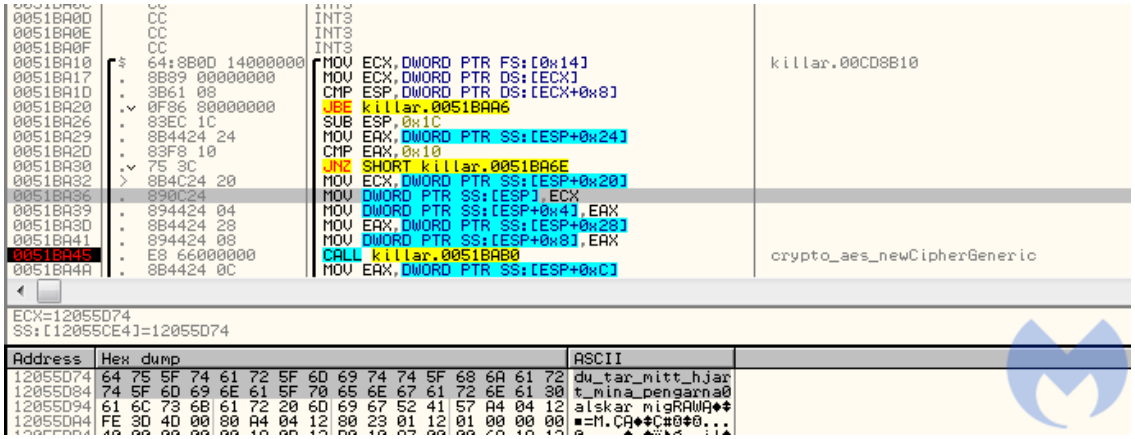


The content of the output file (with .VAGGEN extension) contains:

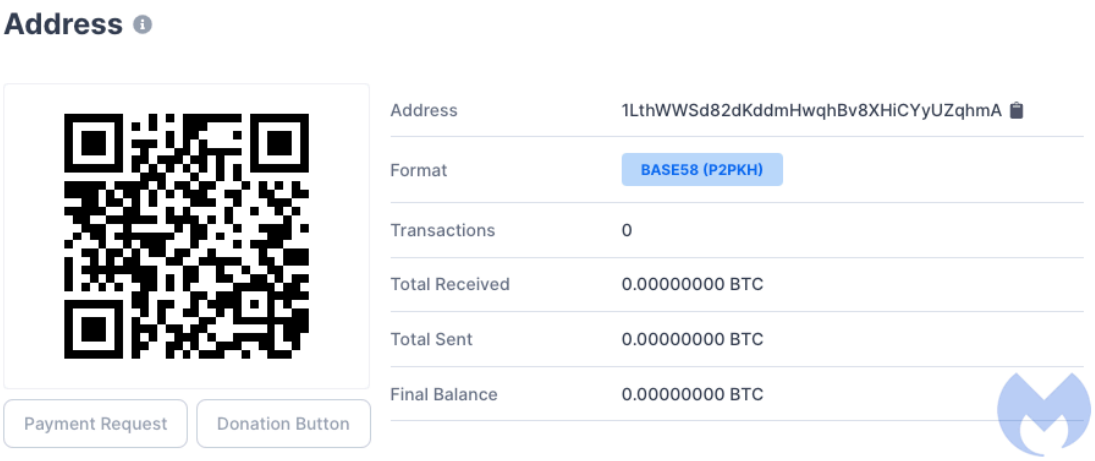
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

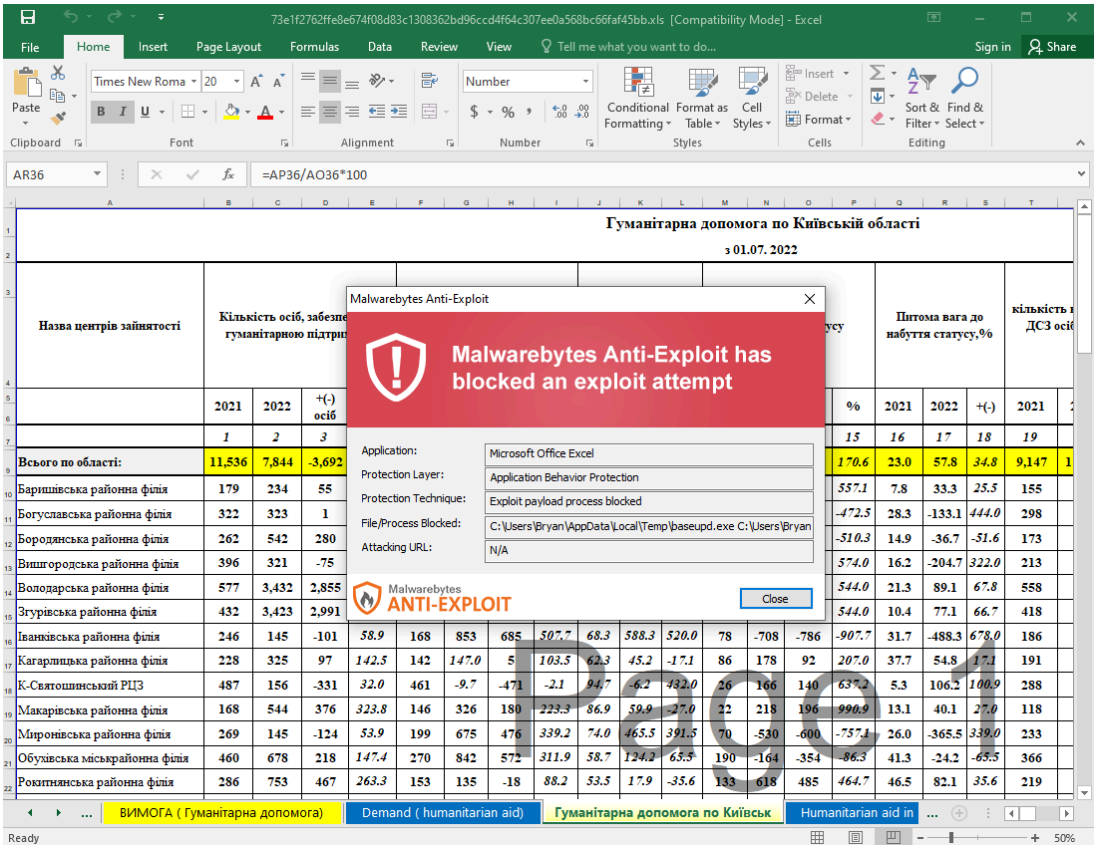
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

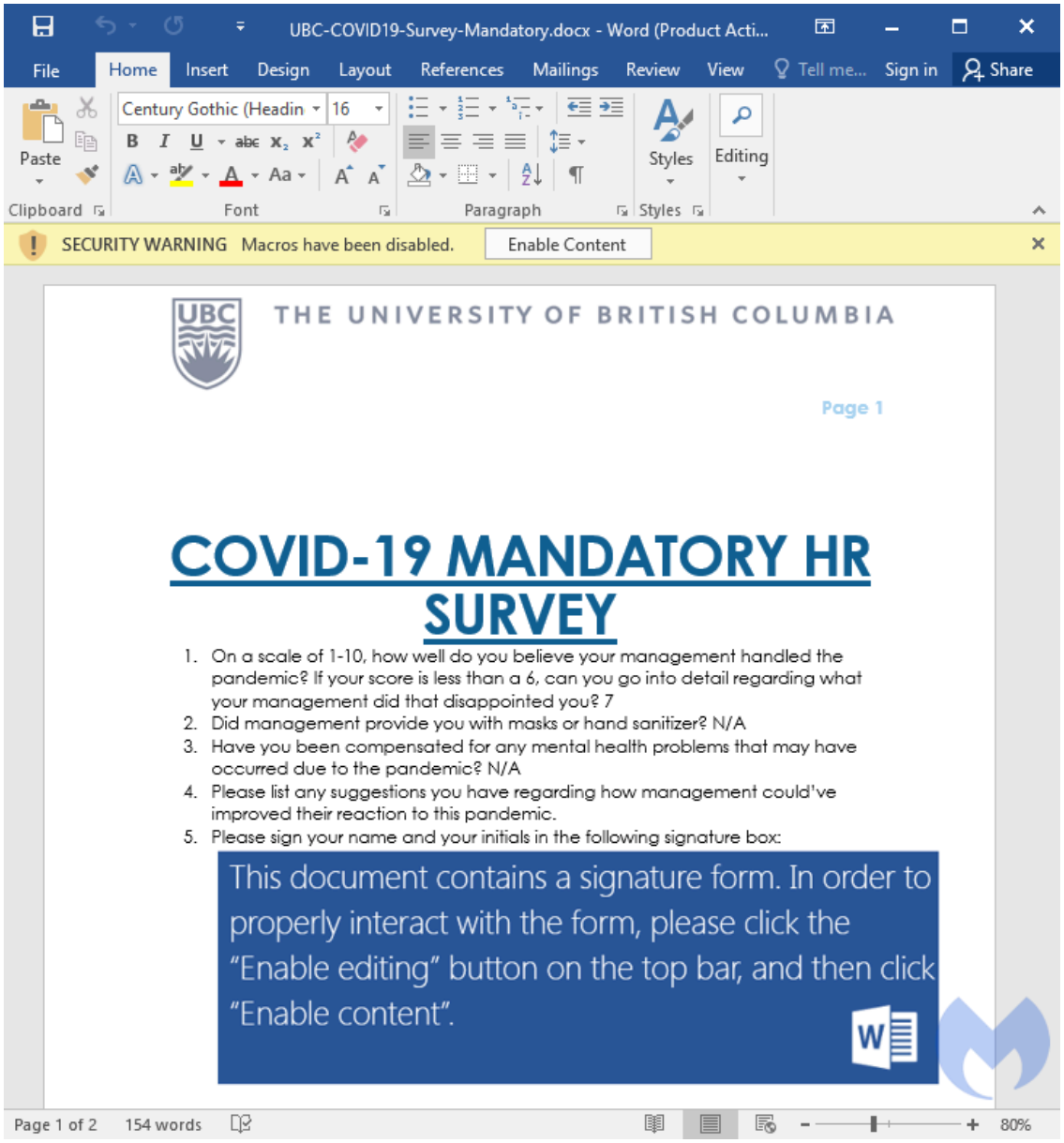
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

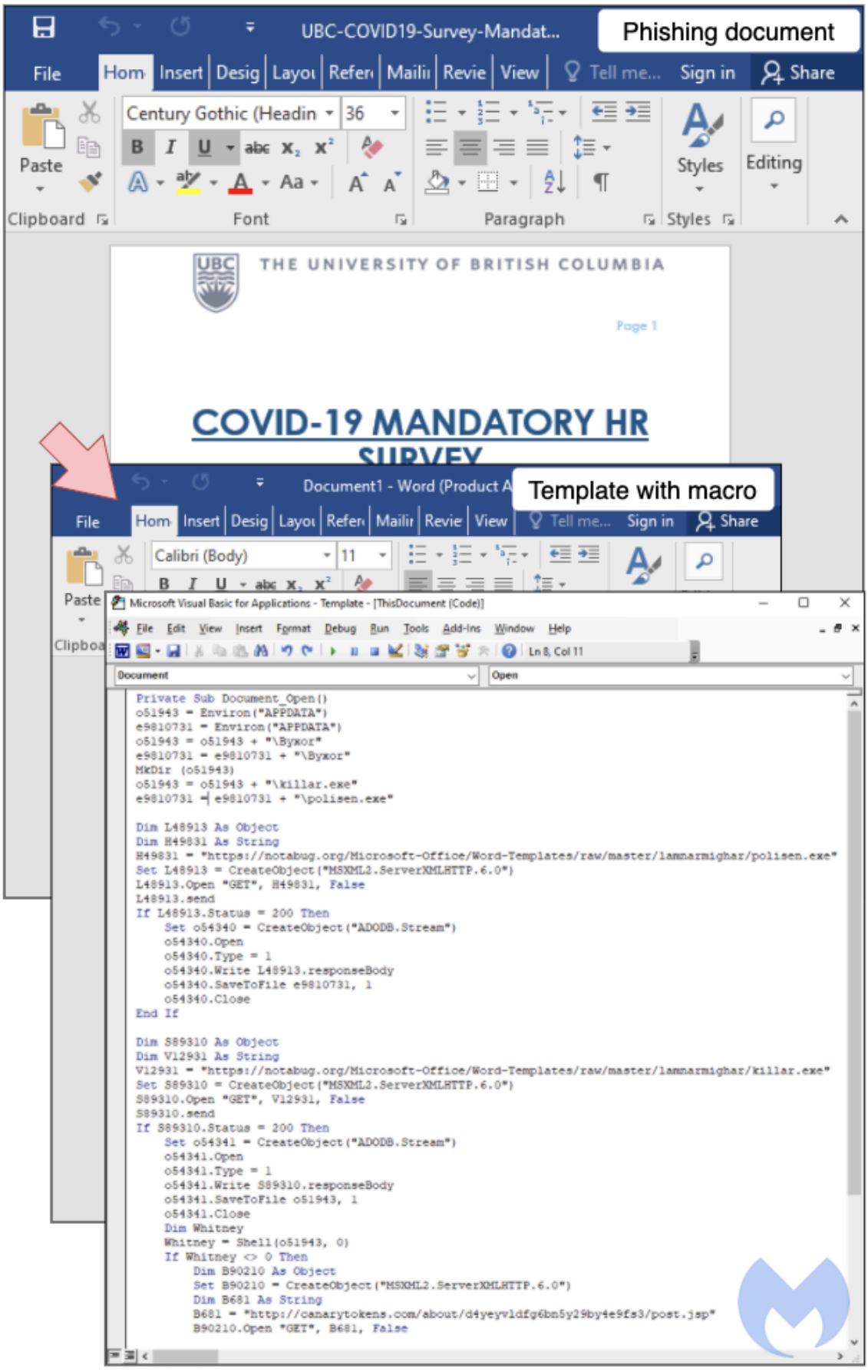
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

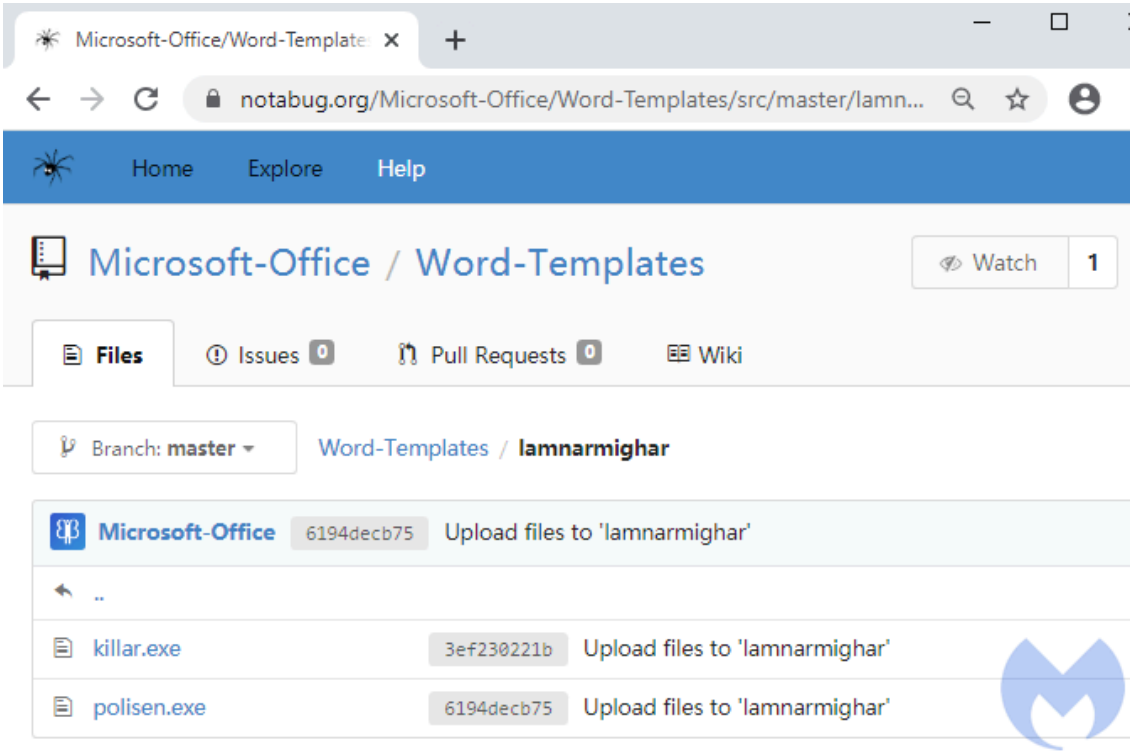
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



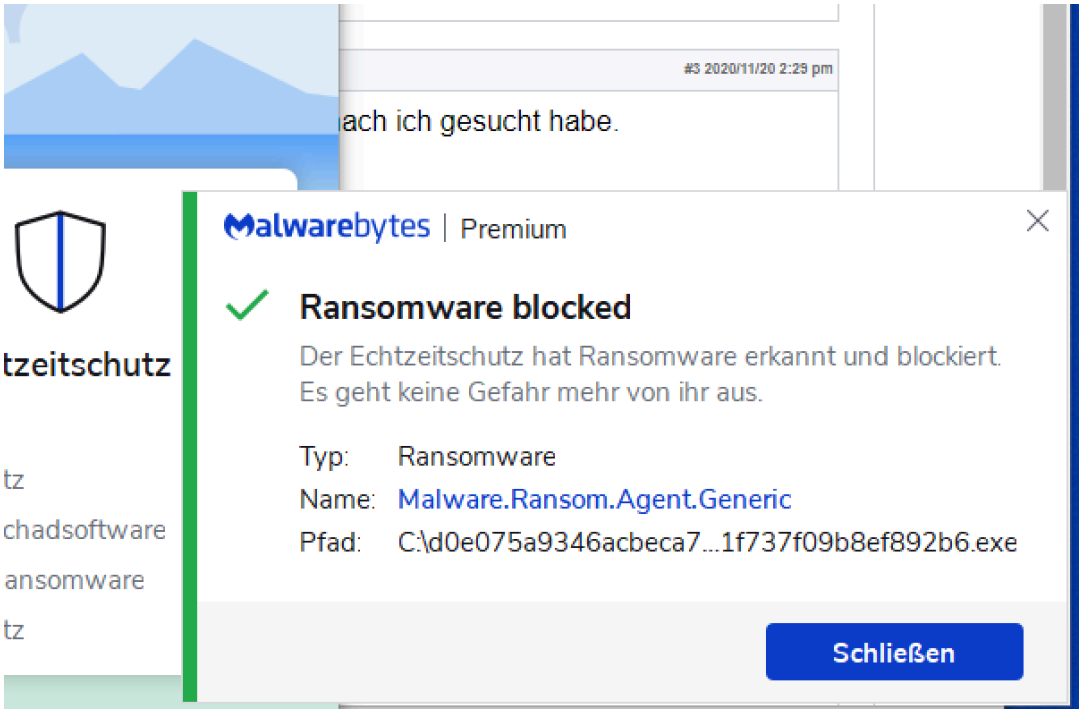
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

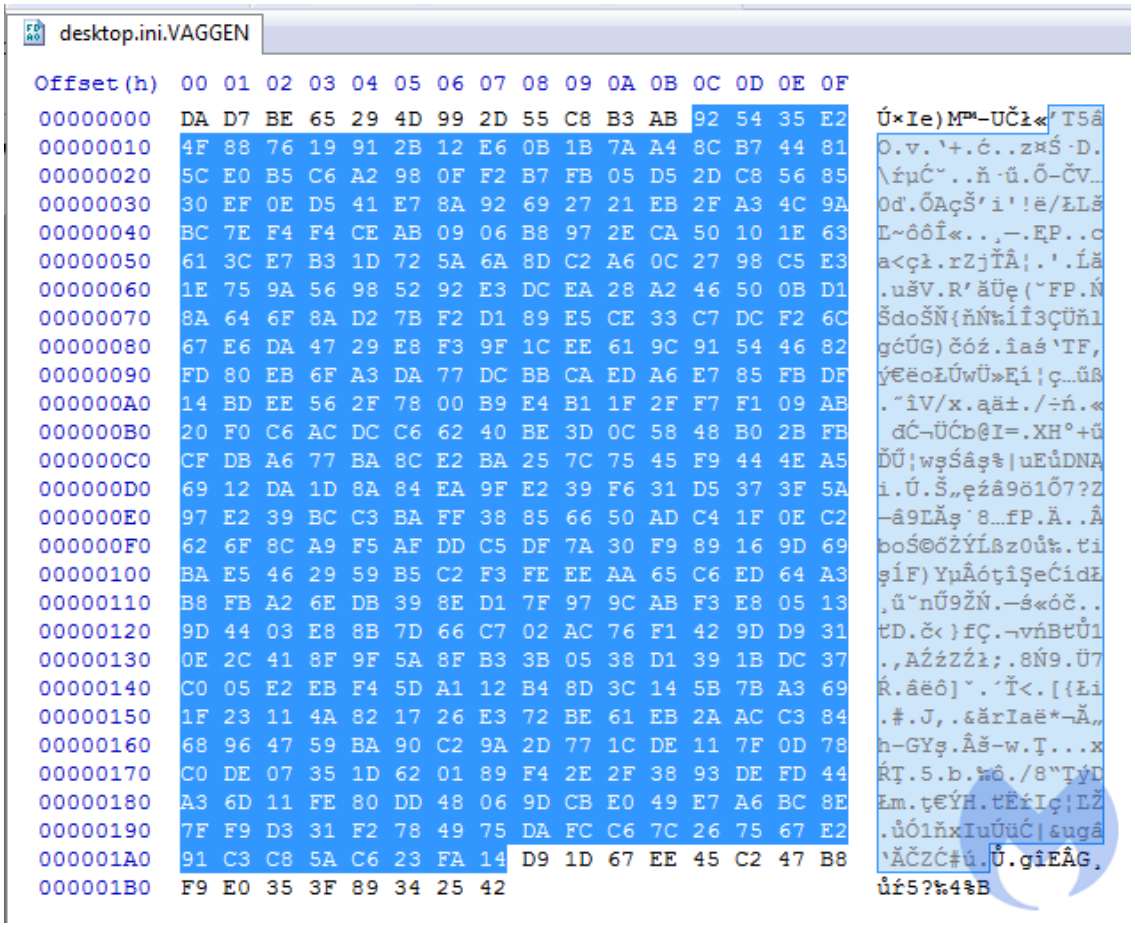
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



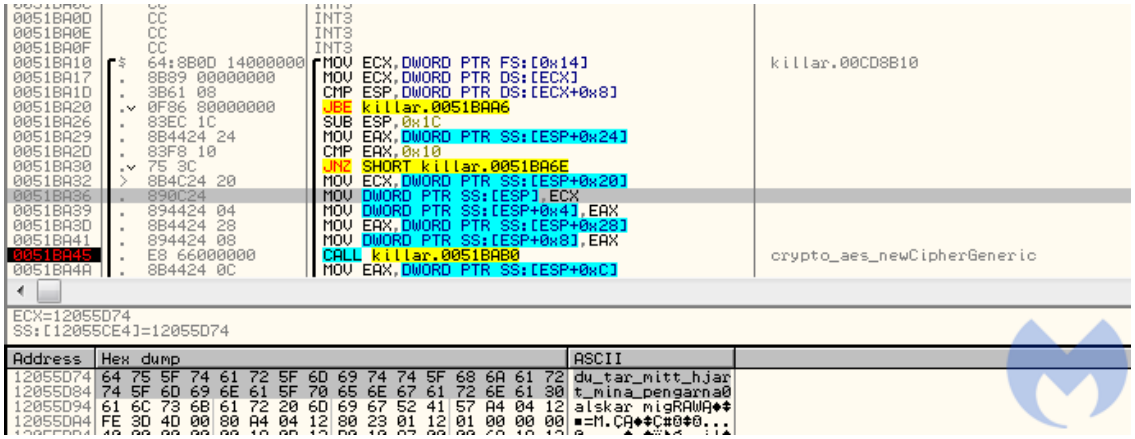
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

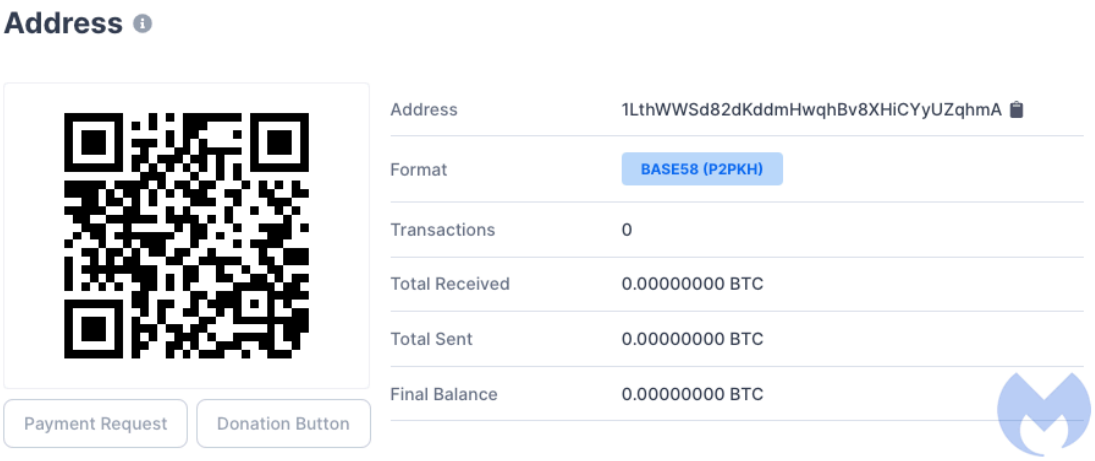
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

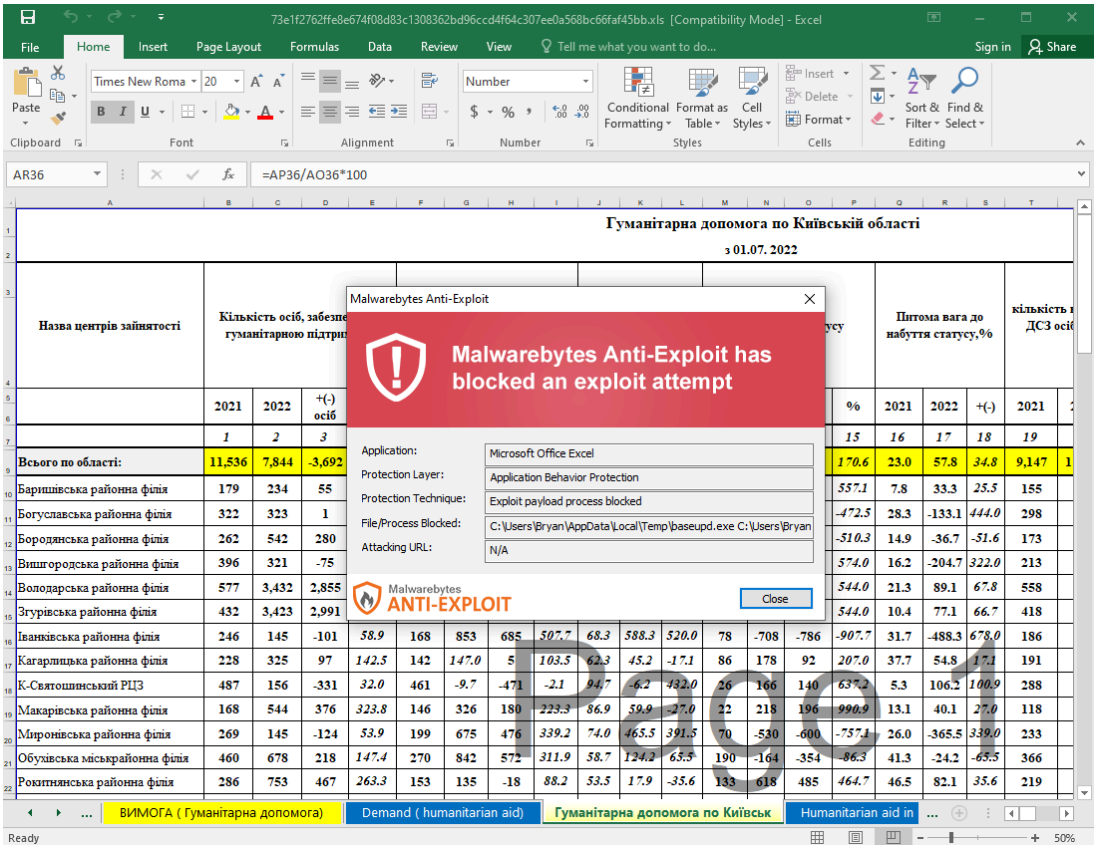
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

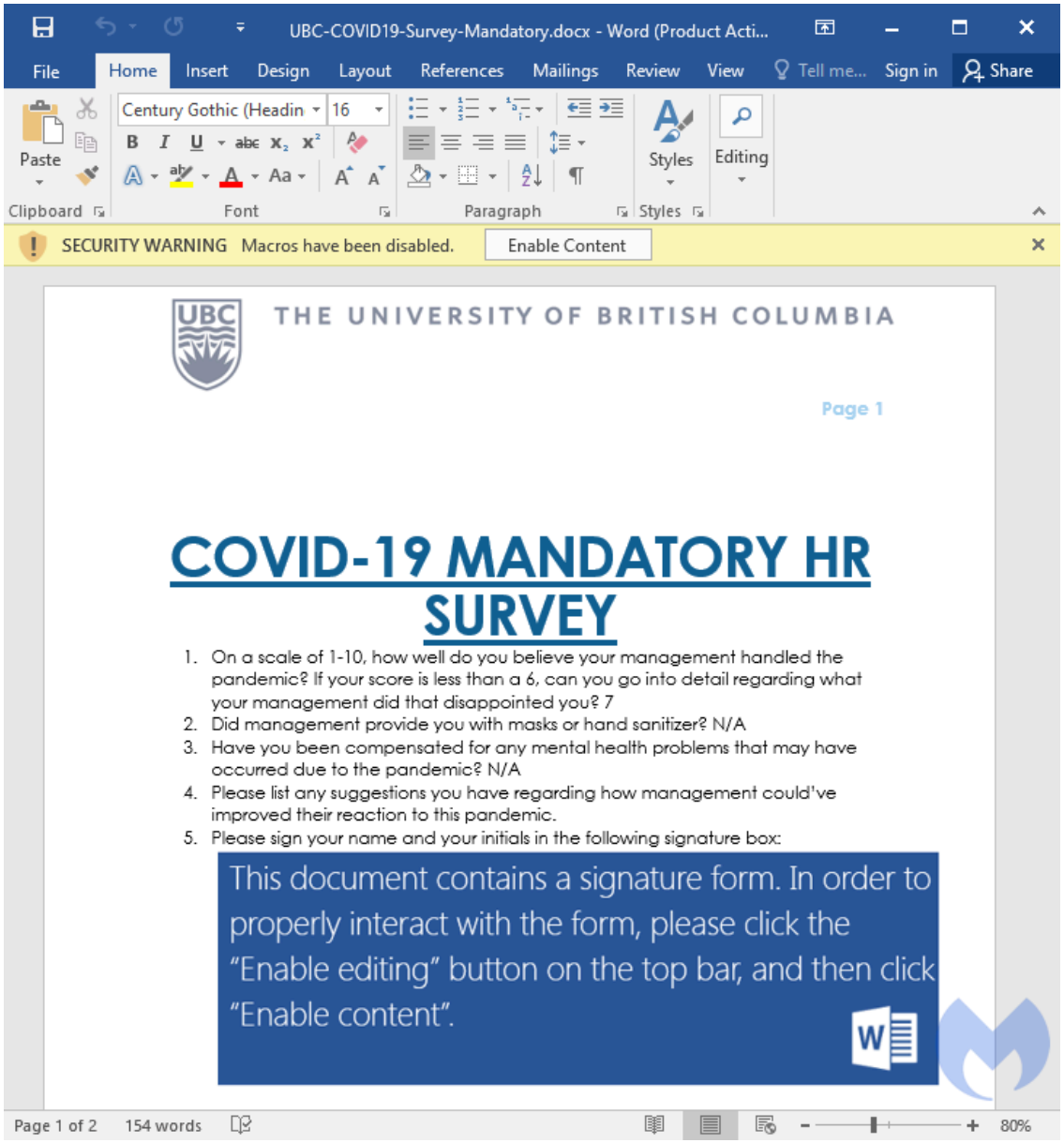
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

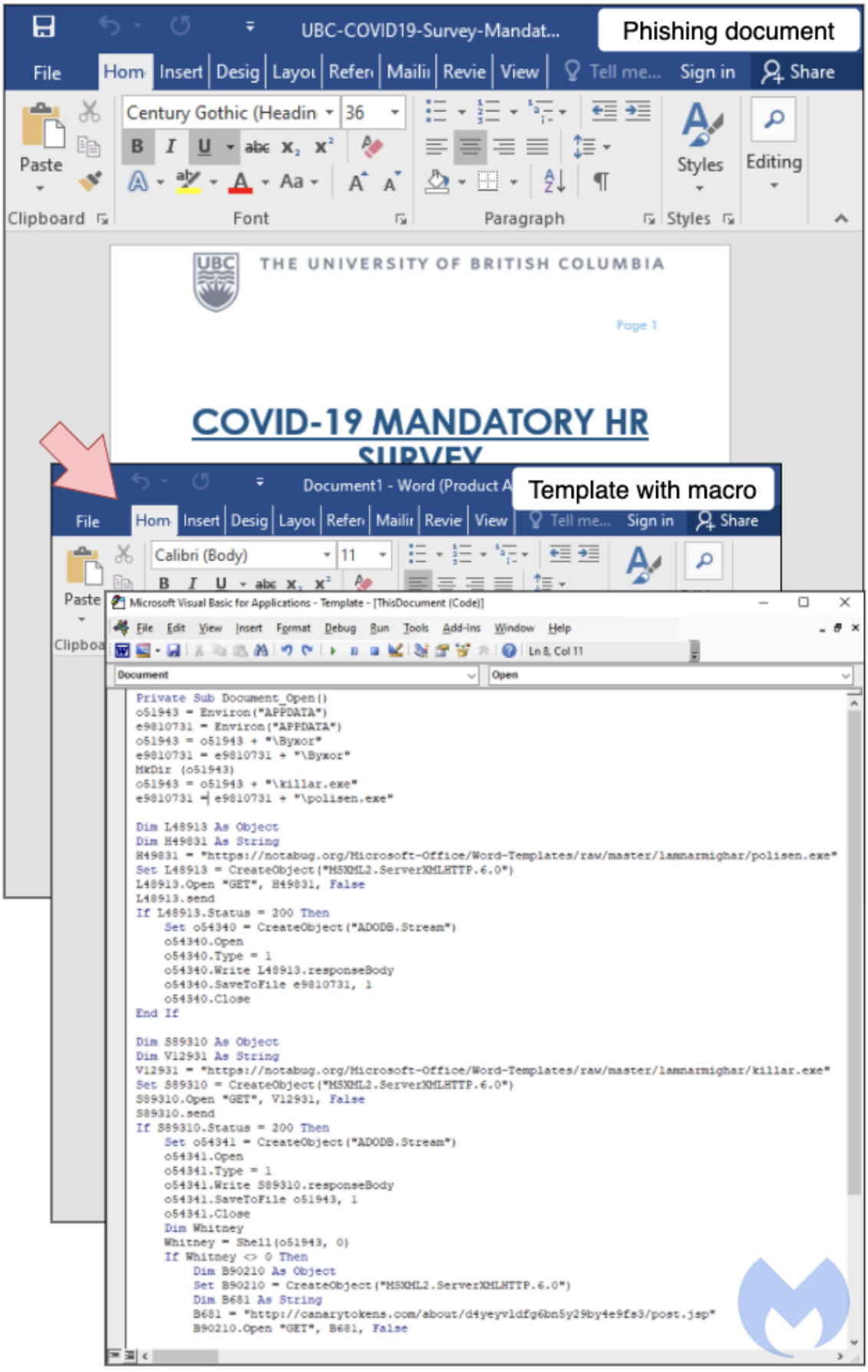
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

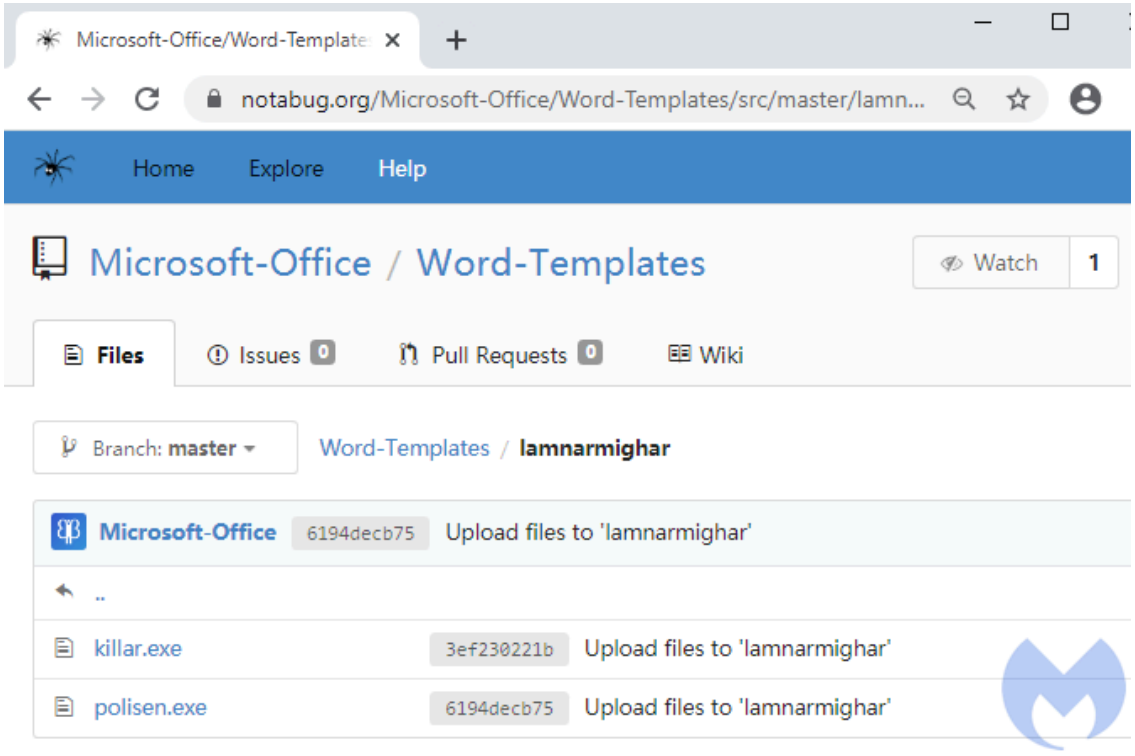
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



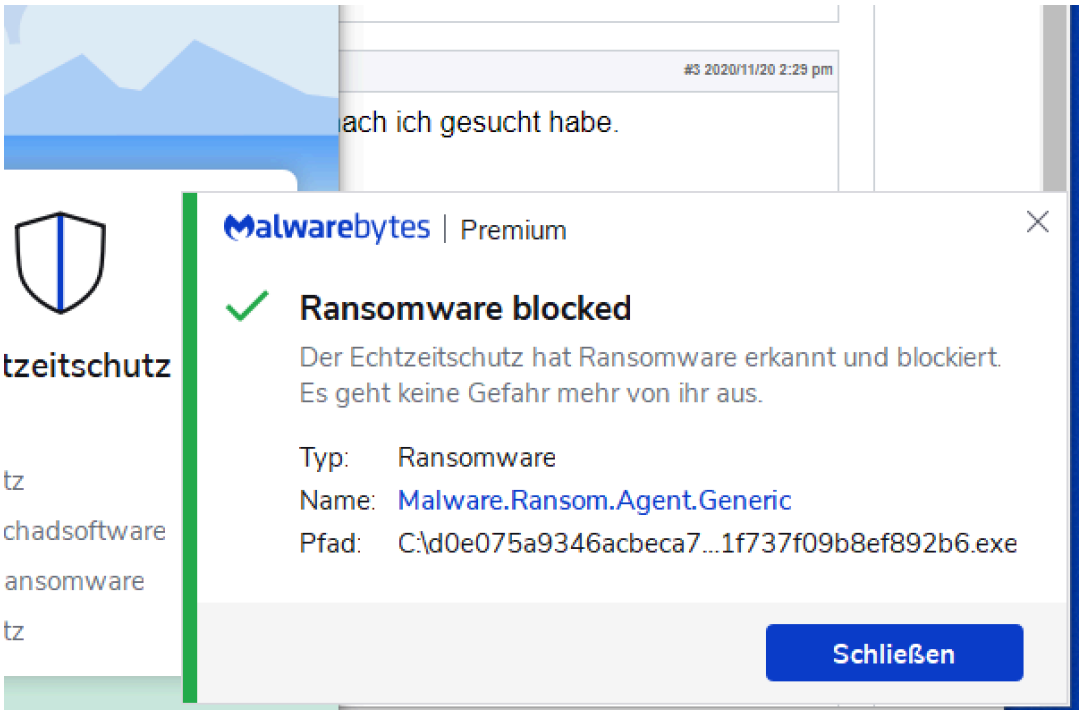
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

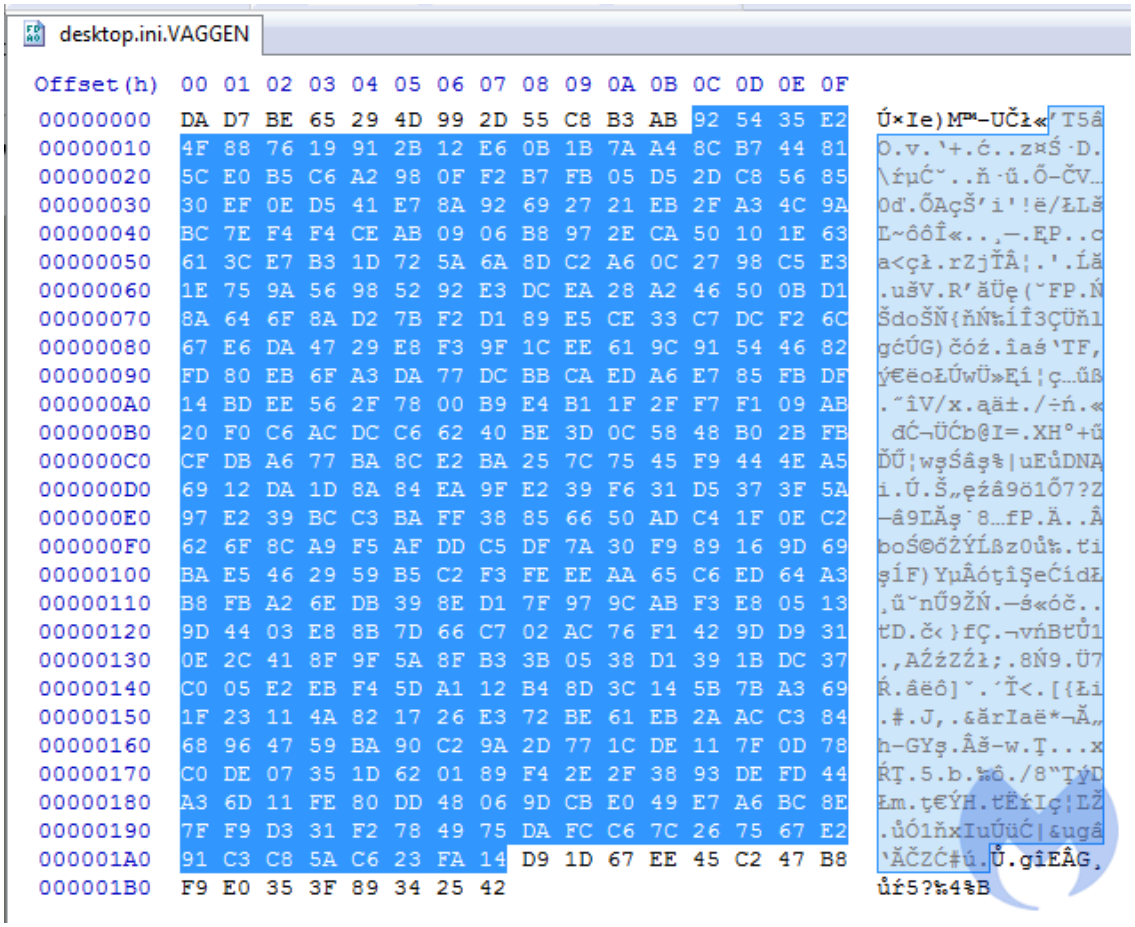
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

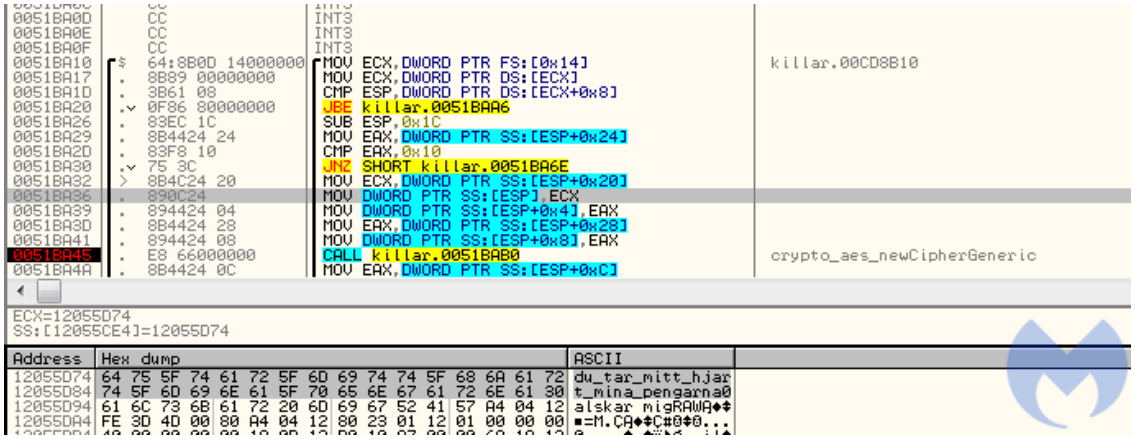


The content of the output file (with .VAGGEN extension) contains:

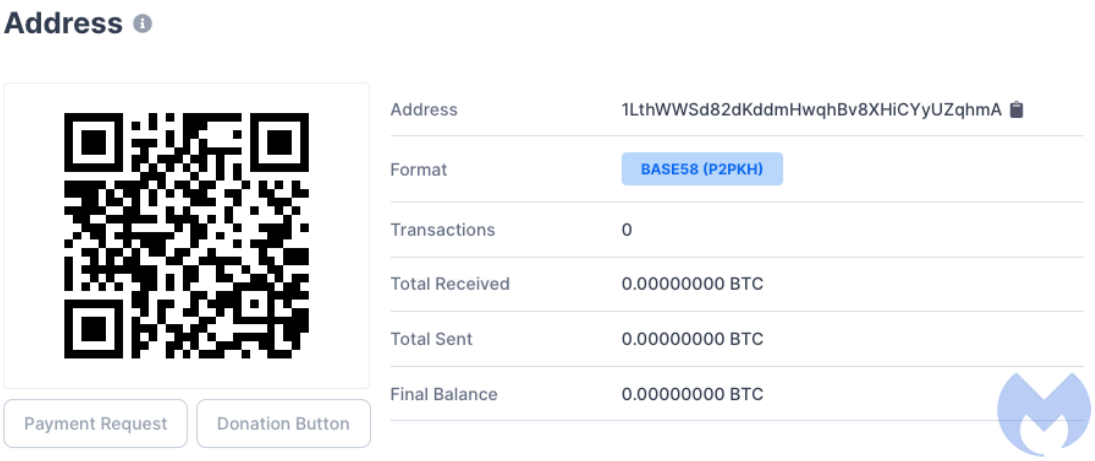
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

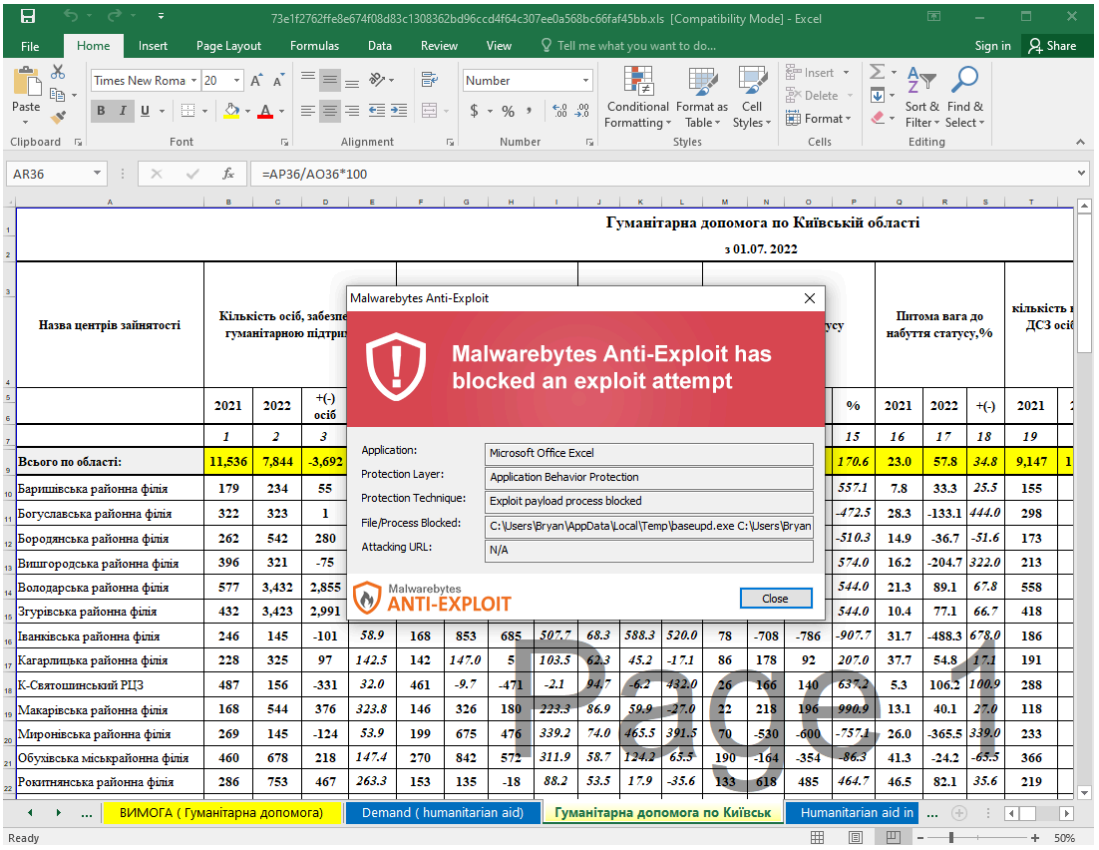
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

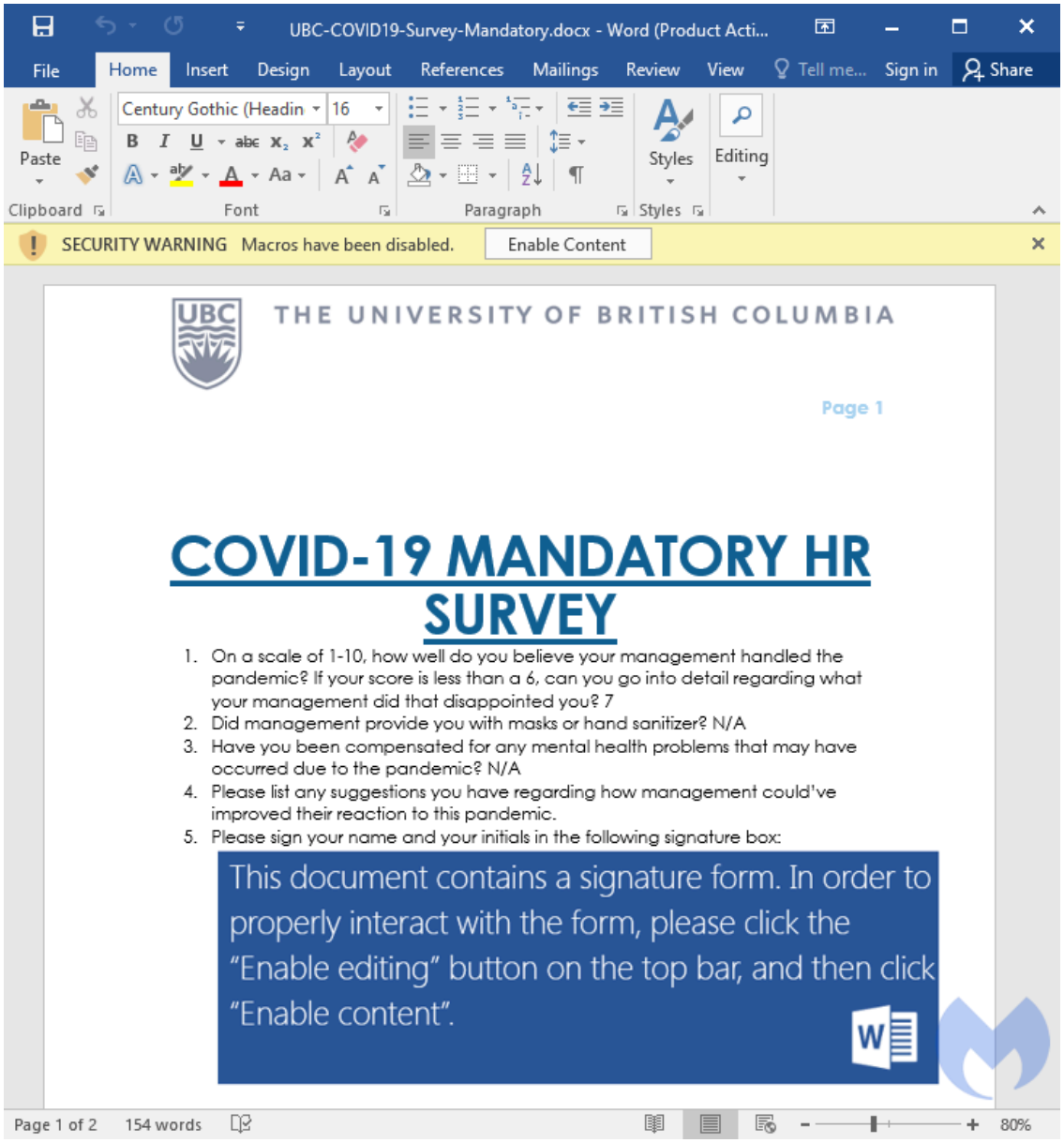
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

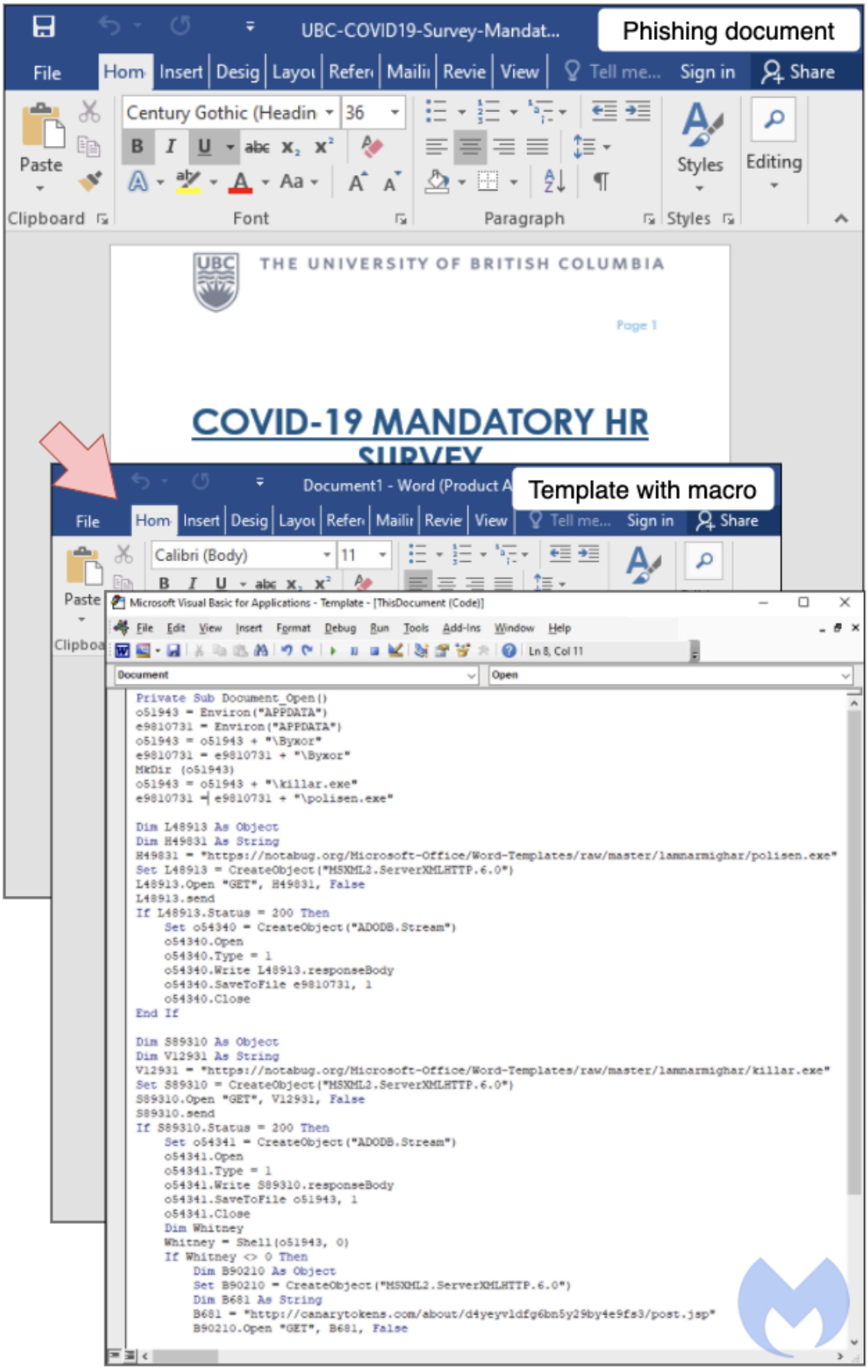
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

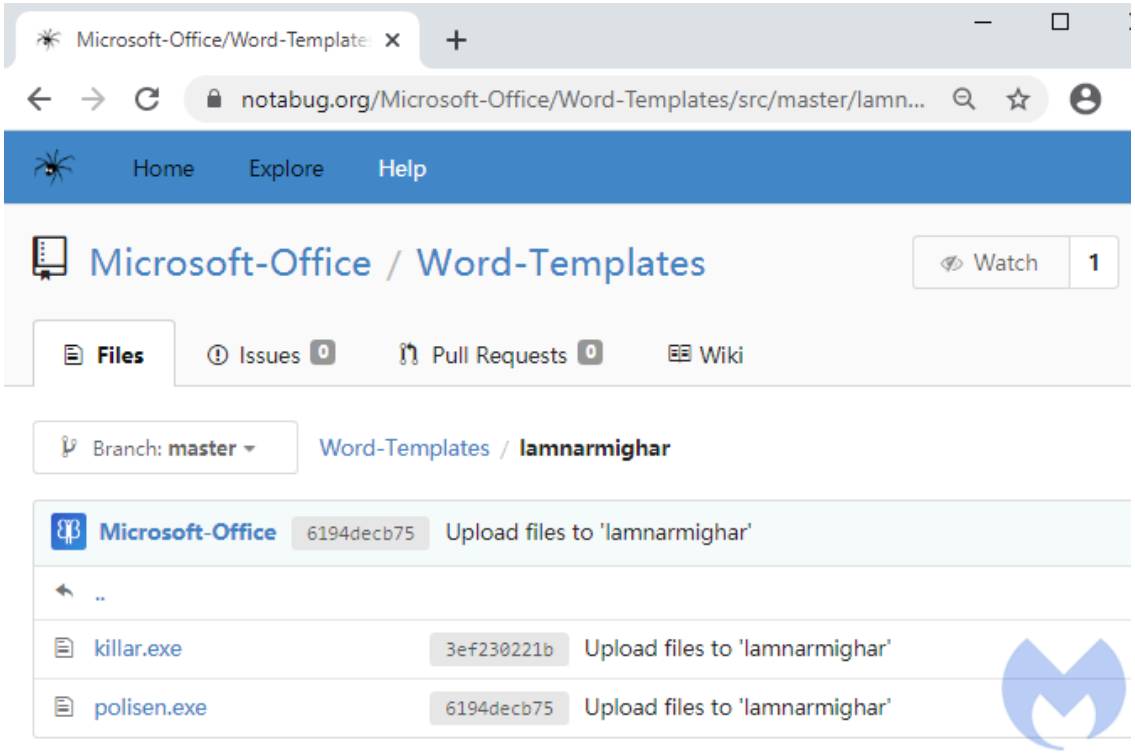
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



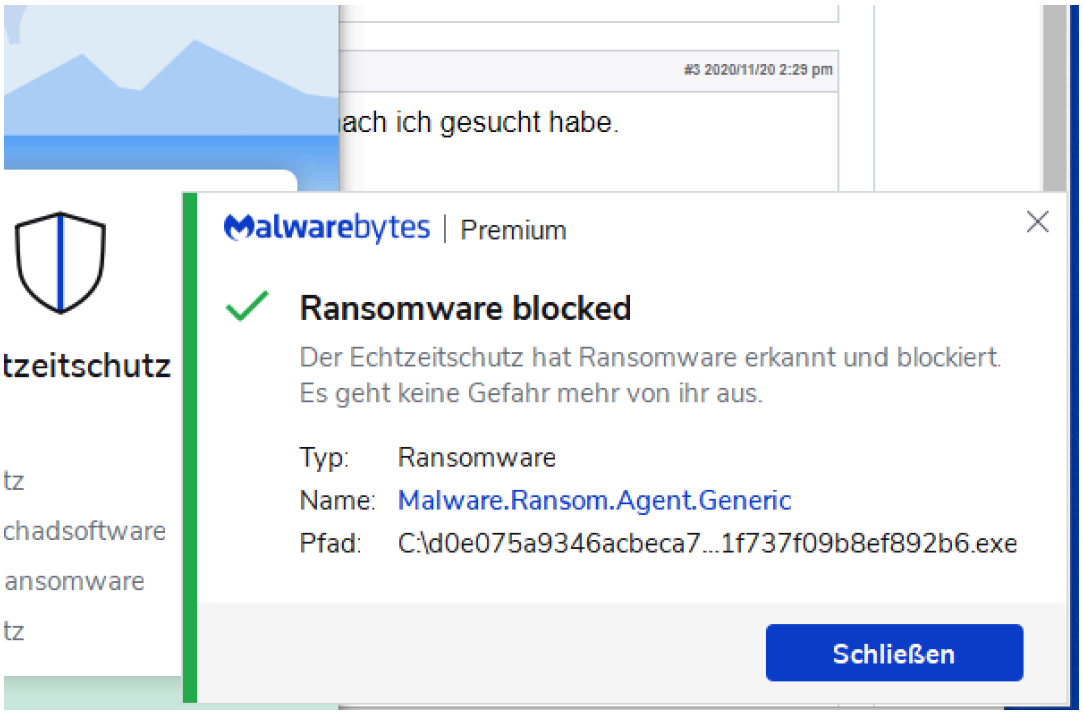
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

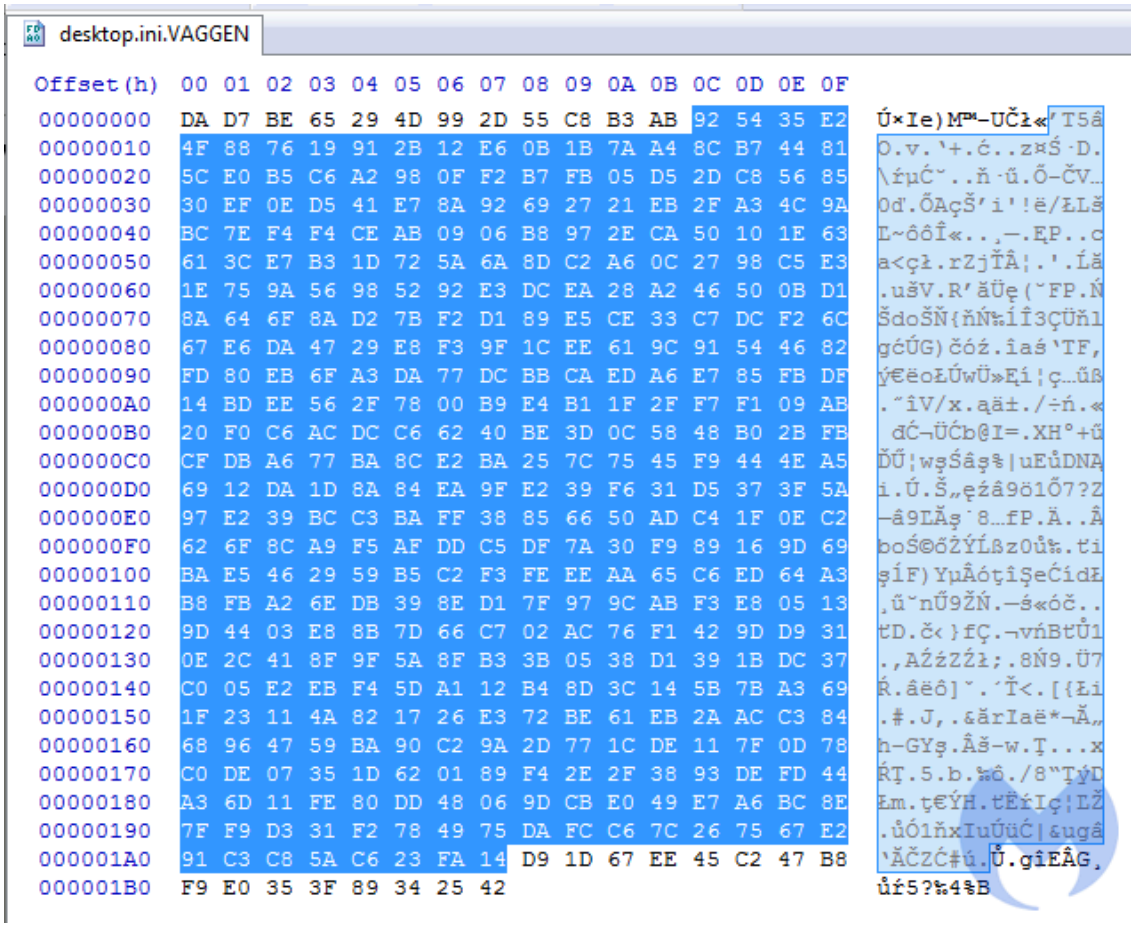
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

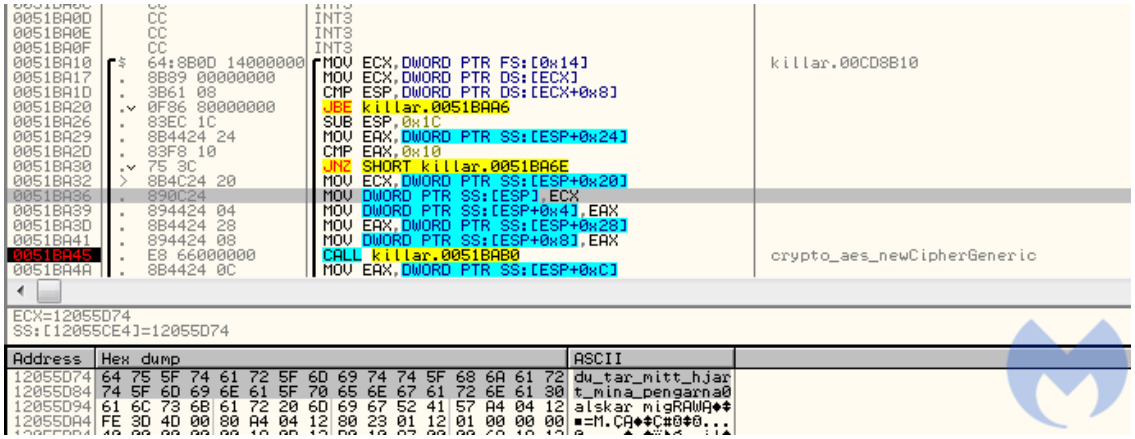


The content of the output file (with .VAGGEN extension) contains:

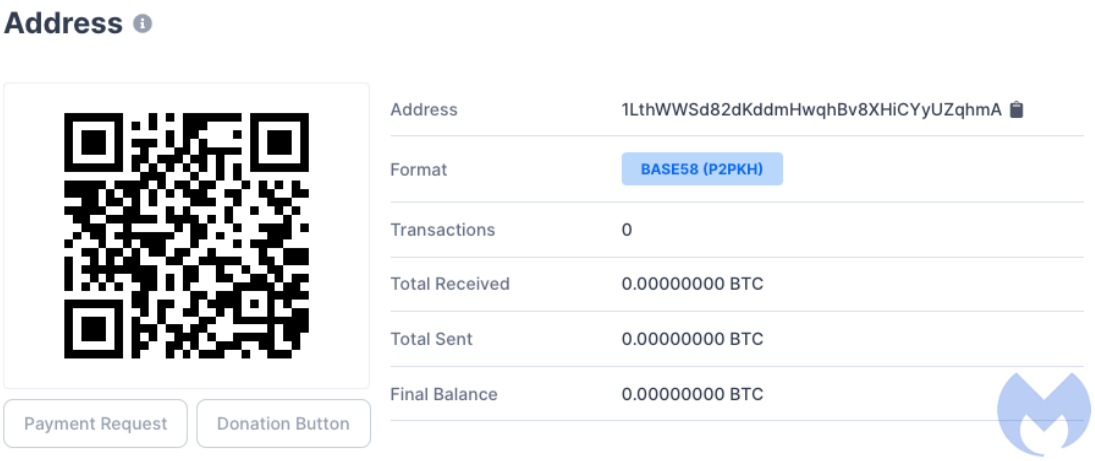
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

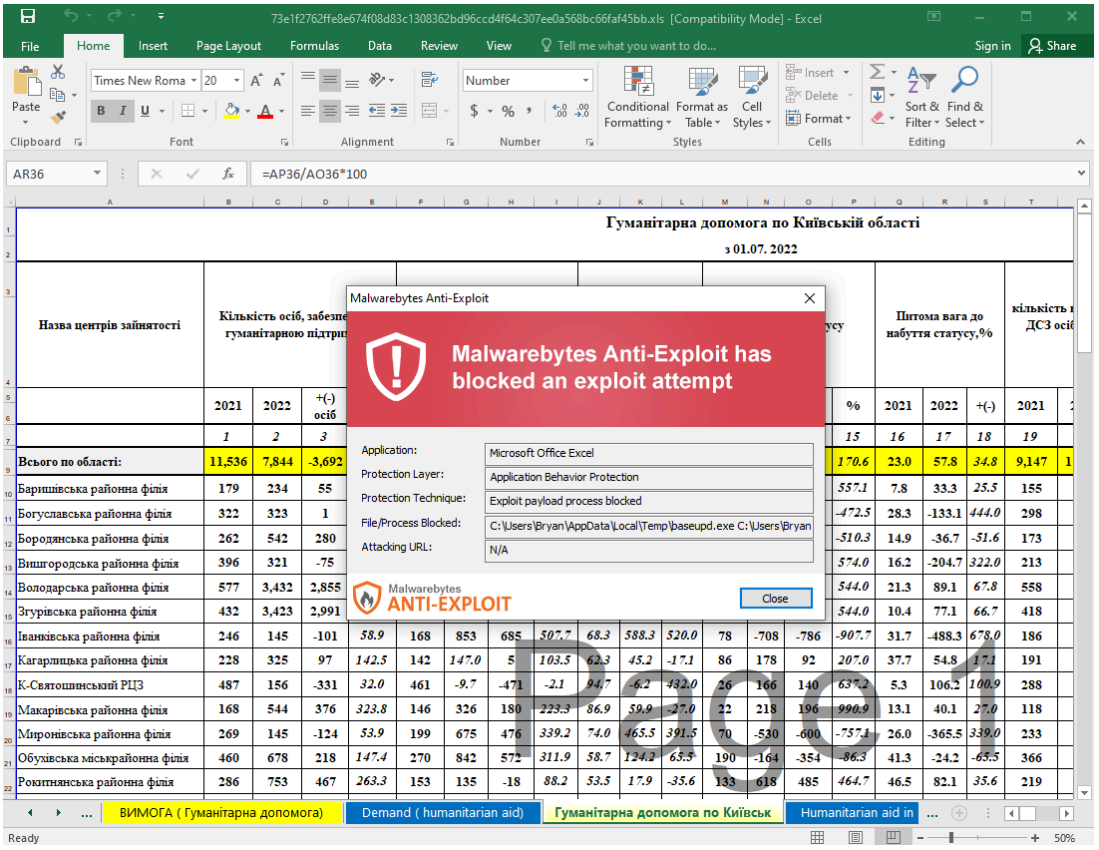
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

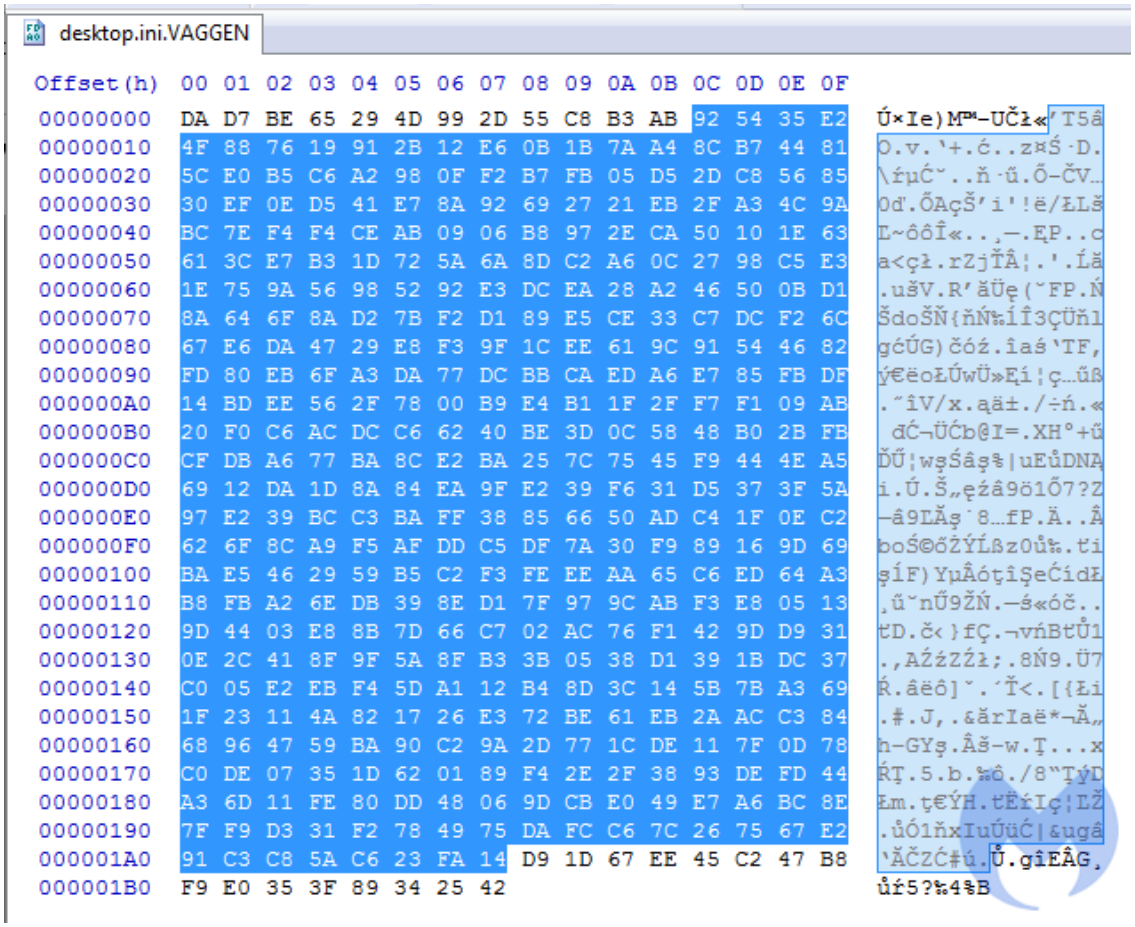
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



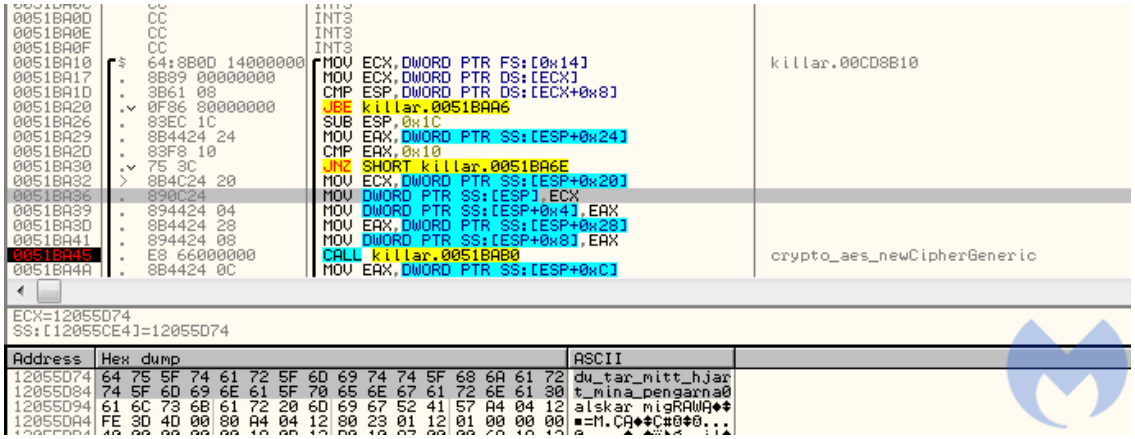
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

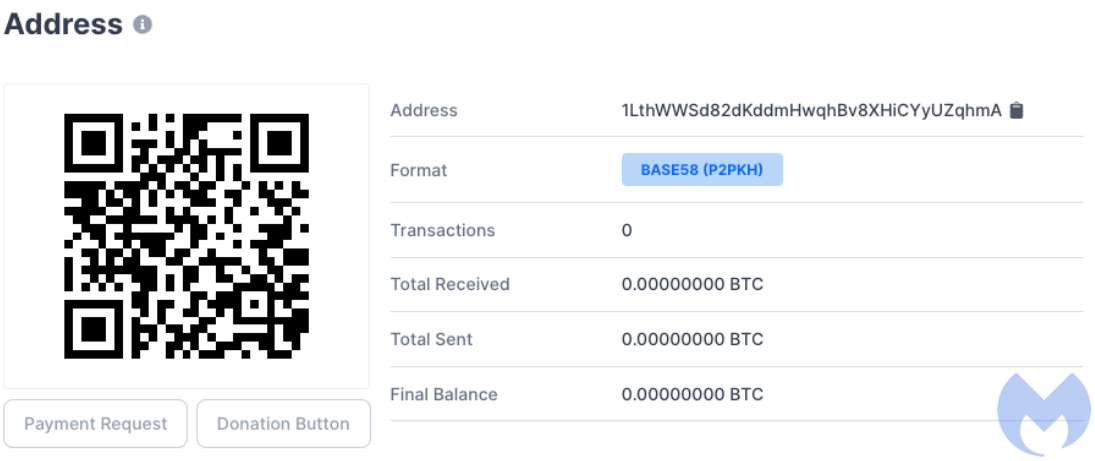
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

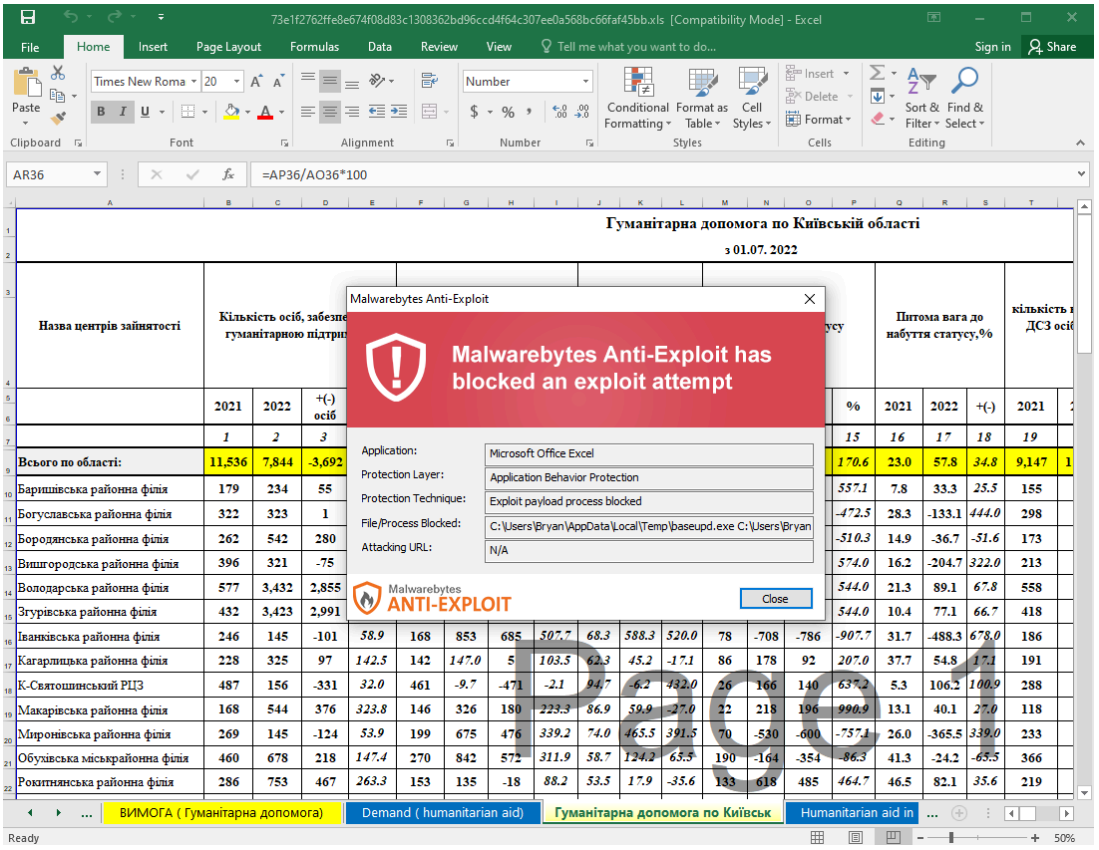
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

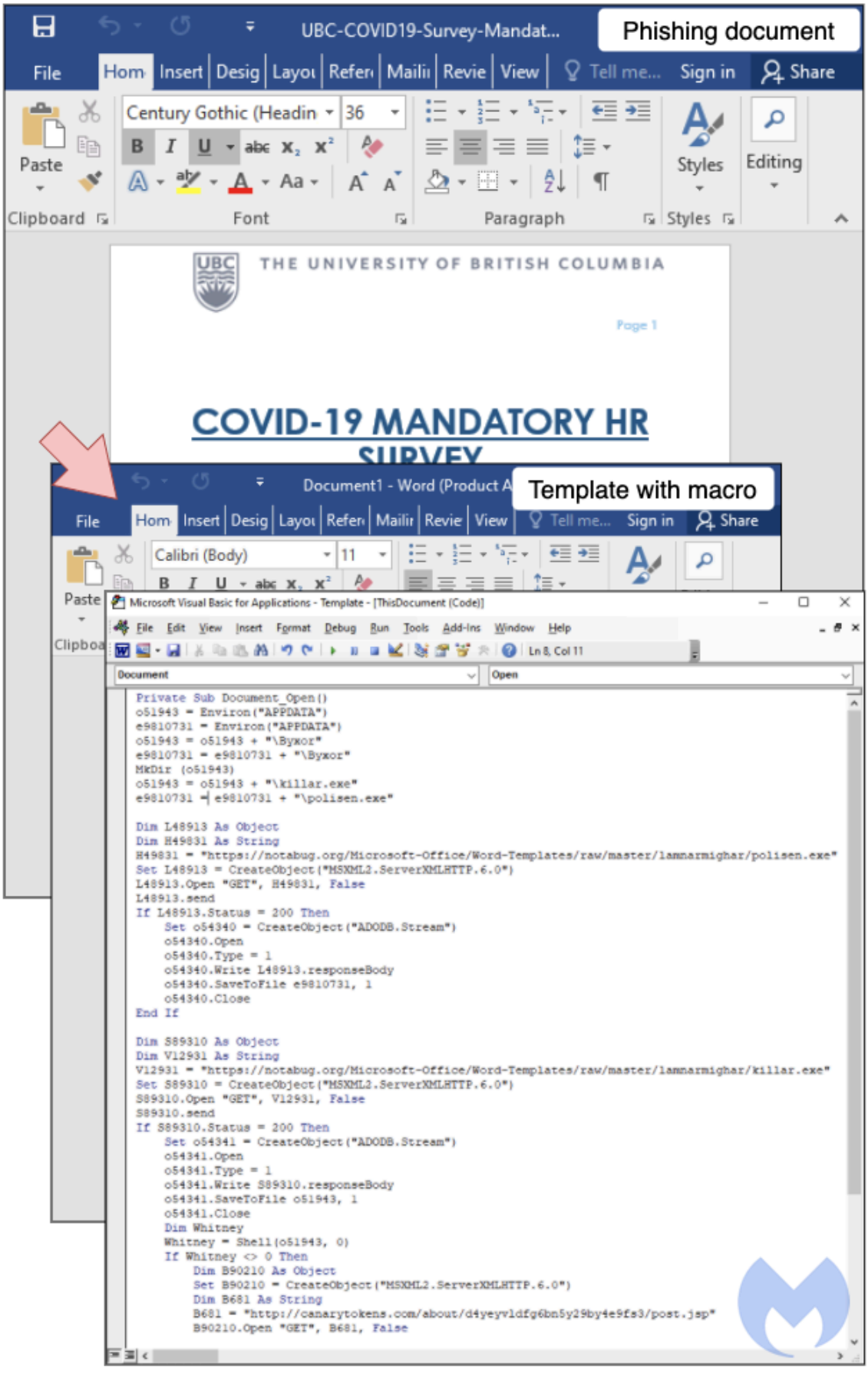
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

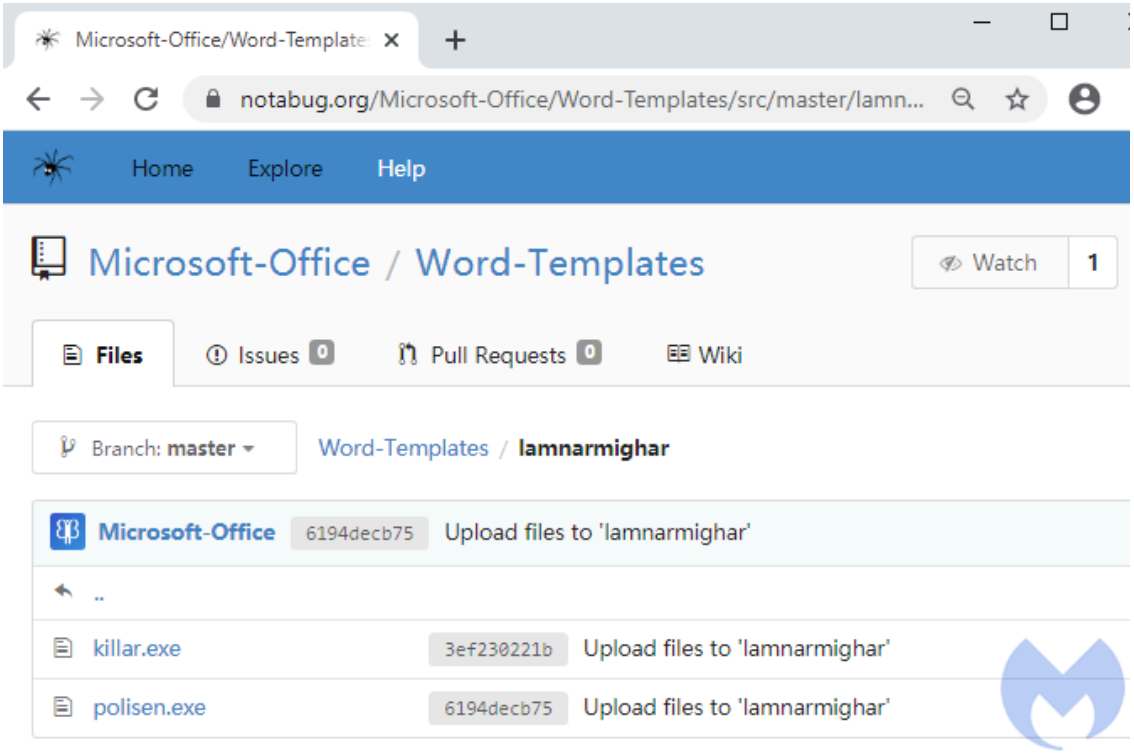
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



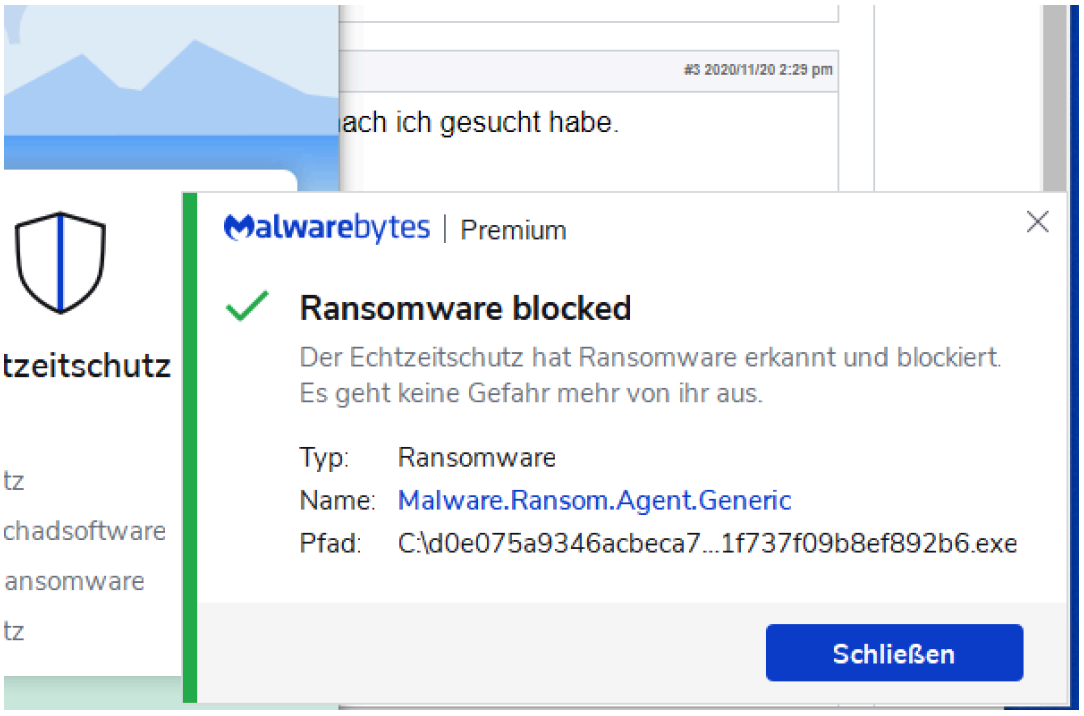
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

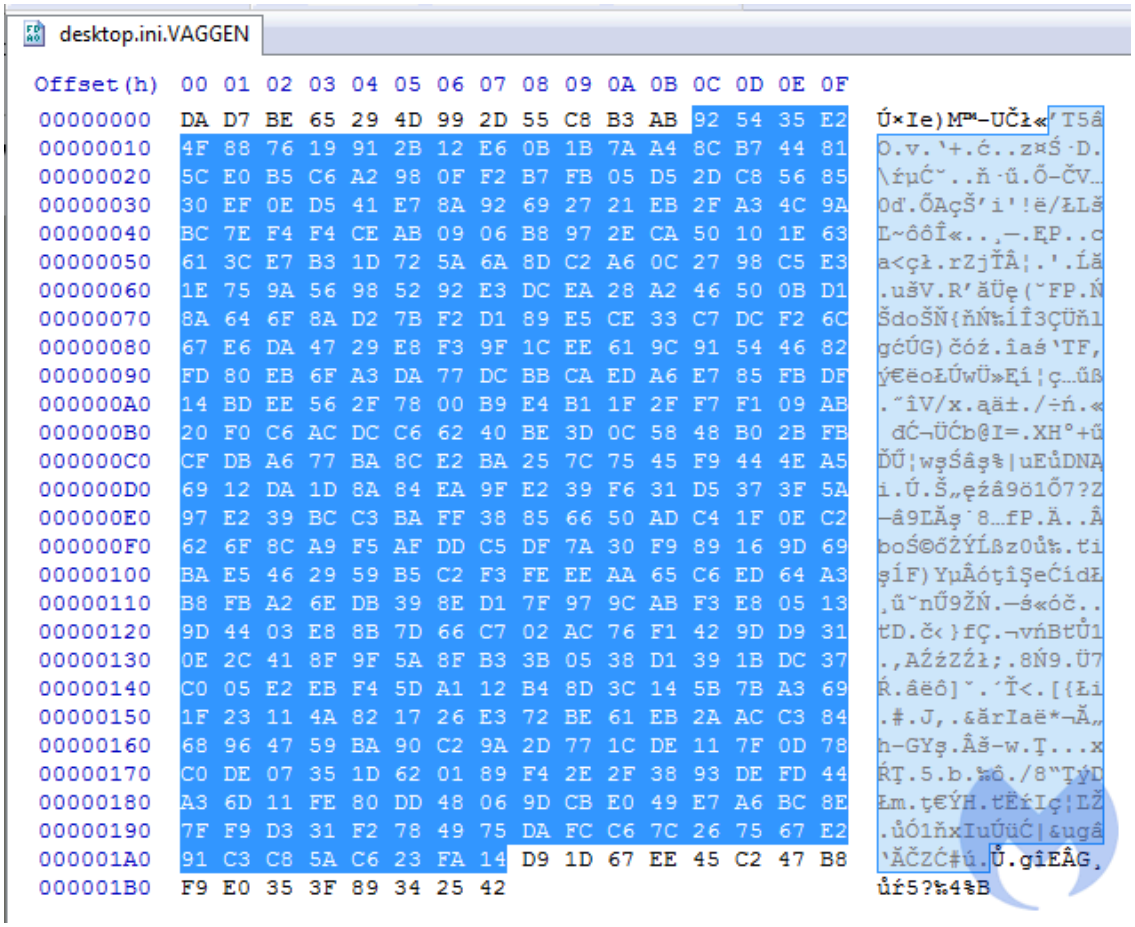
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

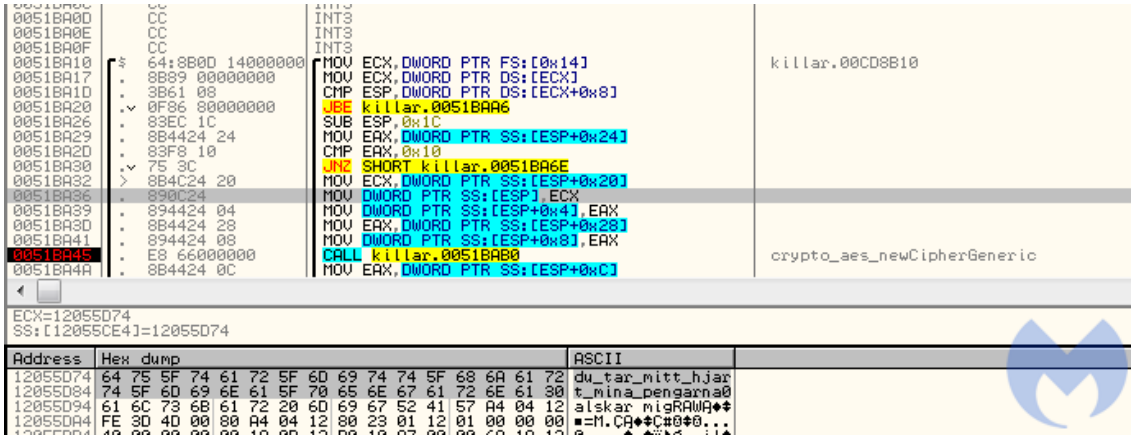


The content of the output file (with .VAGGEN extension) contains:

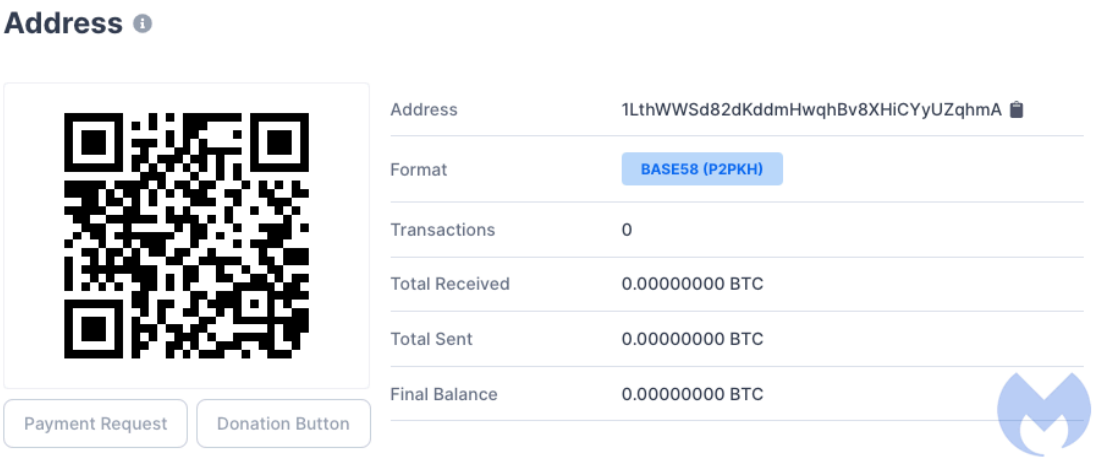
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

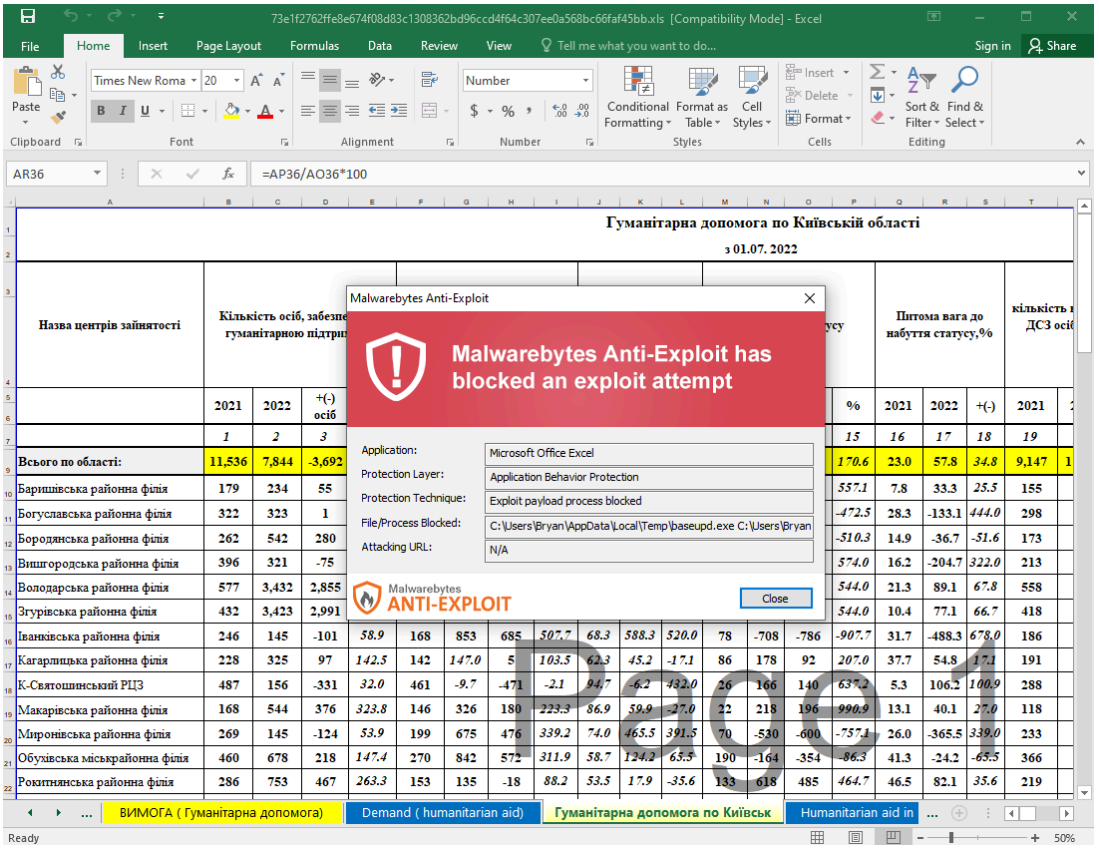
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

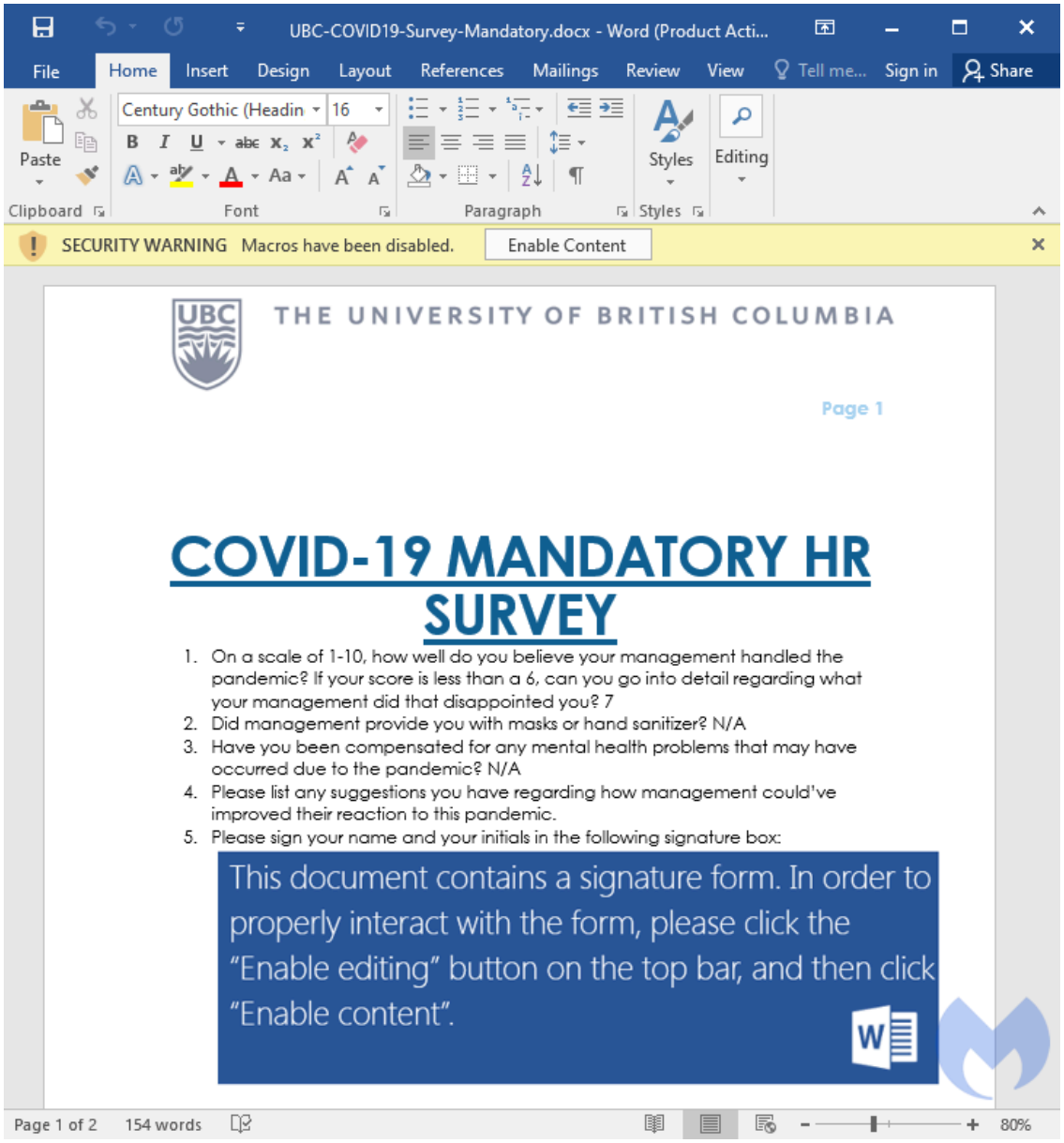
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

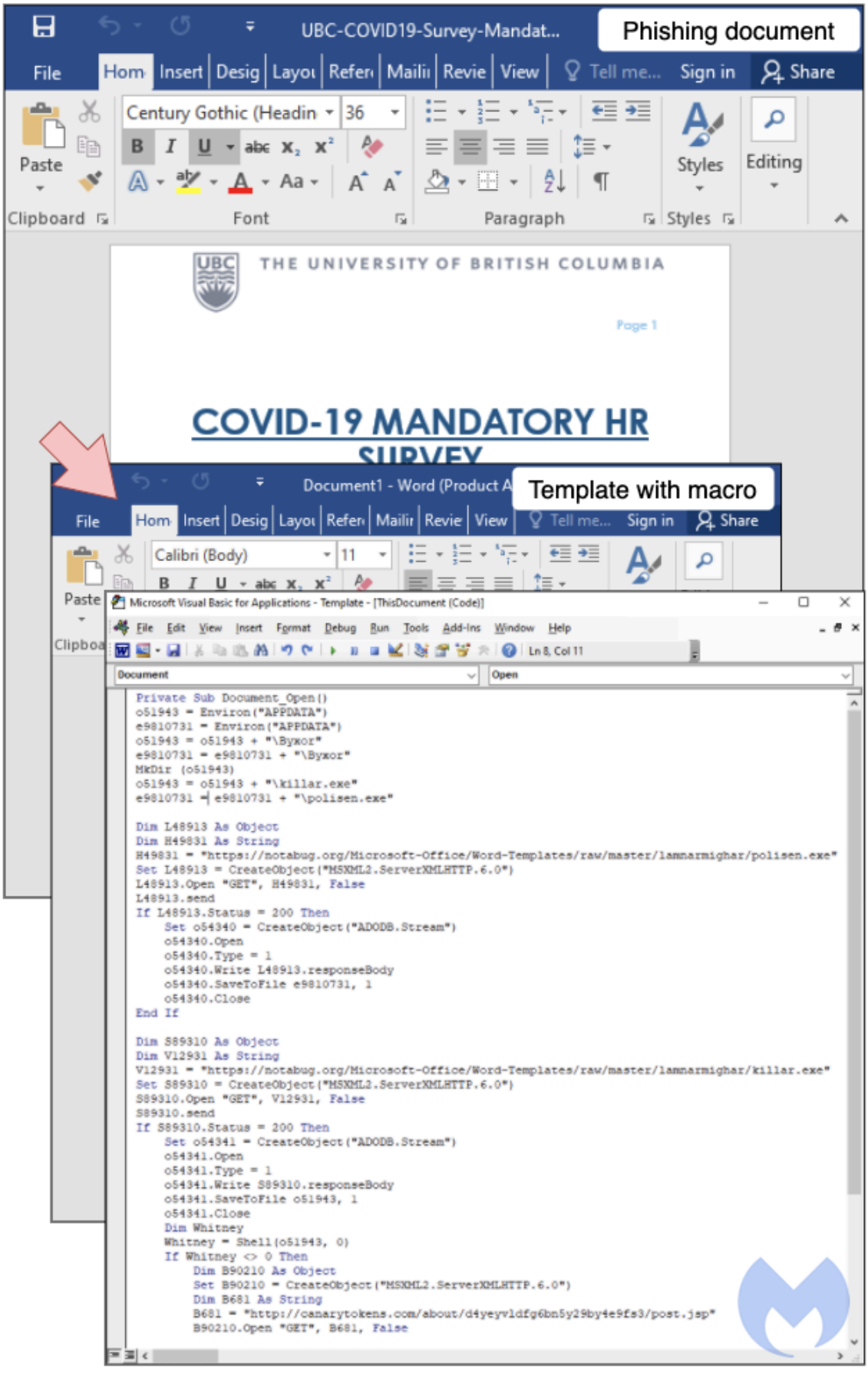
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

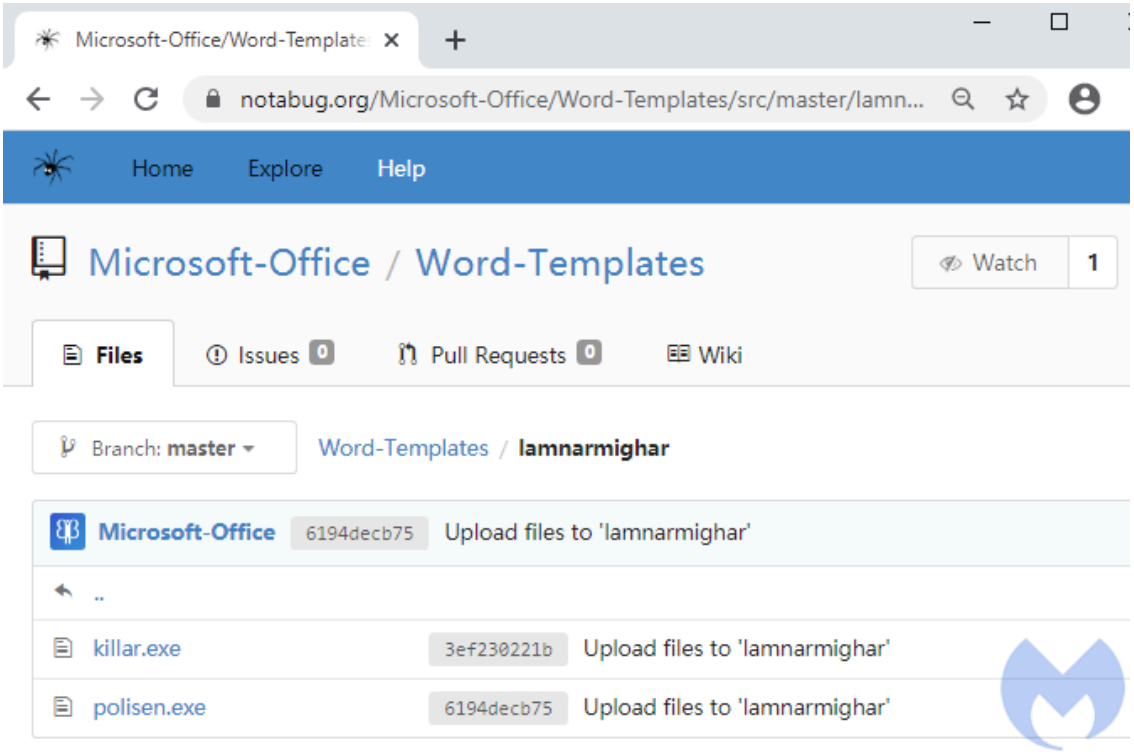
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



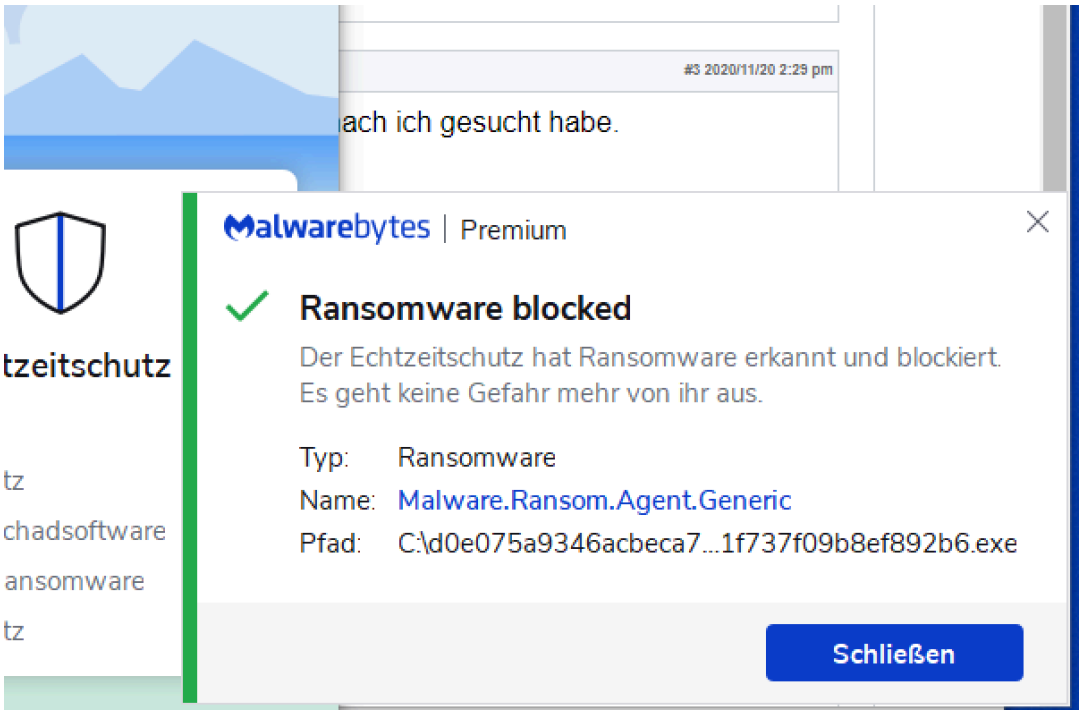
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

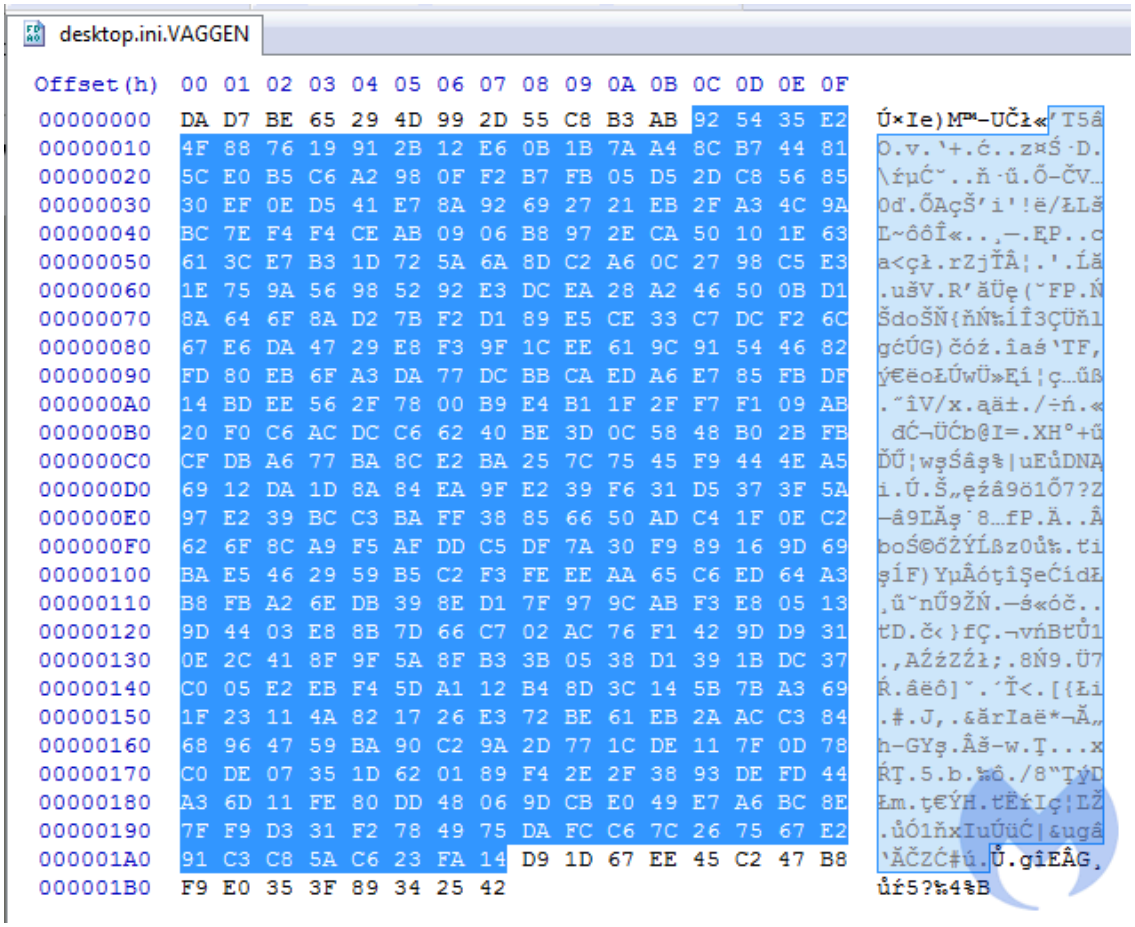
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



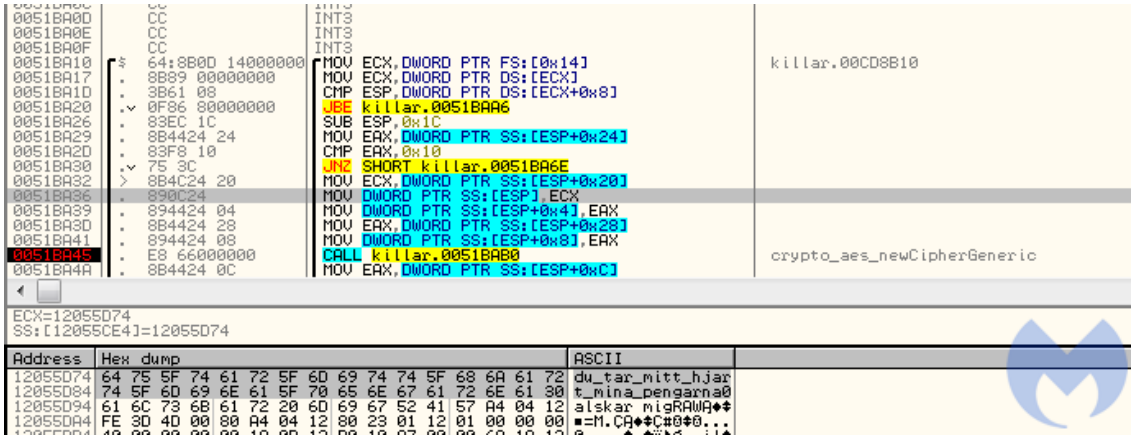
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

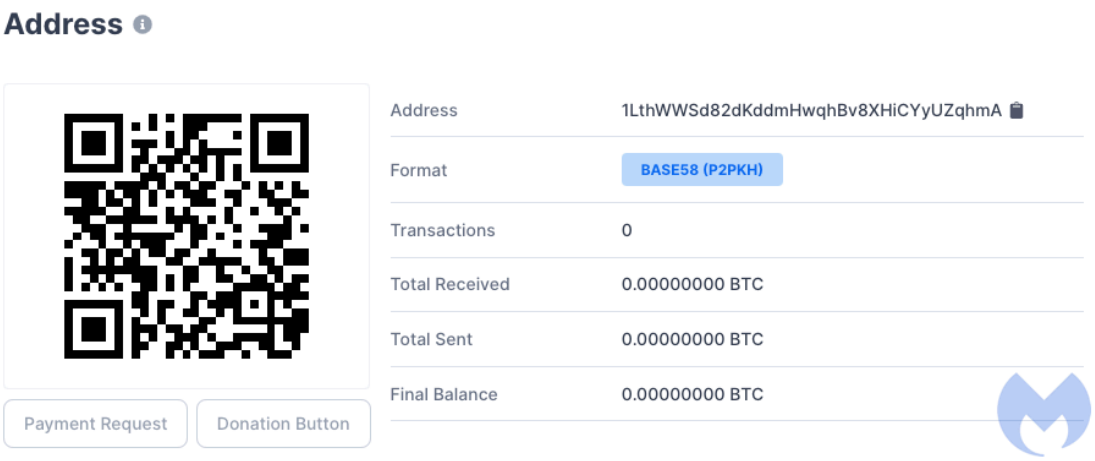
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

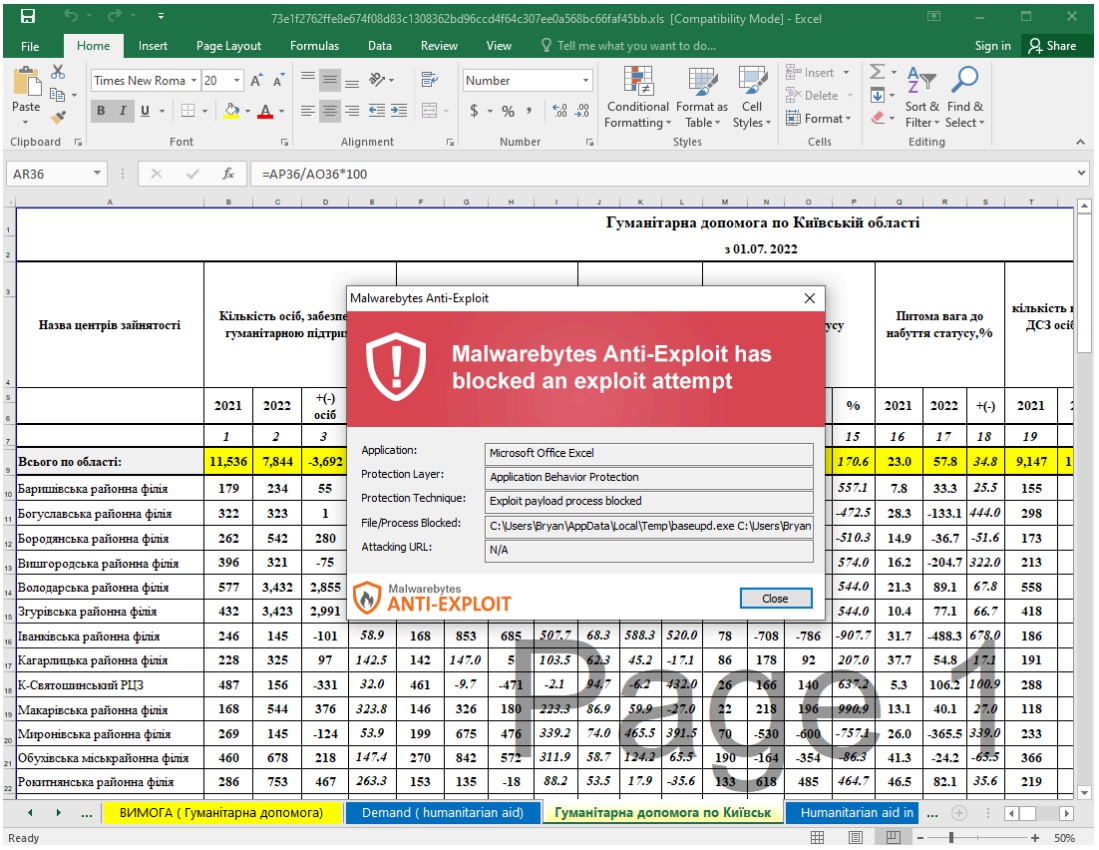
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

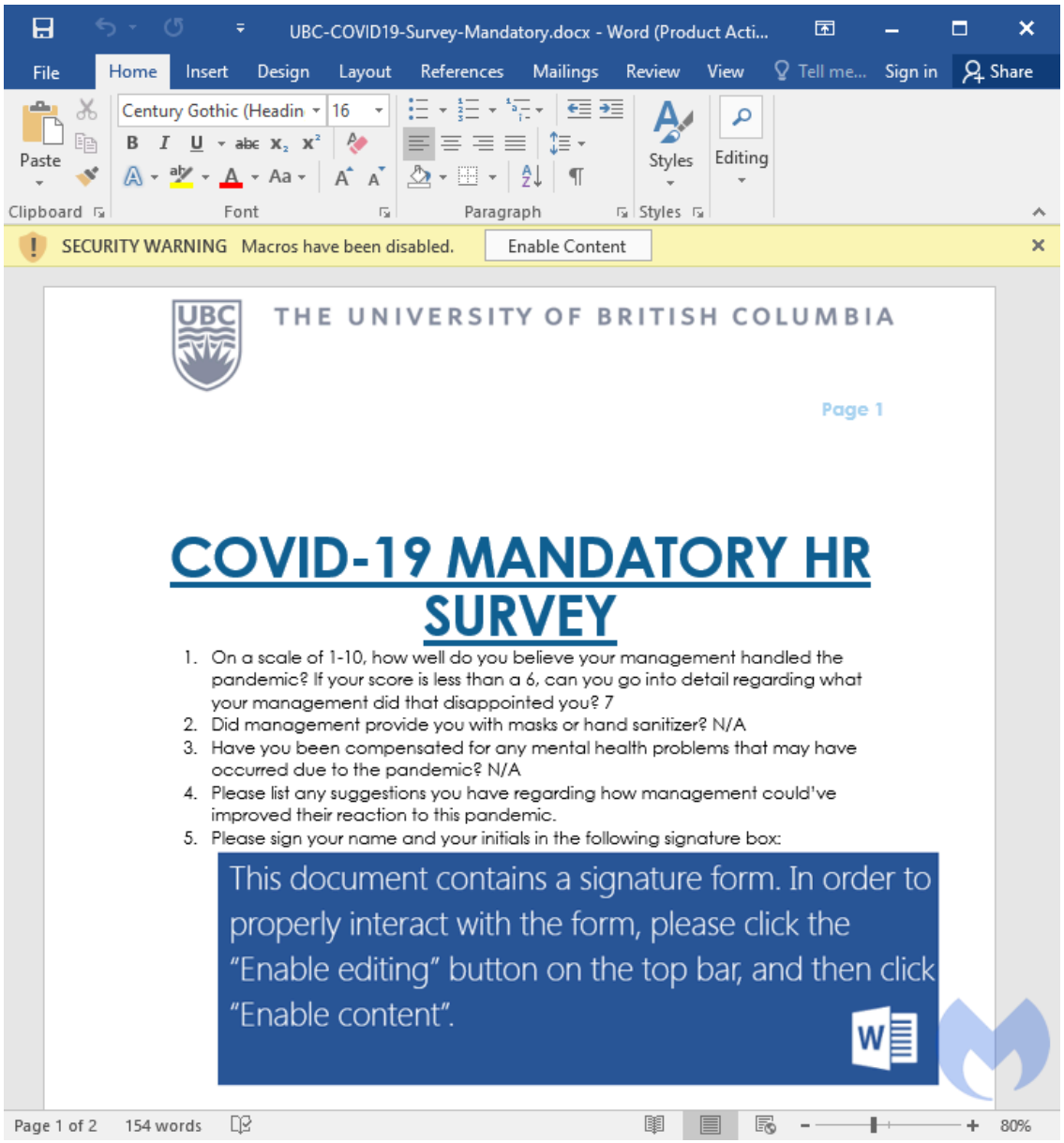
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

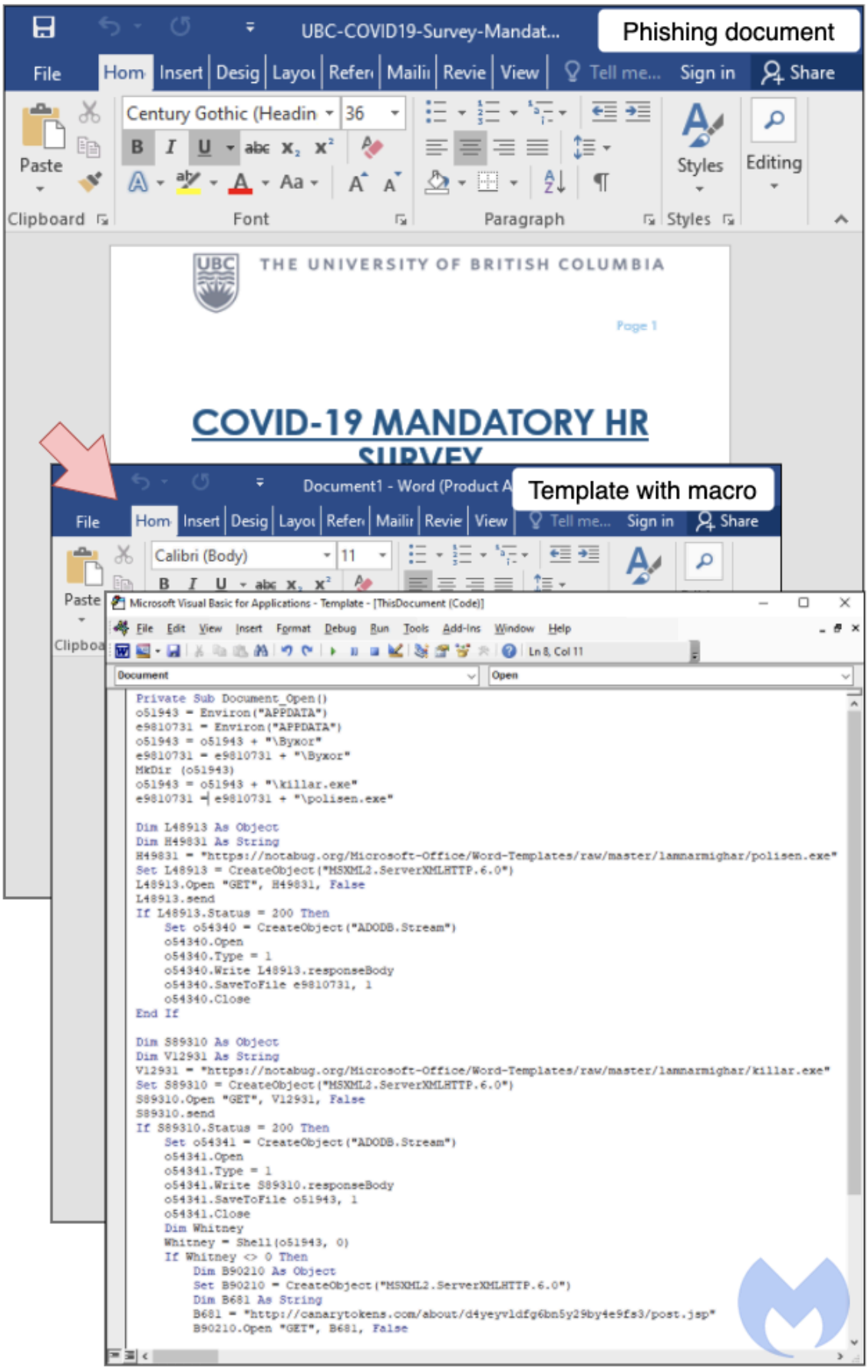
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

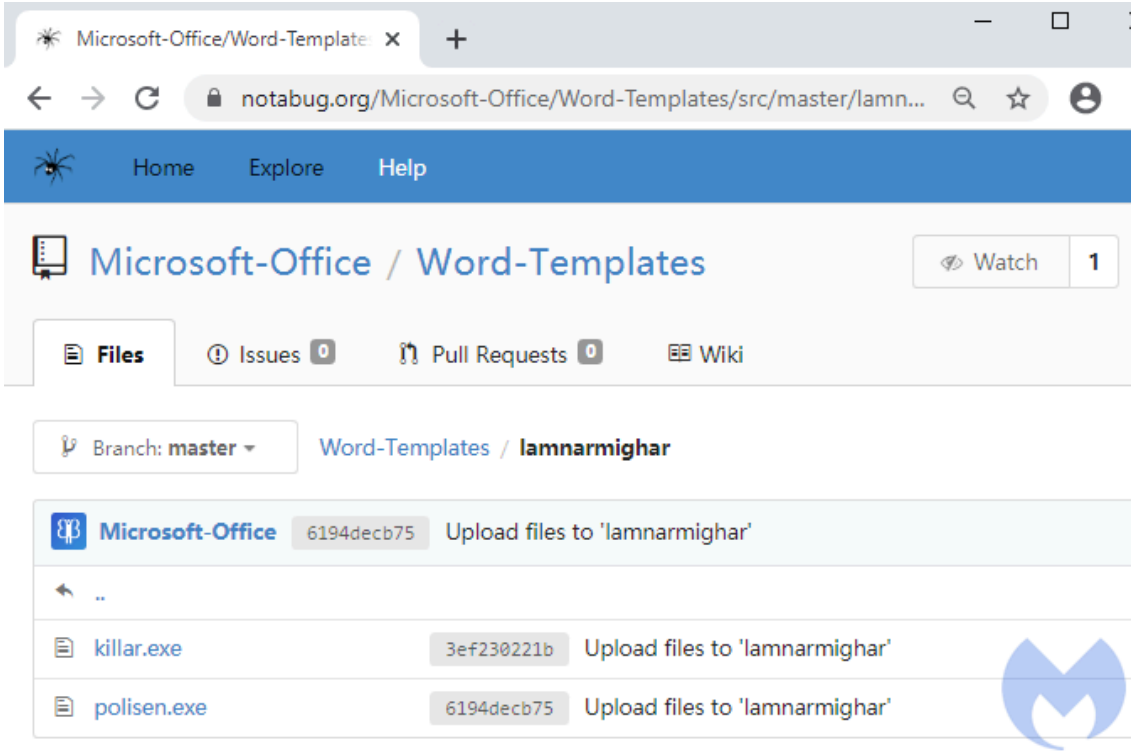
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



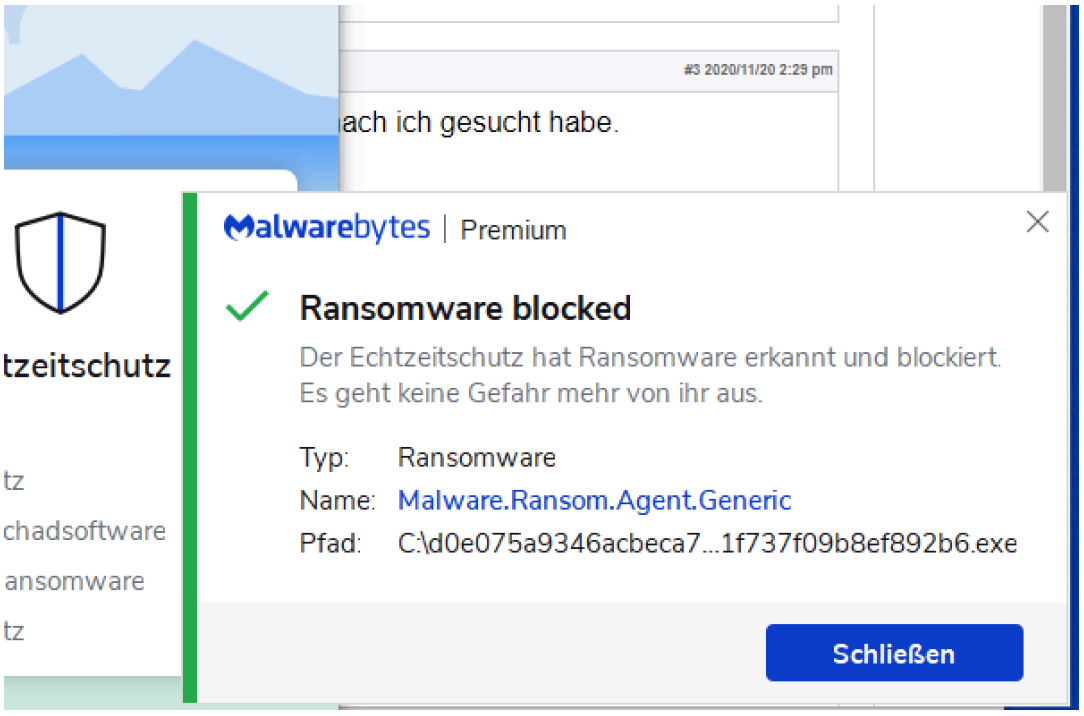
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

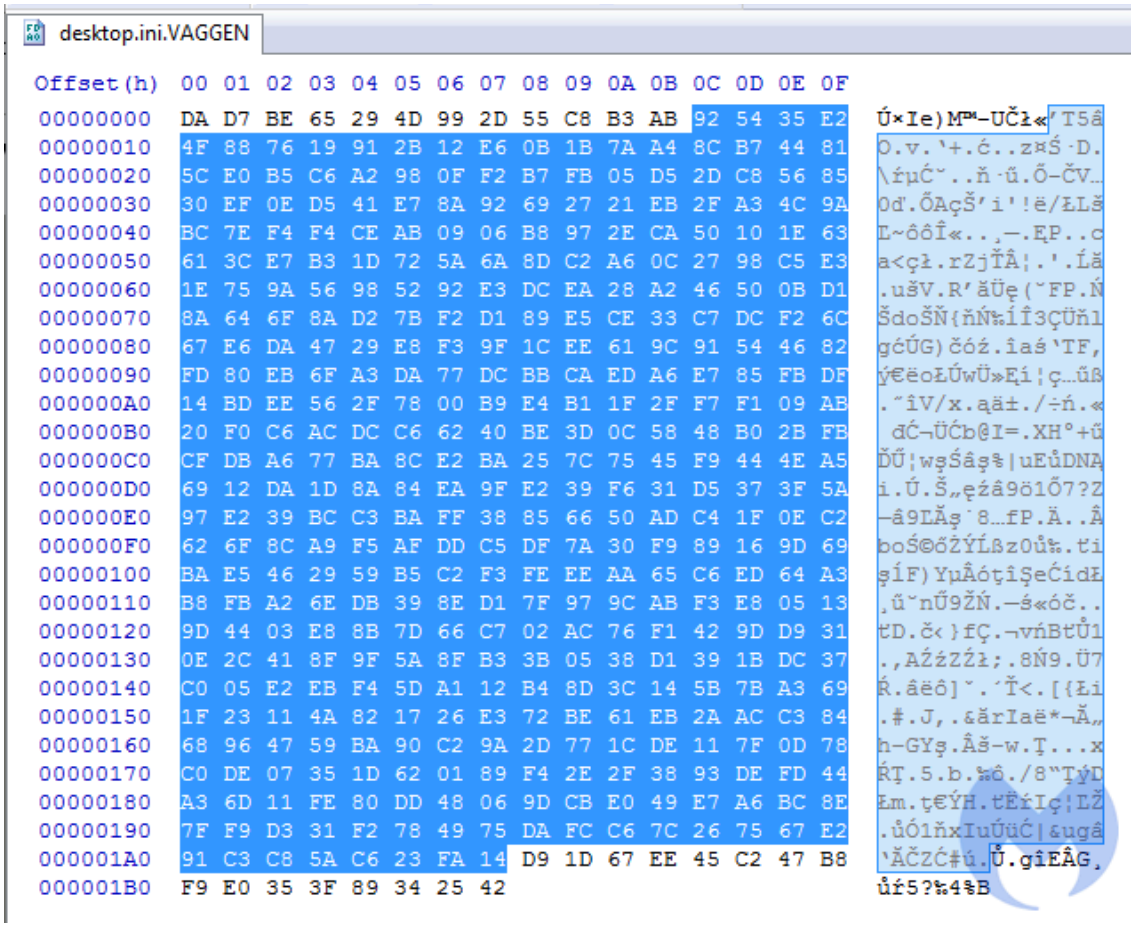
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

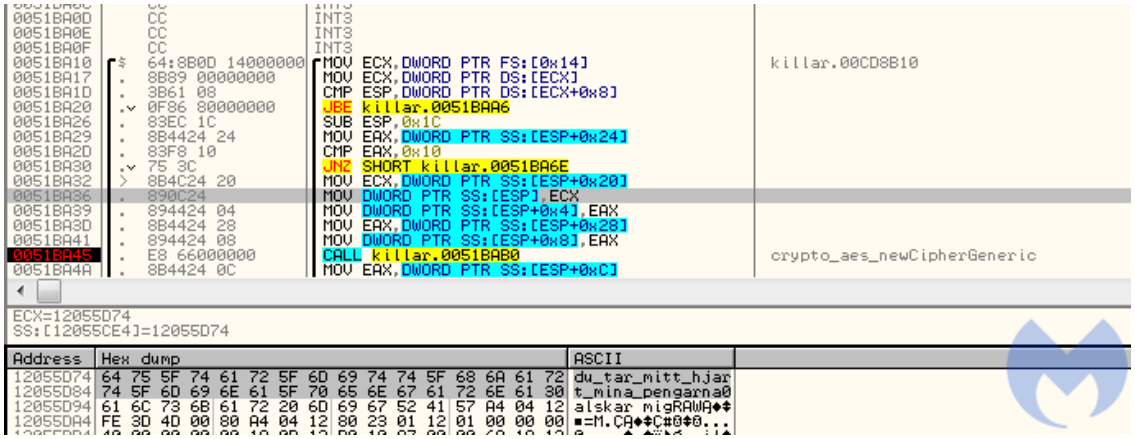


The content of the output file (with .VAGGEN extension) contains:

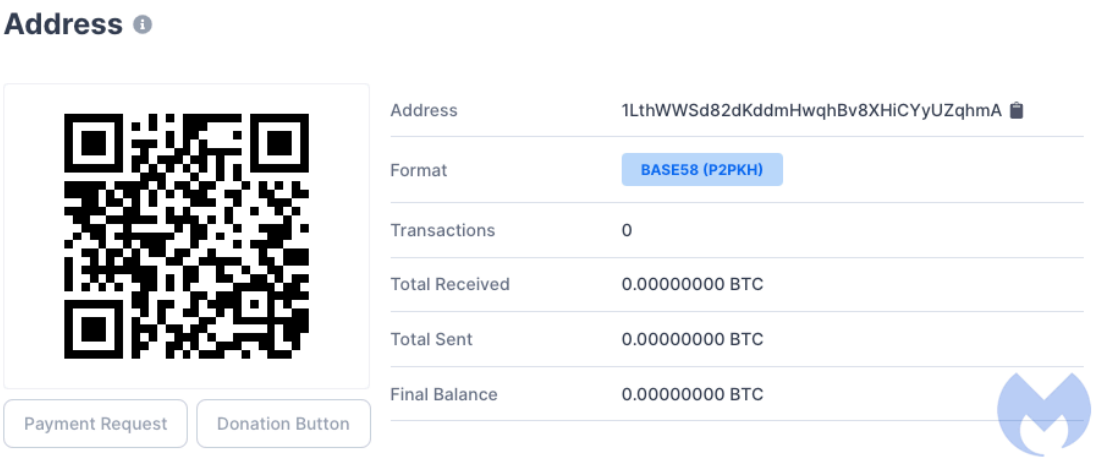
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

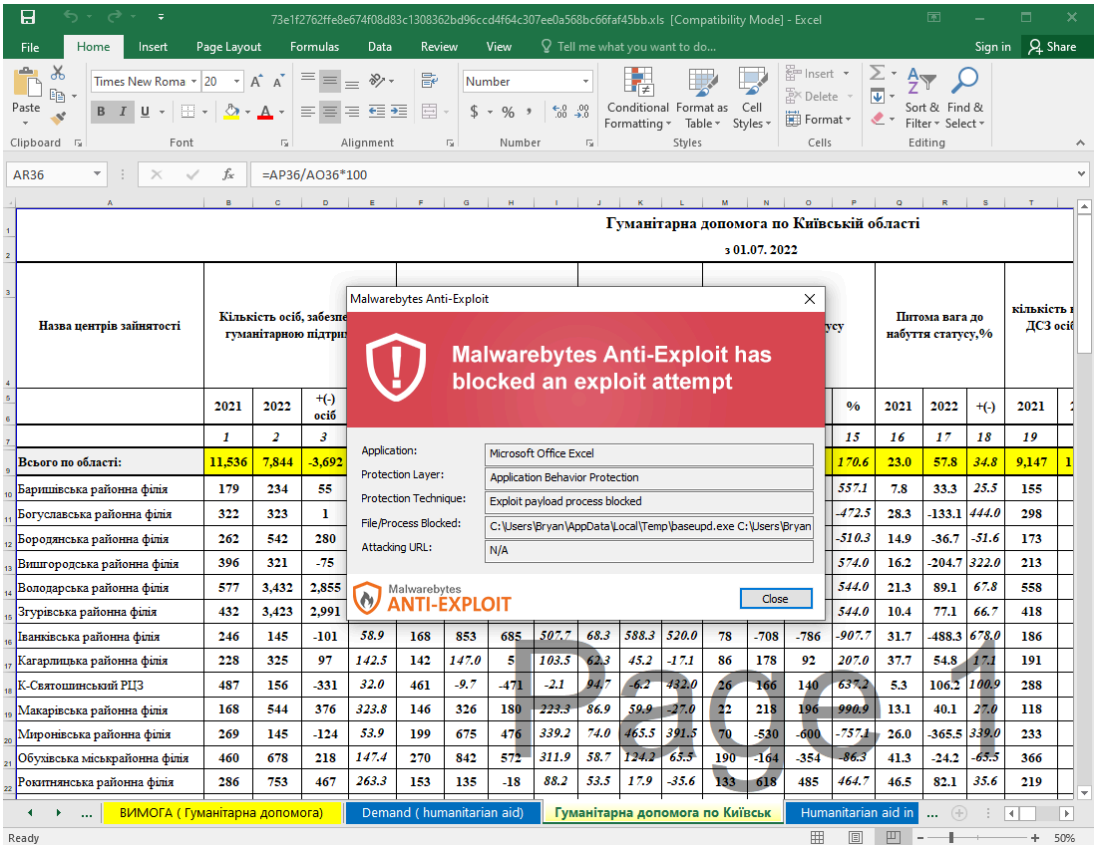
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

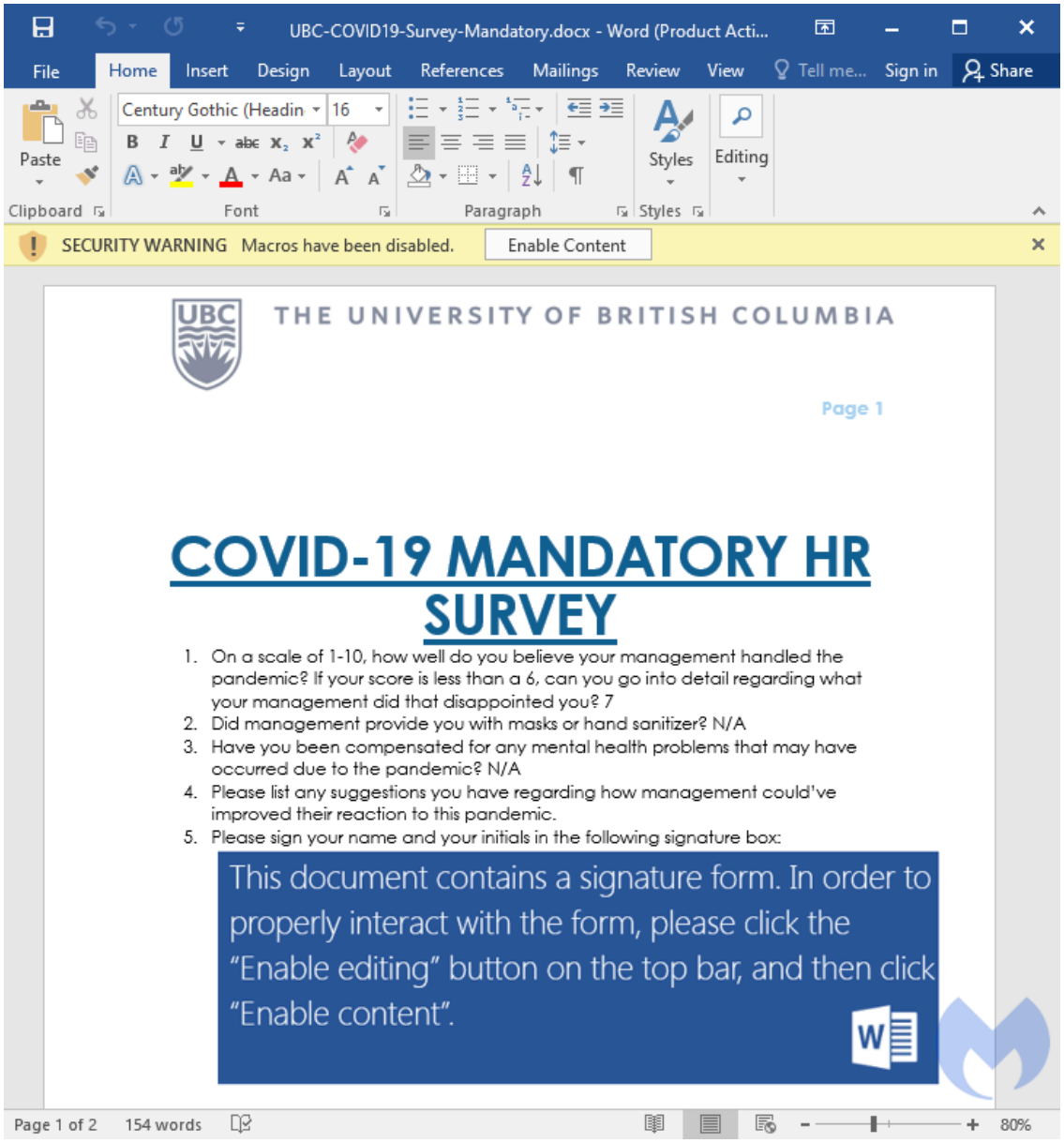
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

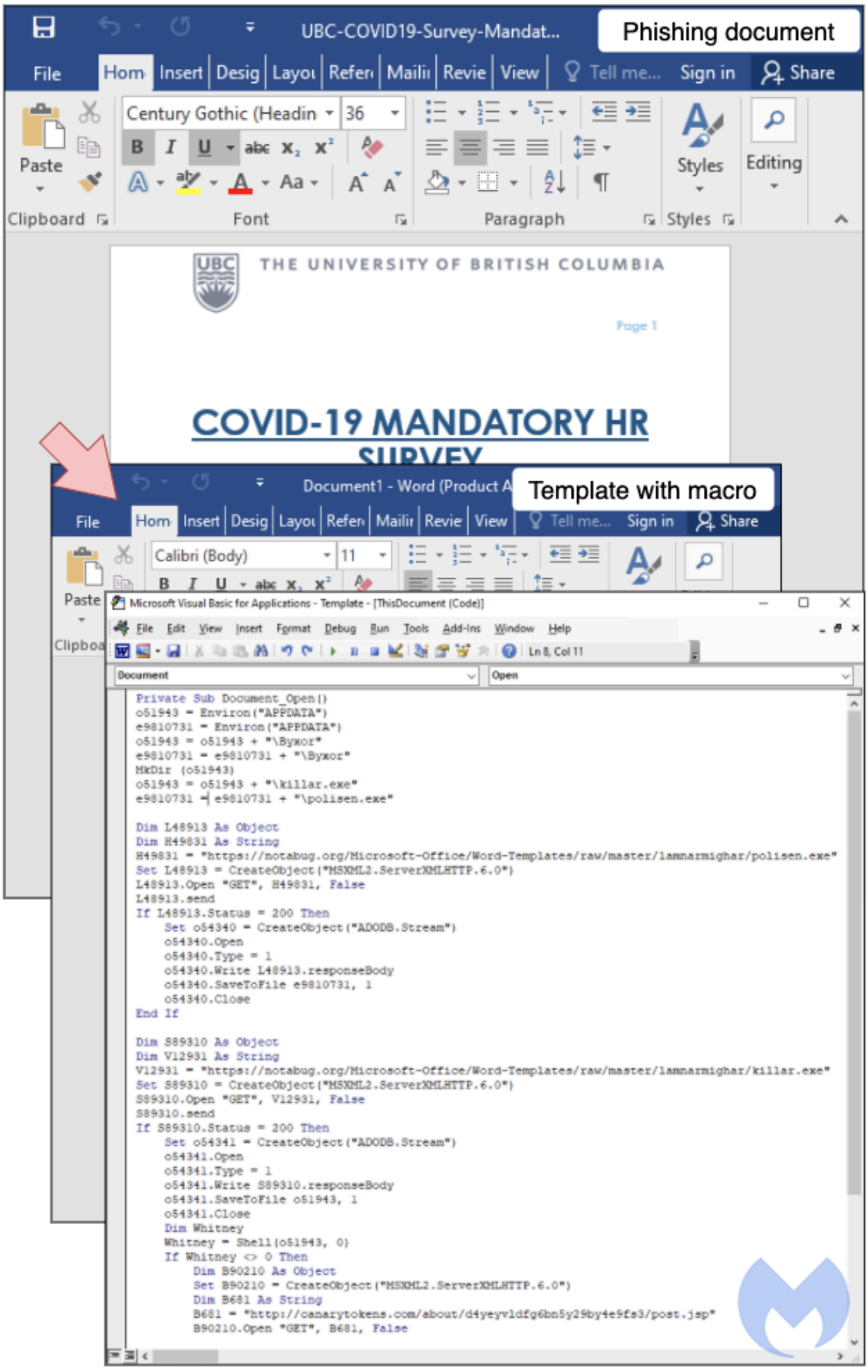
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

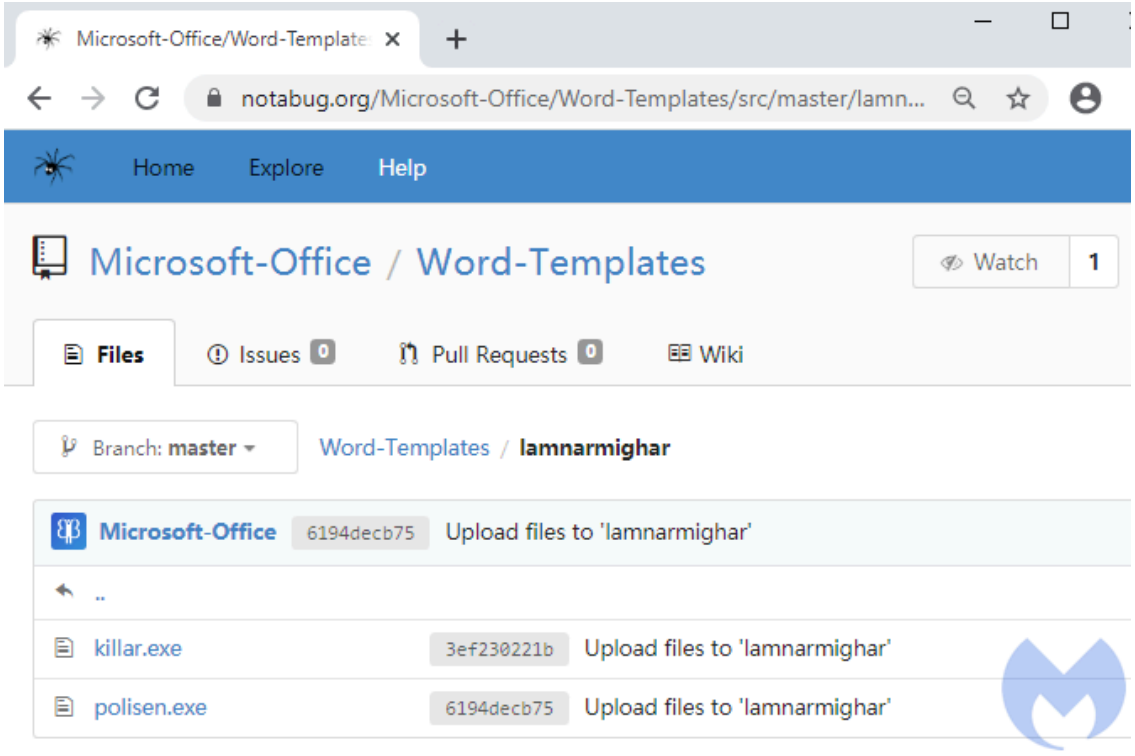
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



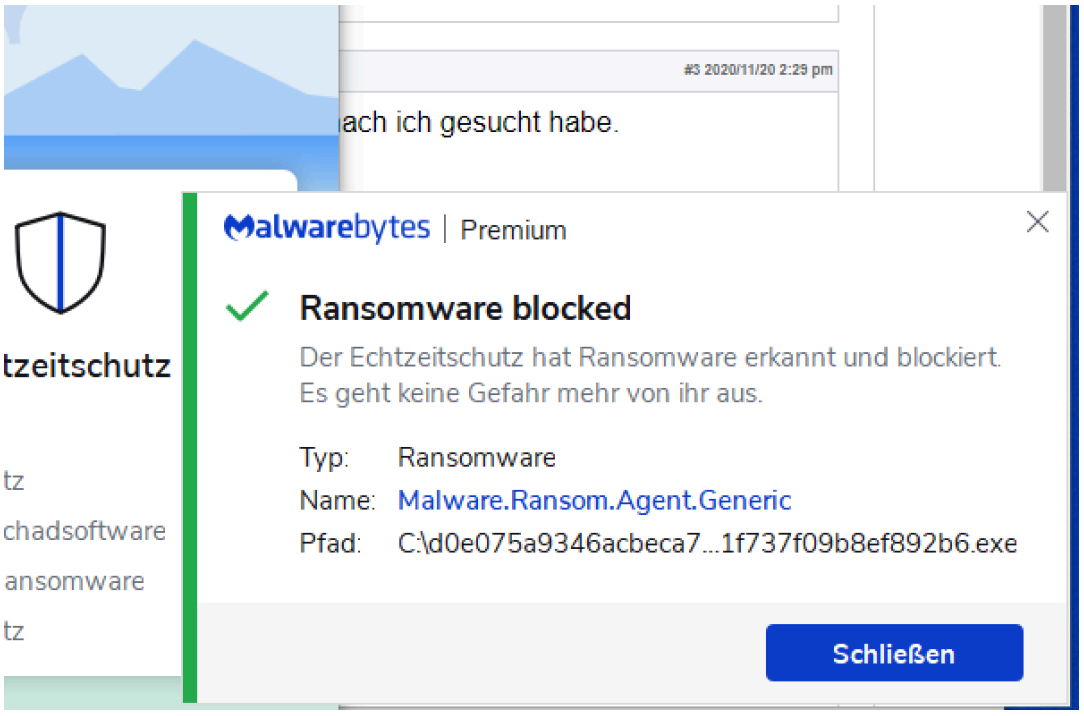
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

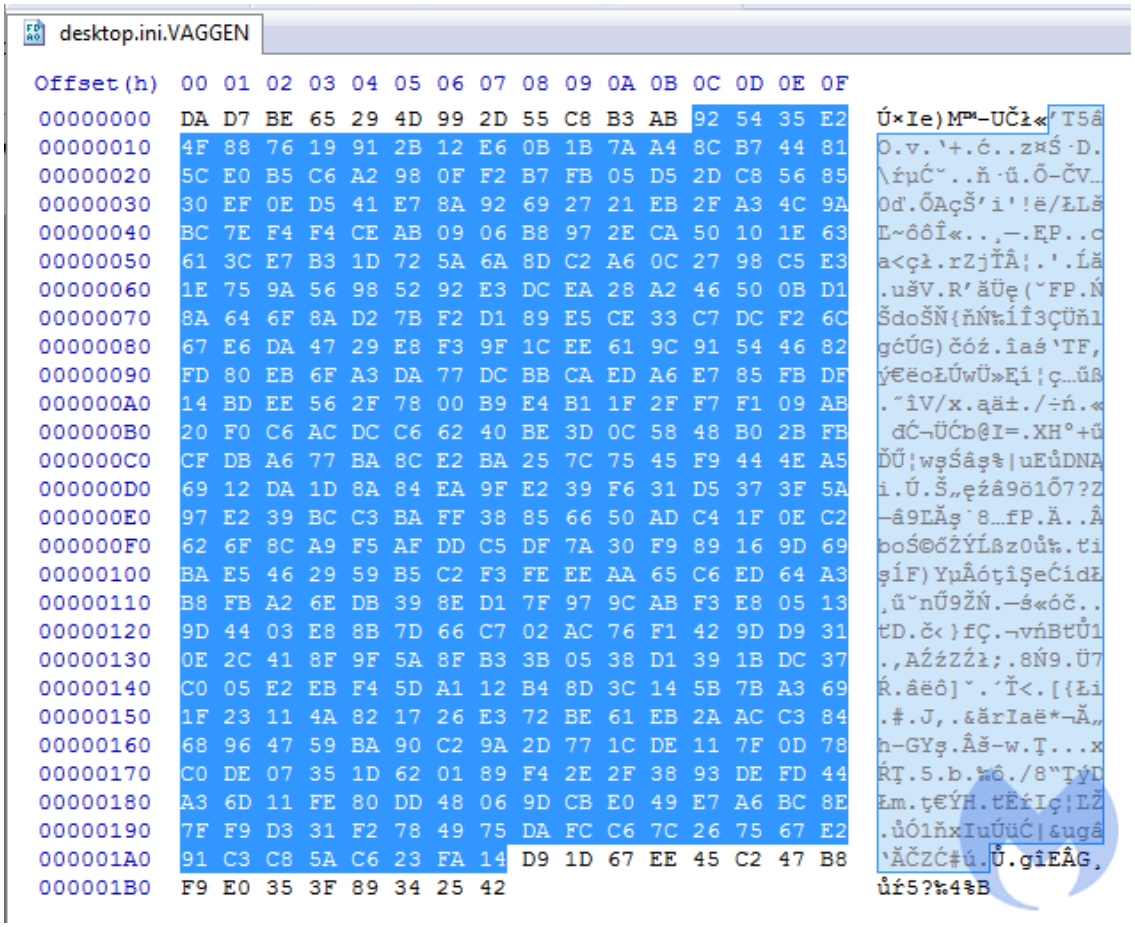
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

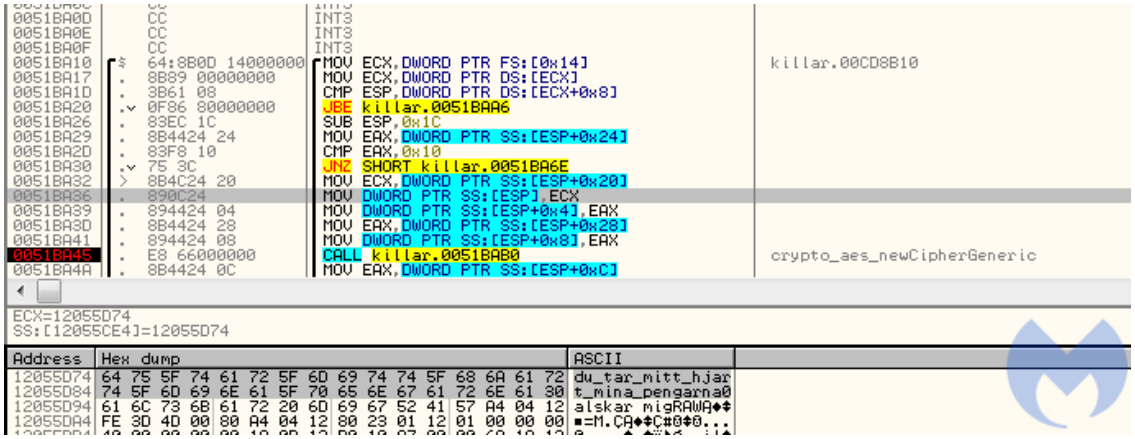


The content of the output file (with .VAGGEN extension) contains:

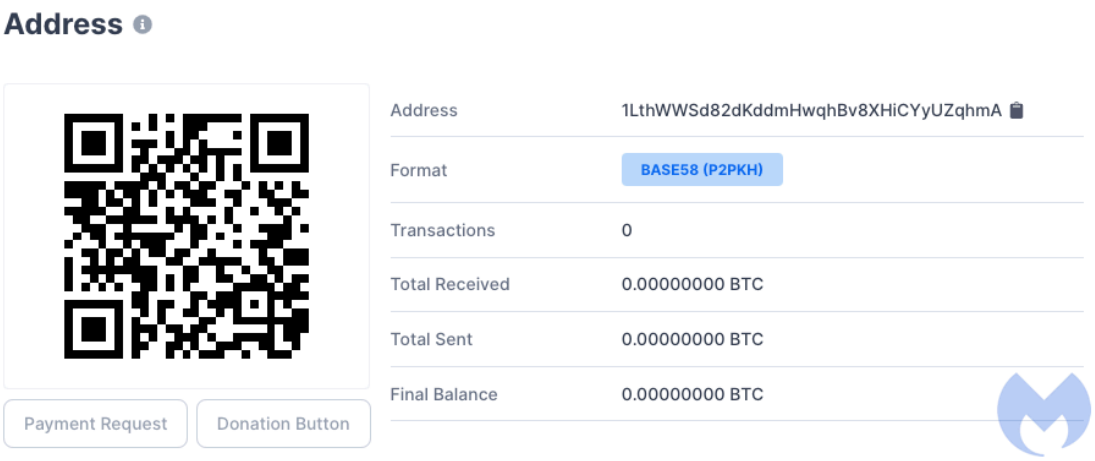
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

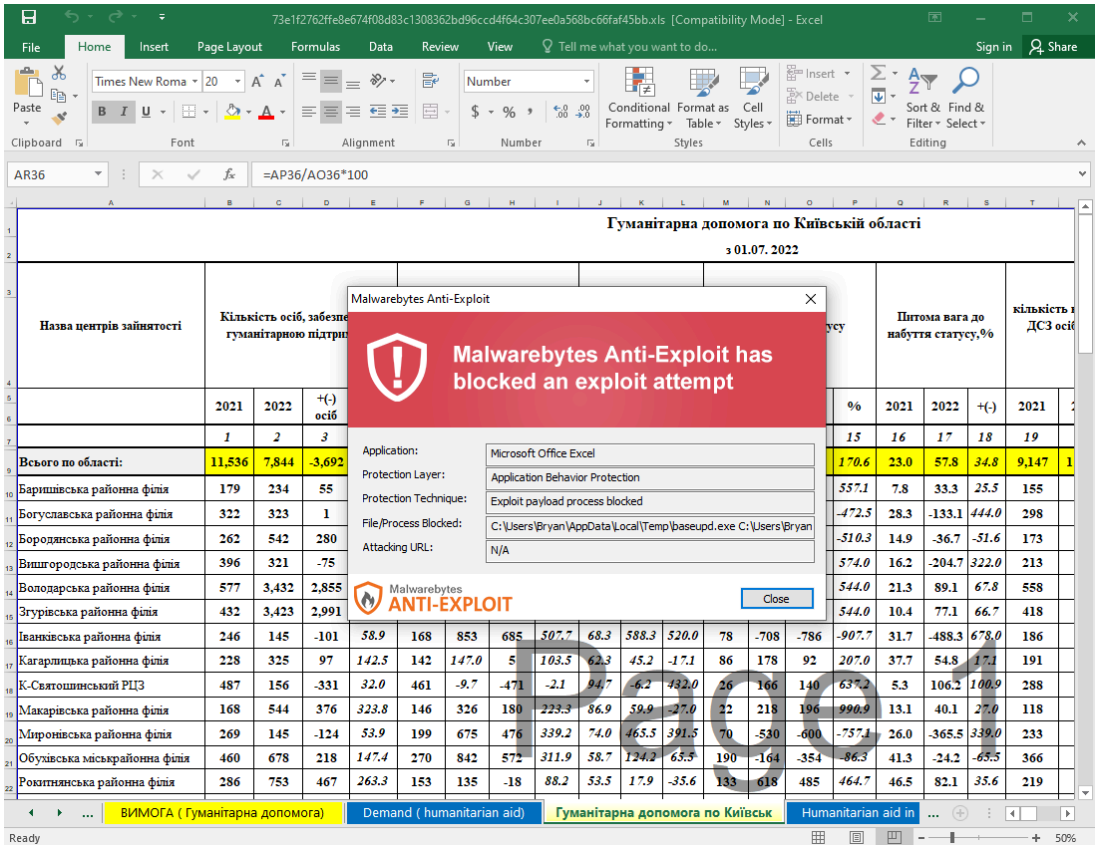
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

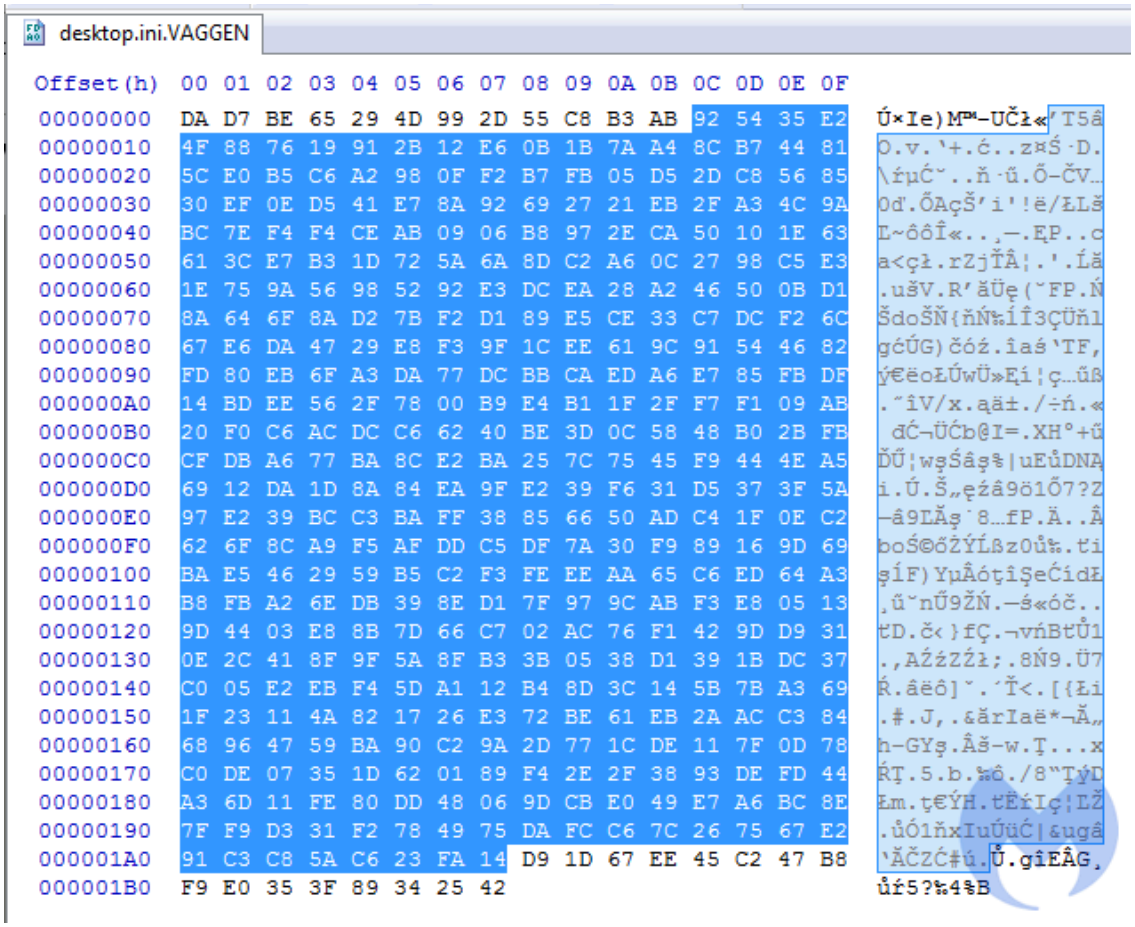
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

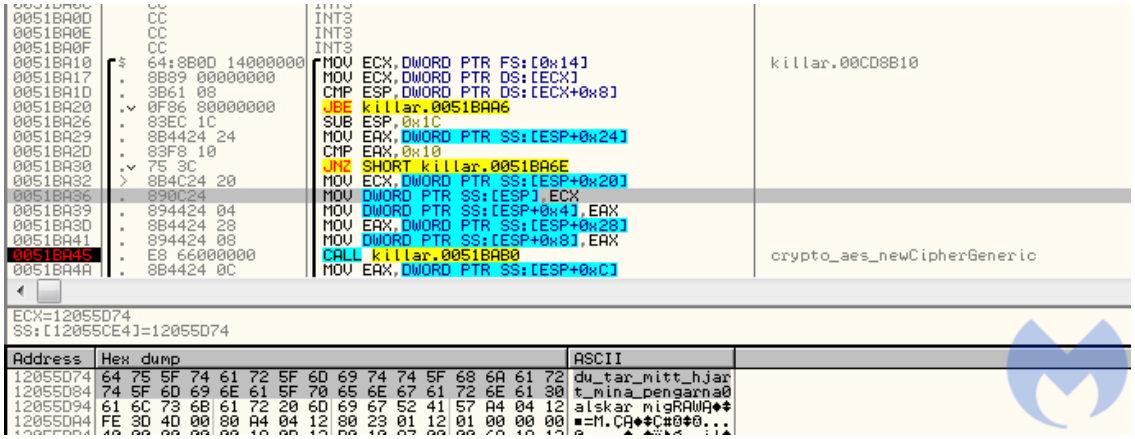


The content of the output file (with .VAGGEN extension) contains:

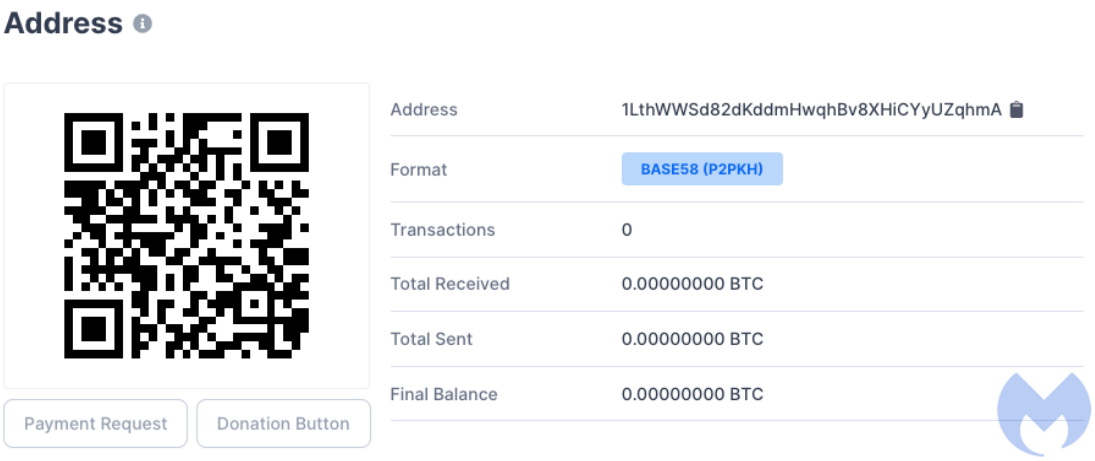
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

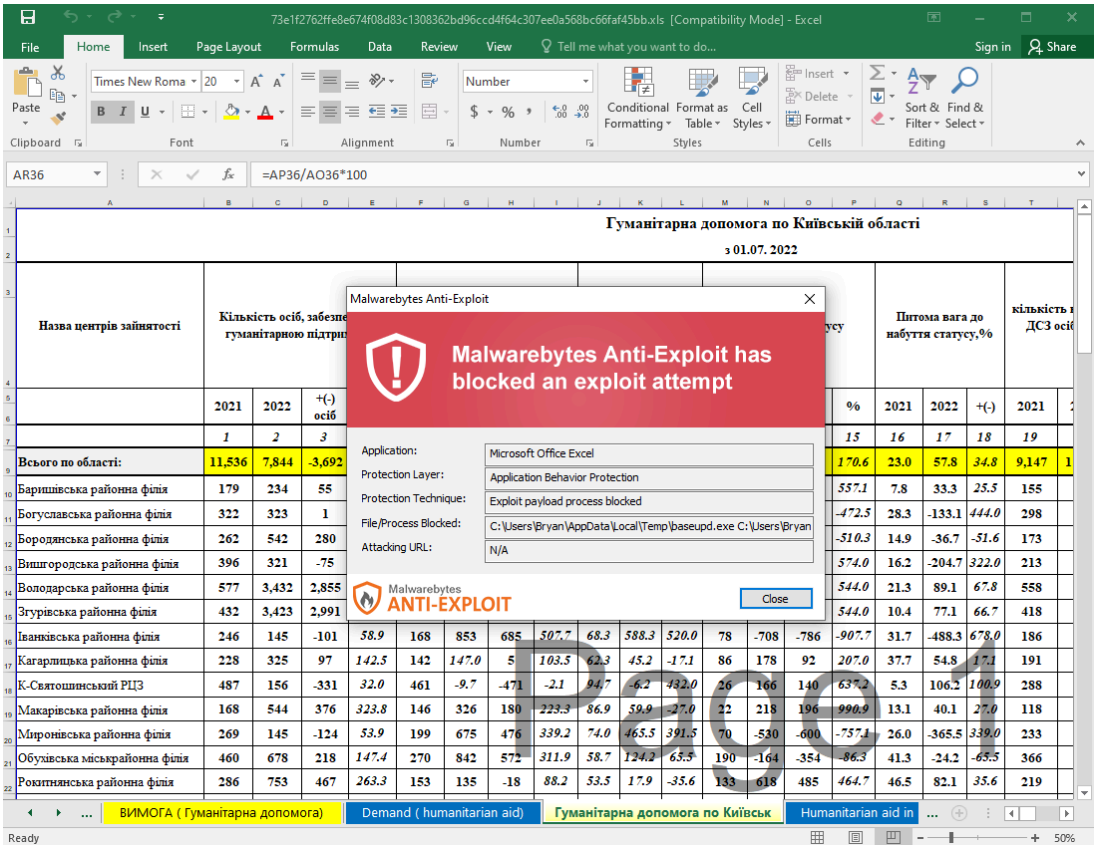
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

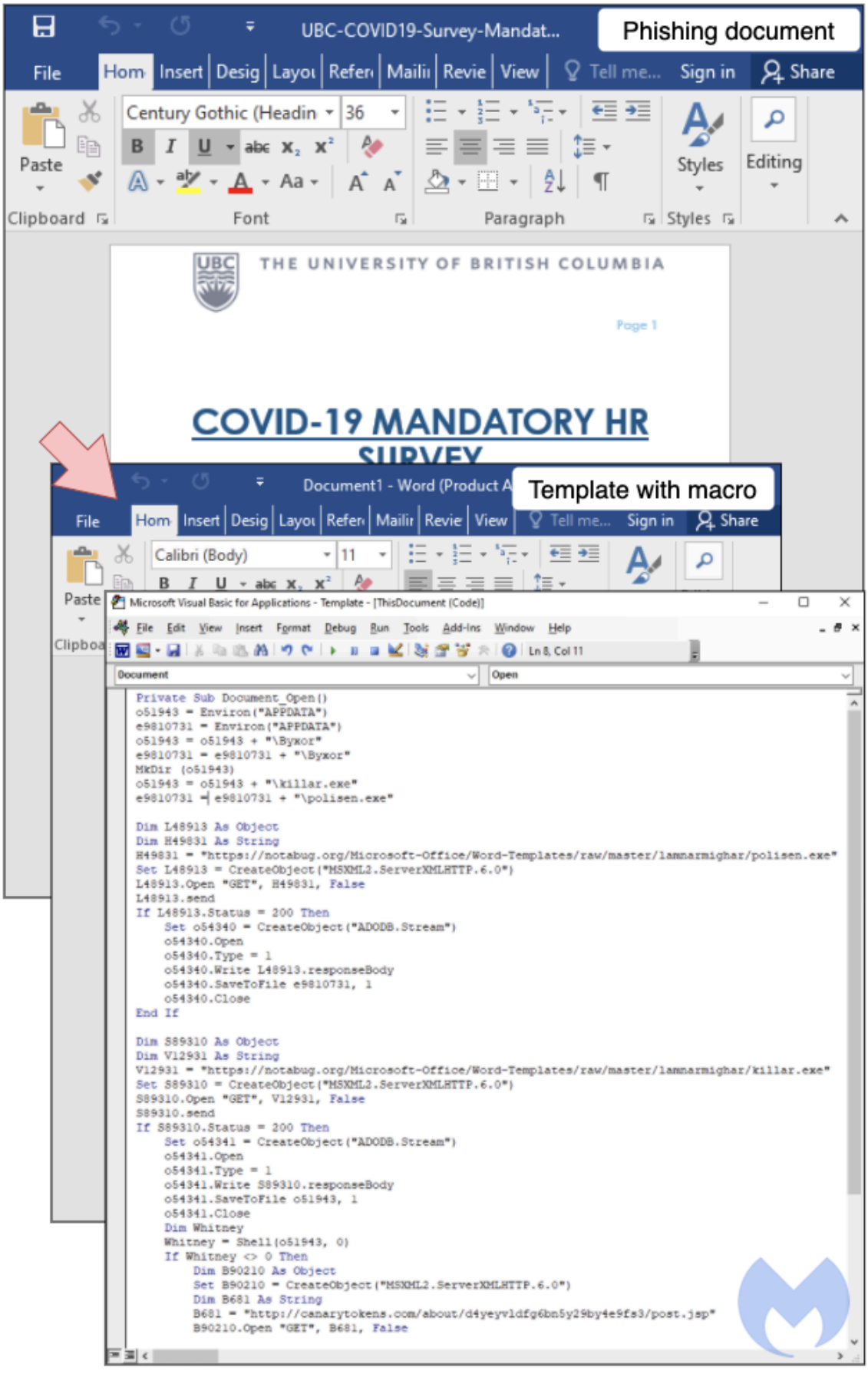
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

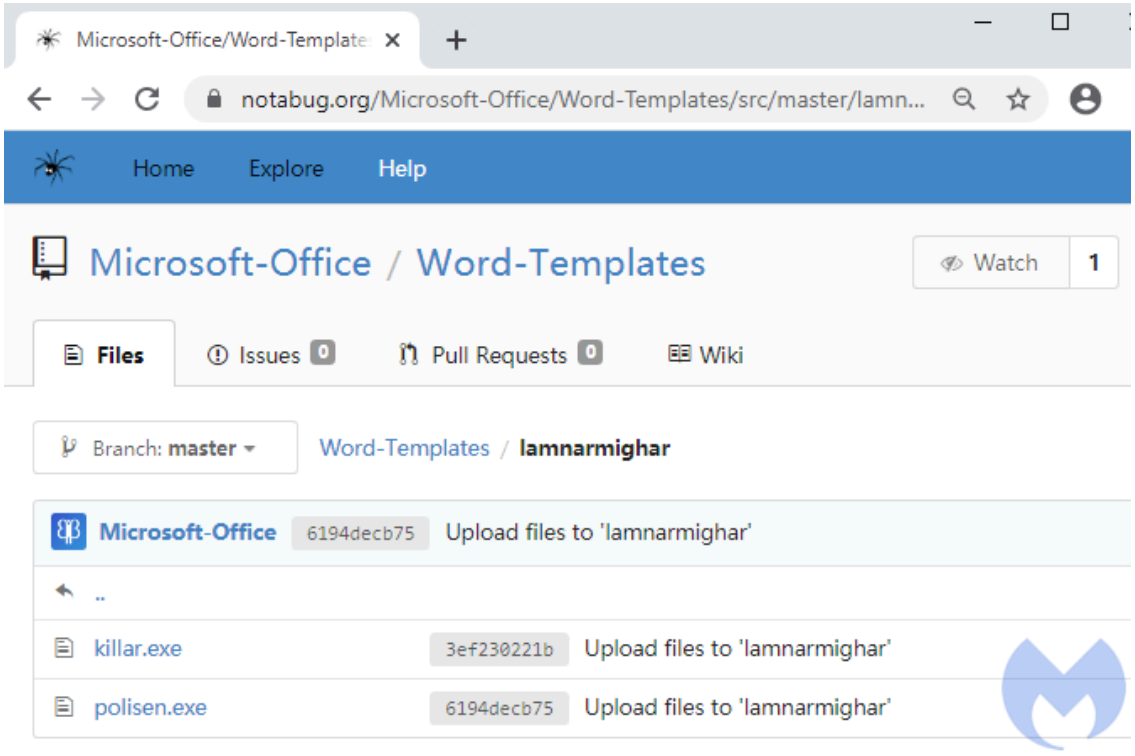
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



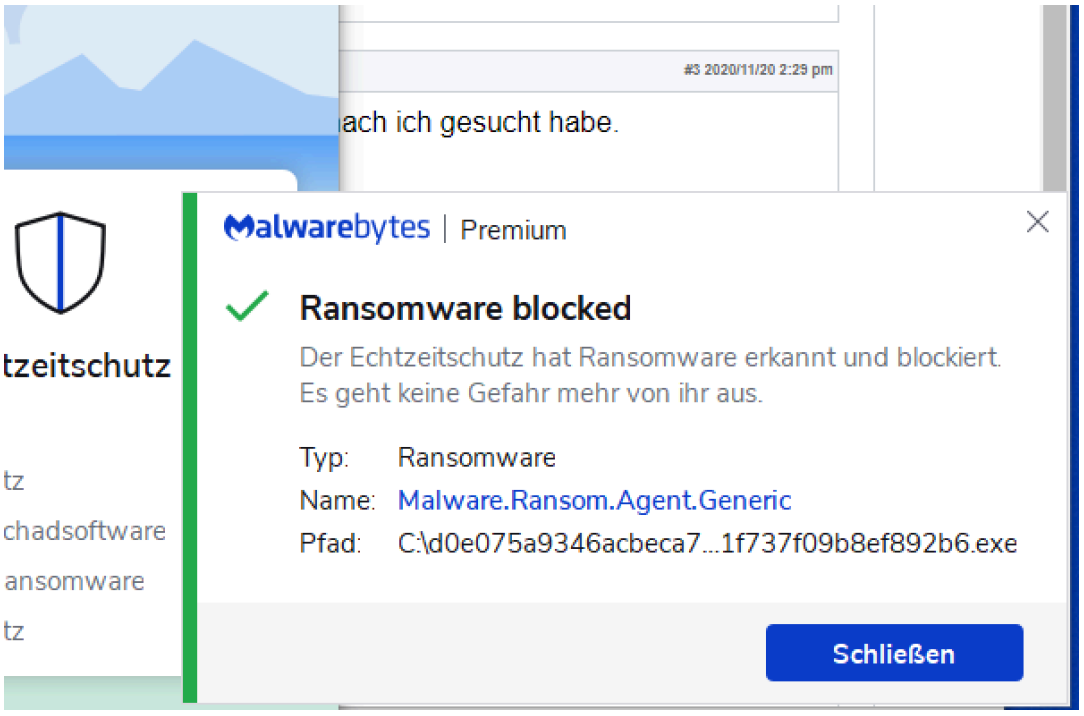
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

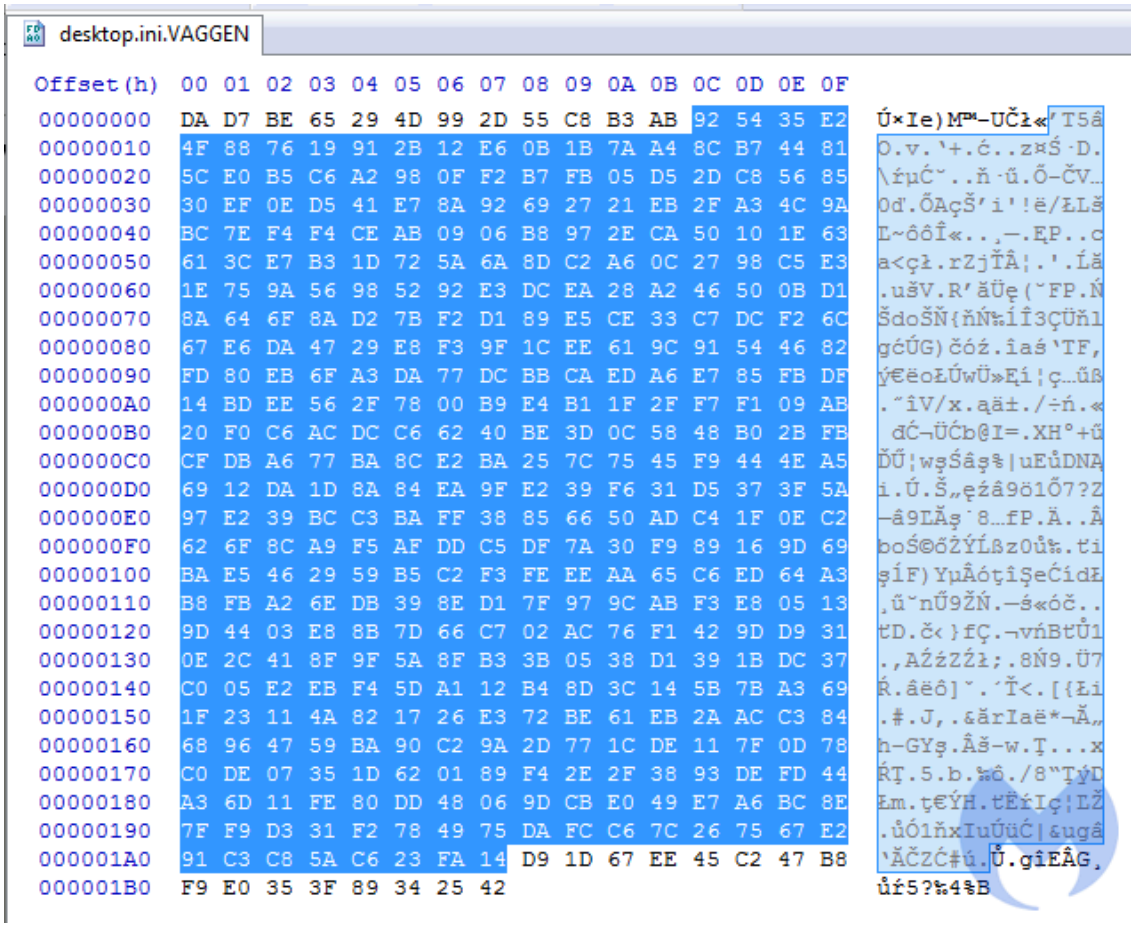
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

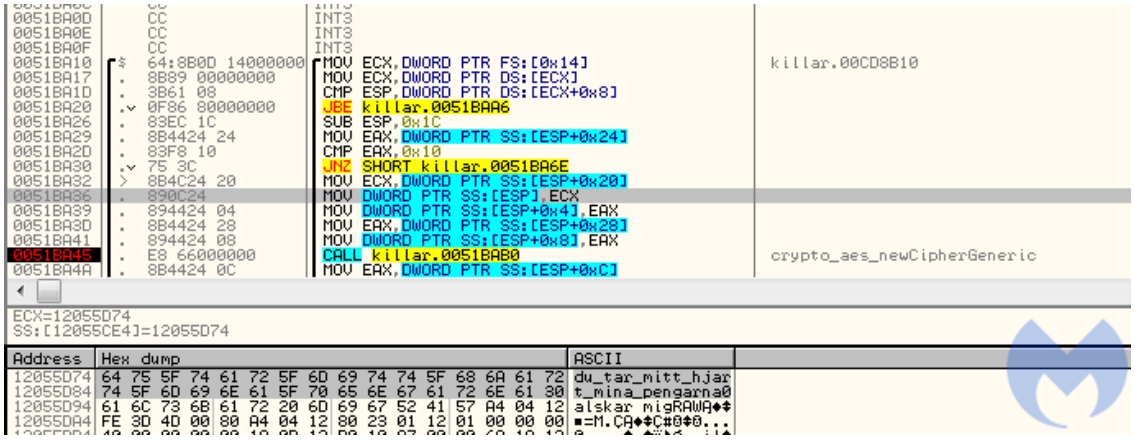


The content of the output file (with .VAGGEN extension) contains:

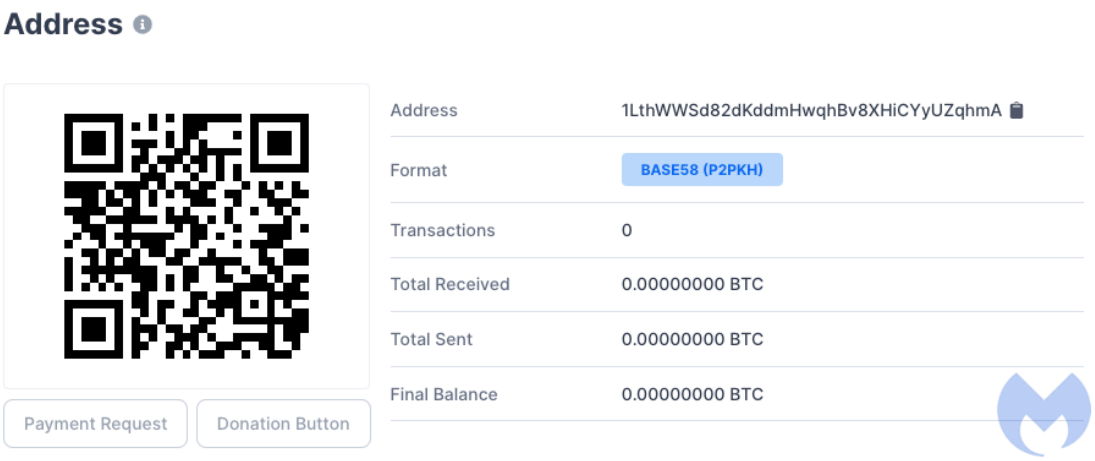
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

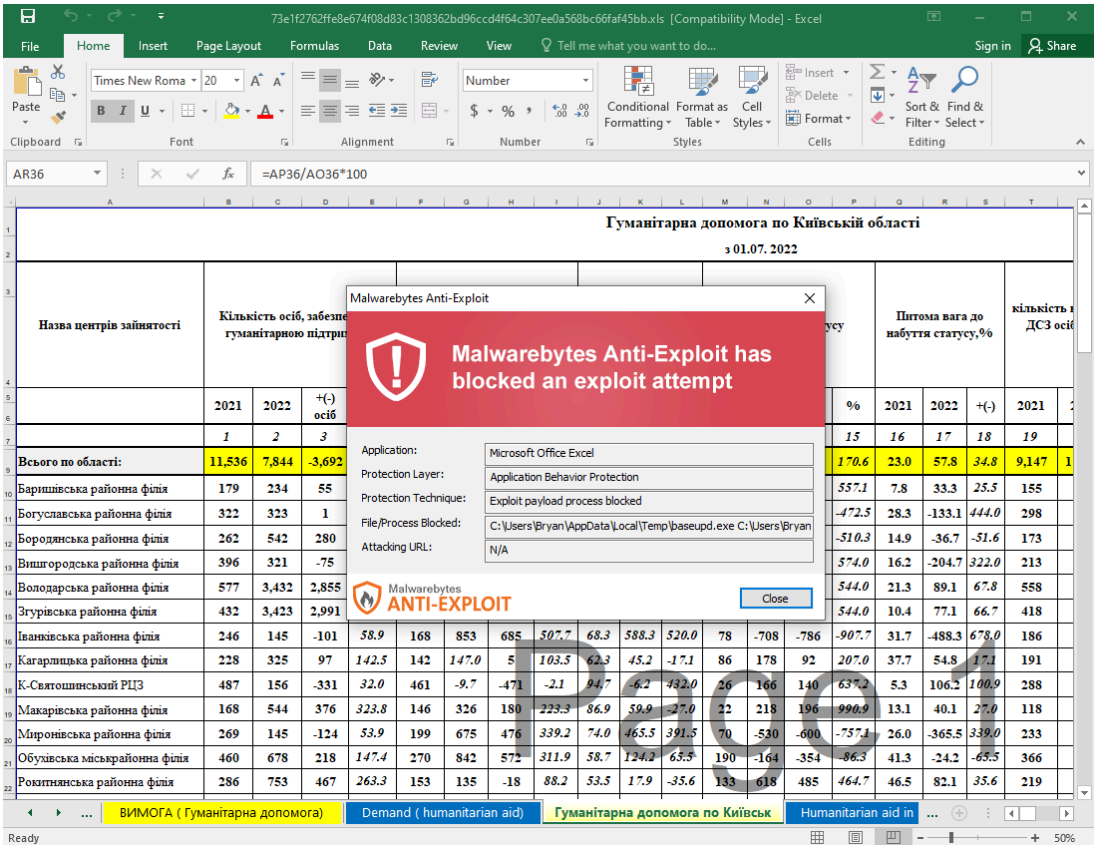
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

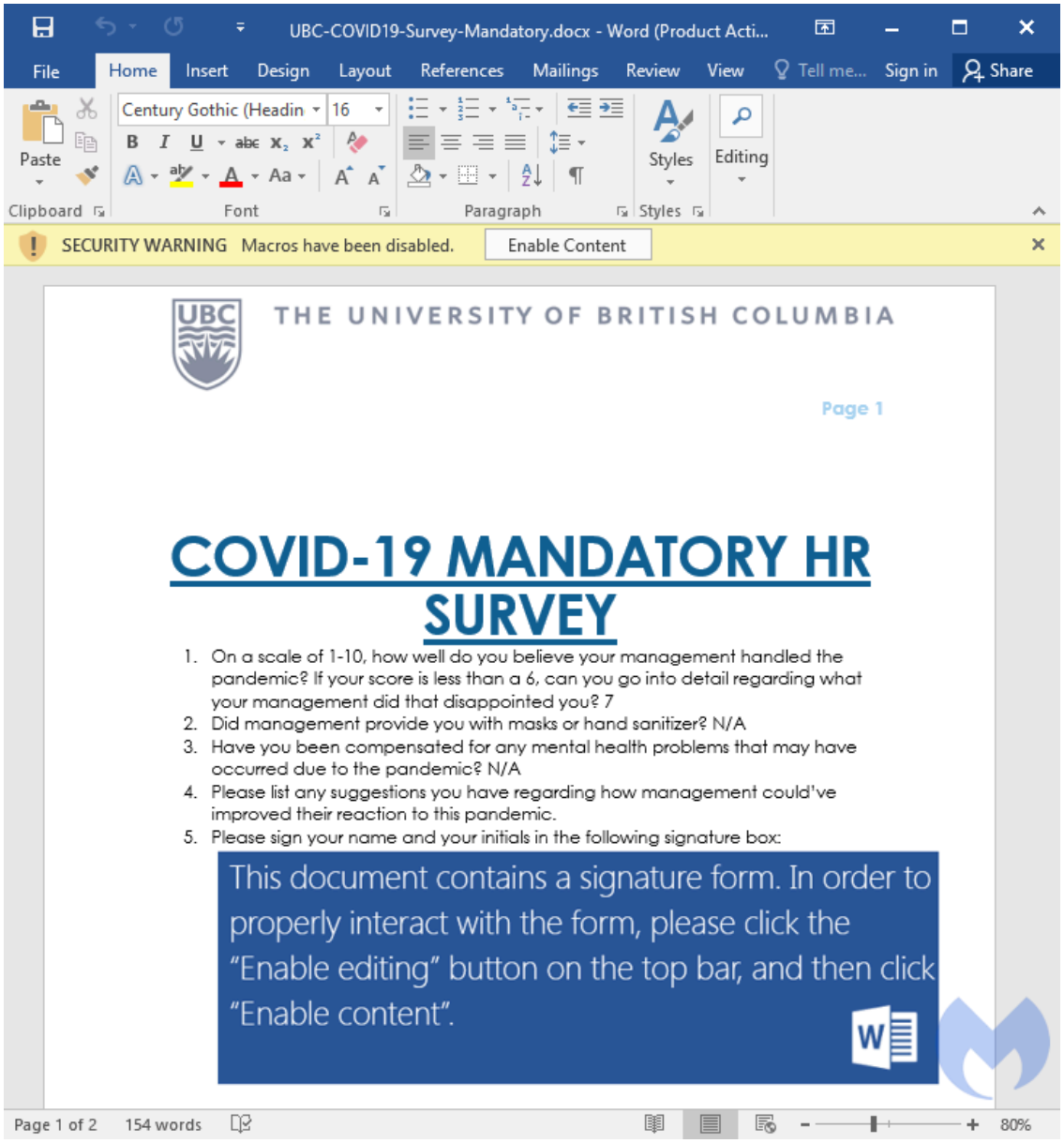
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

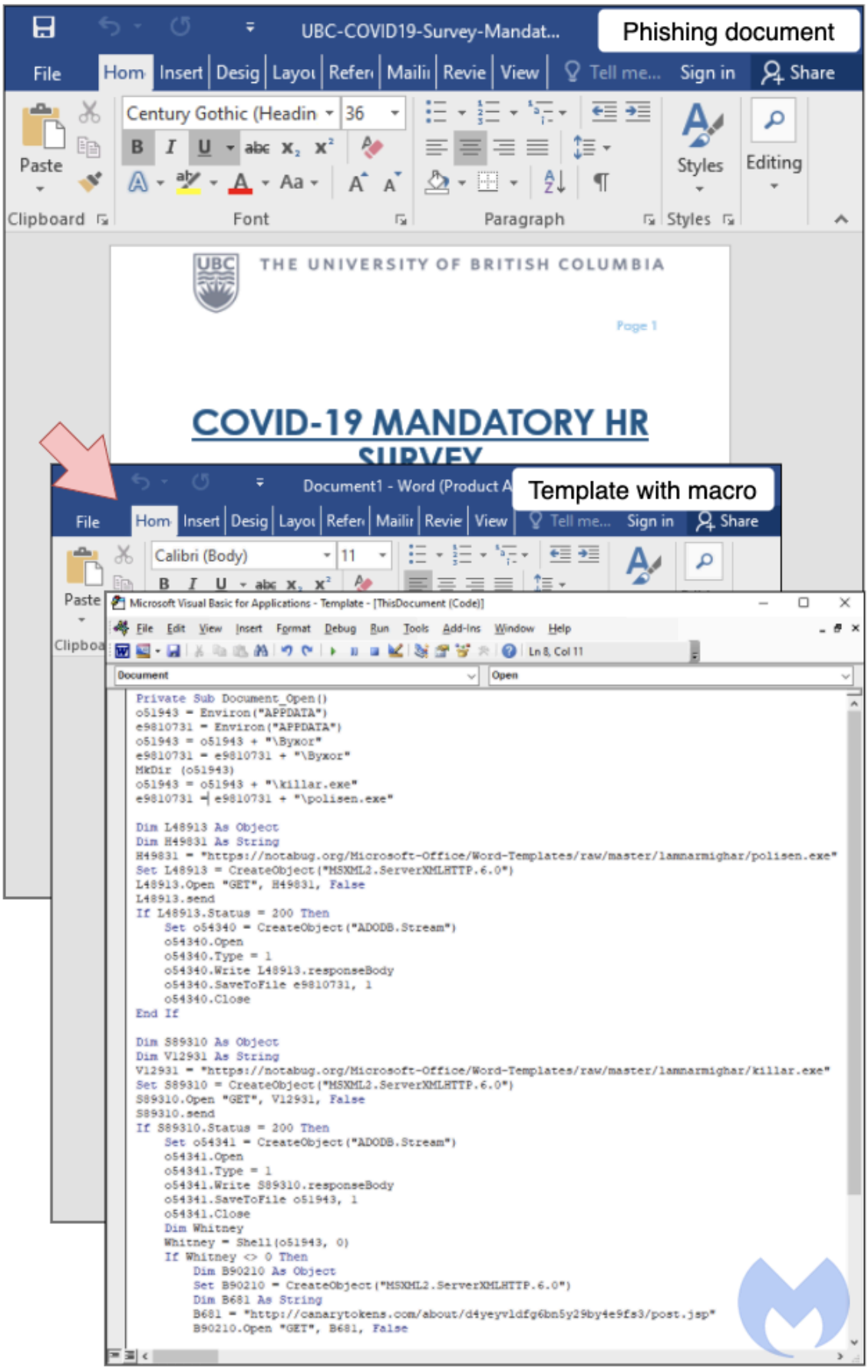
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

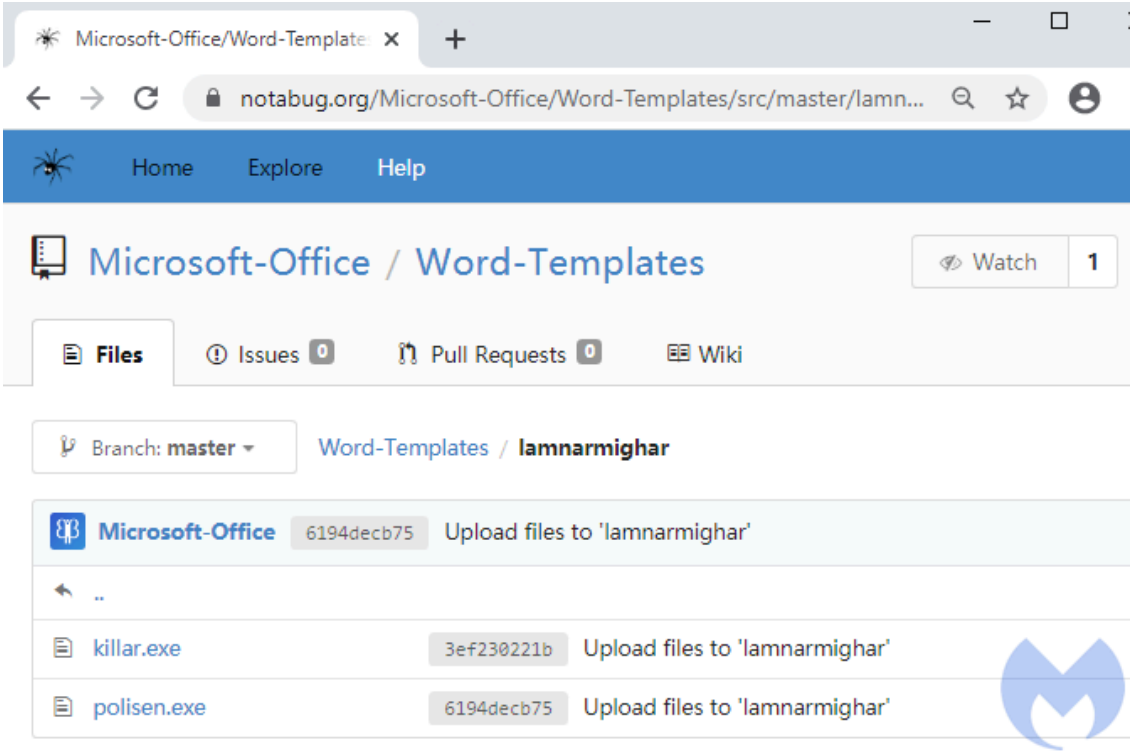
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



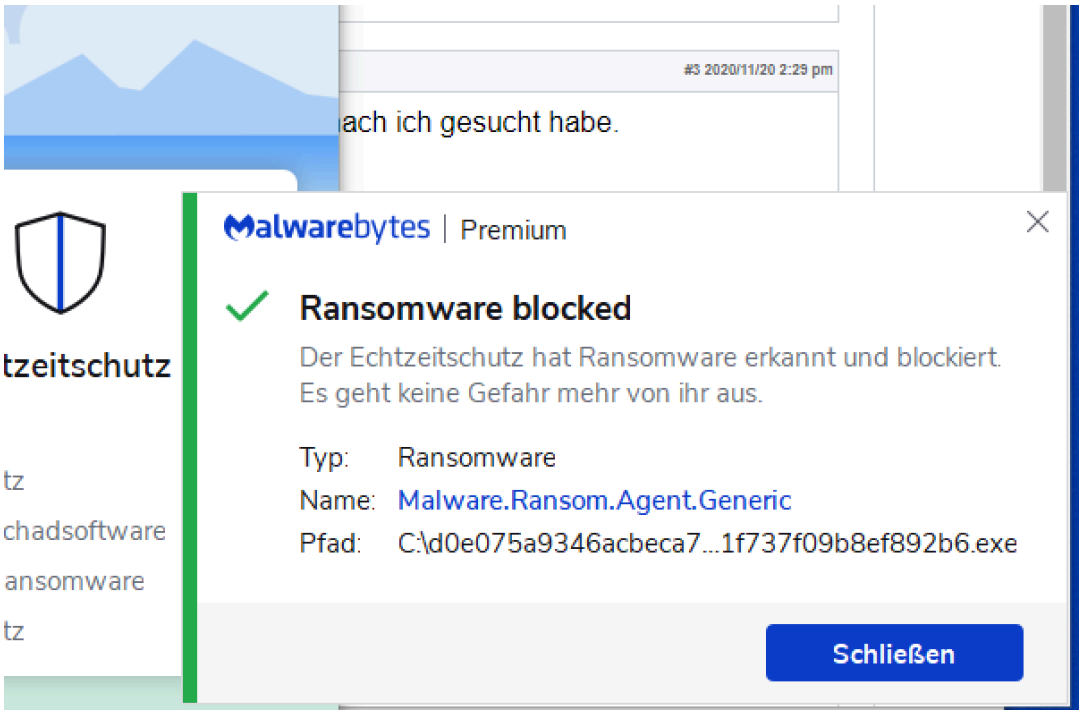
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

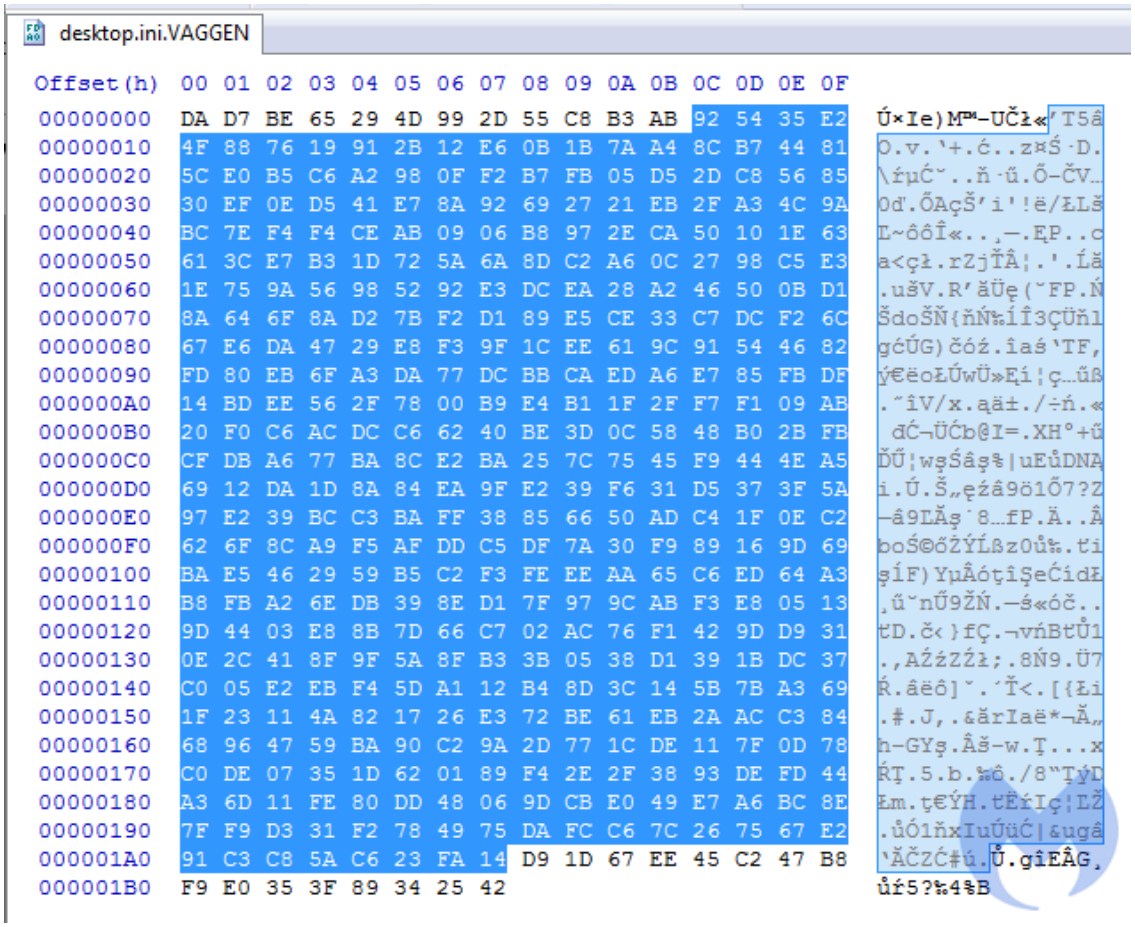
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



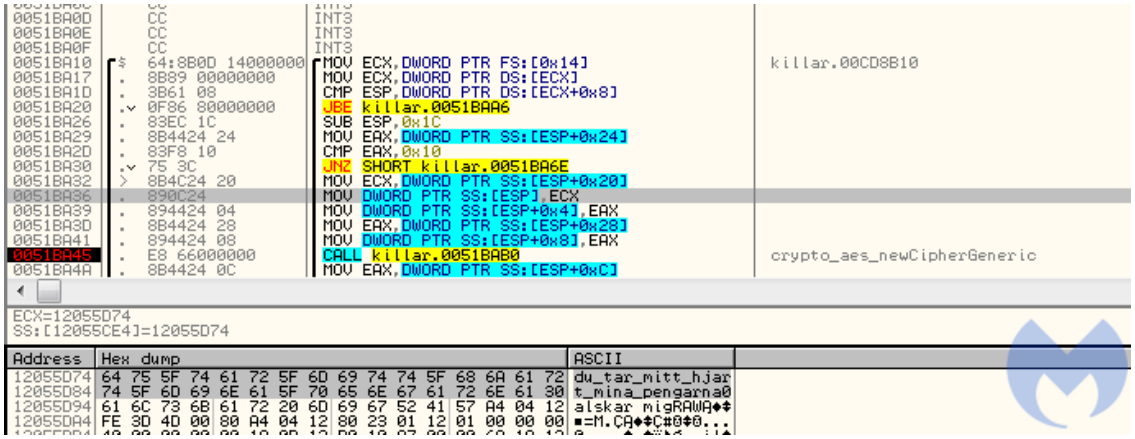
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

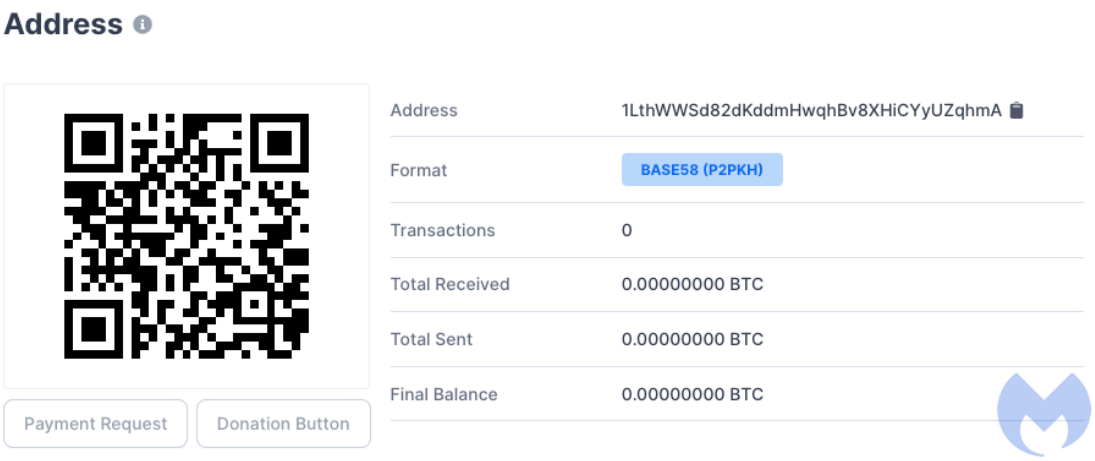
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

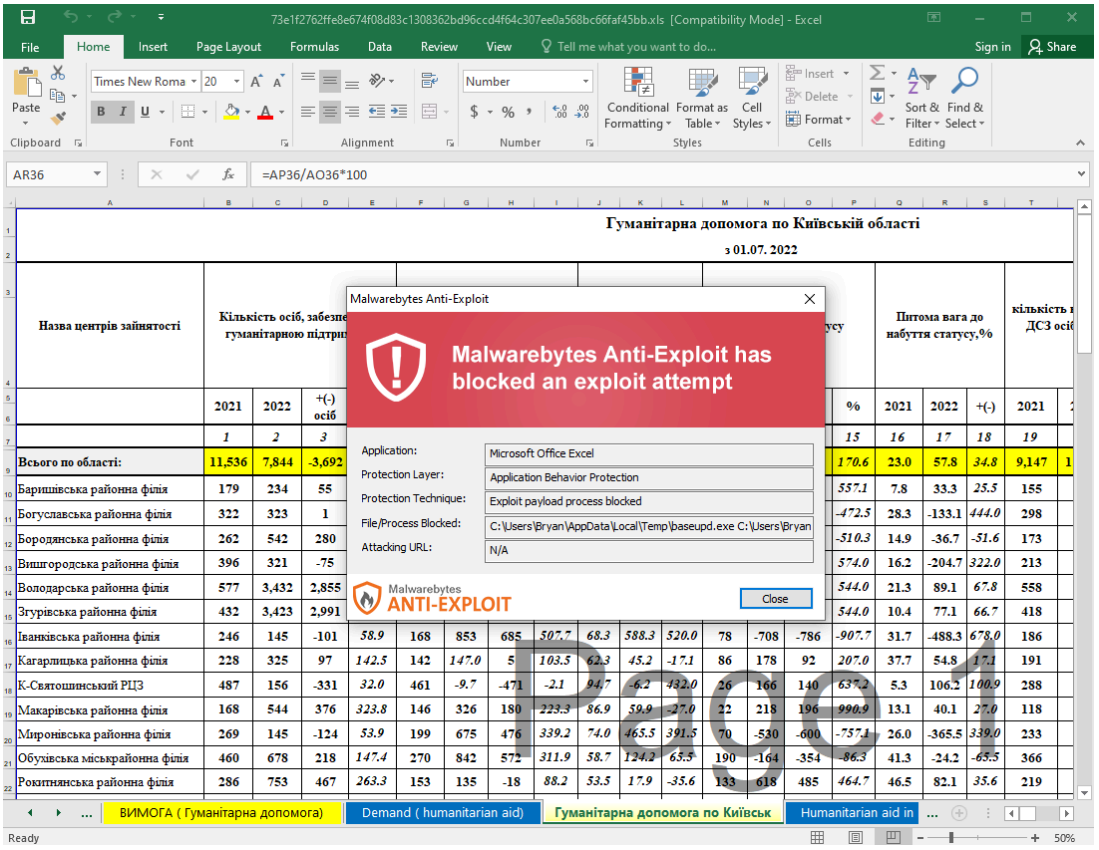
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

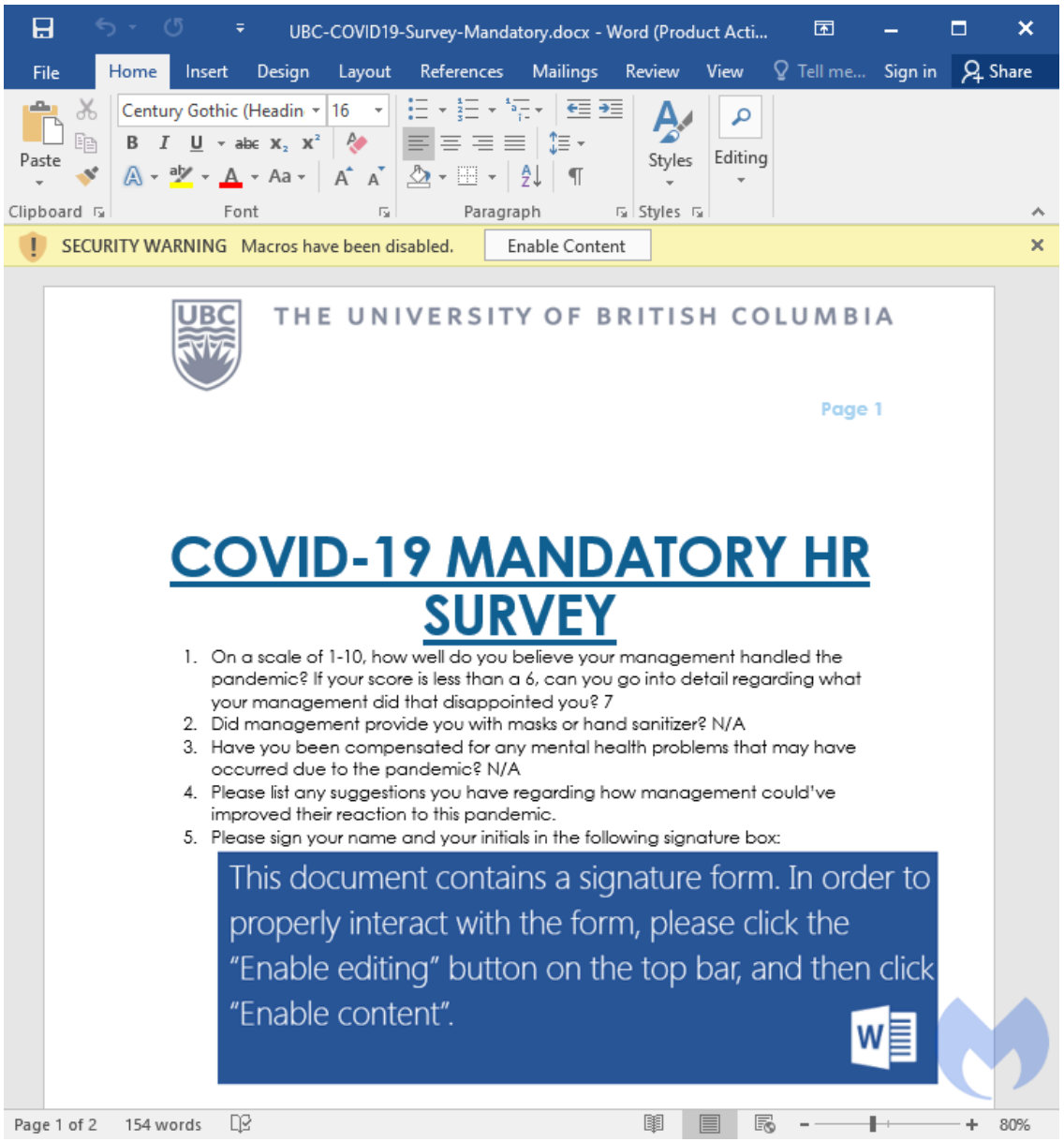
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

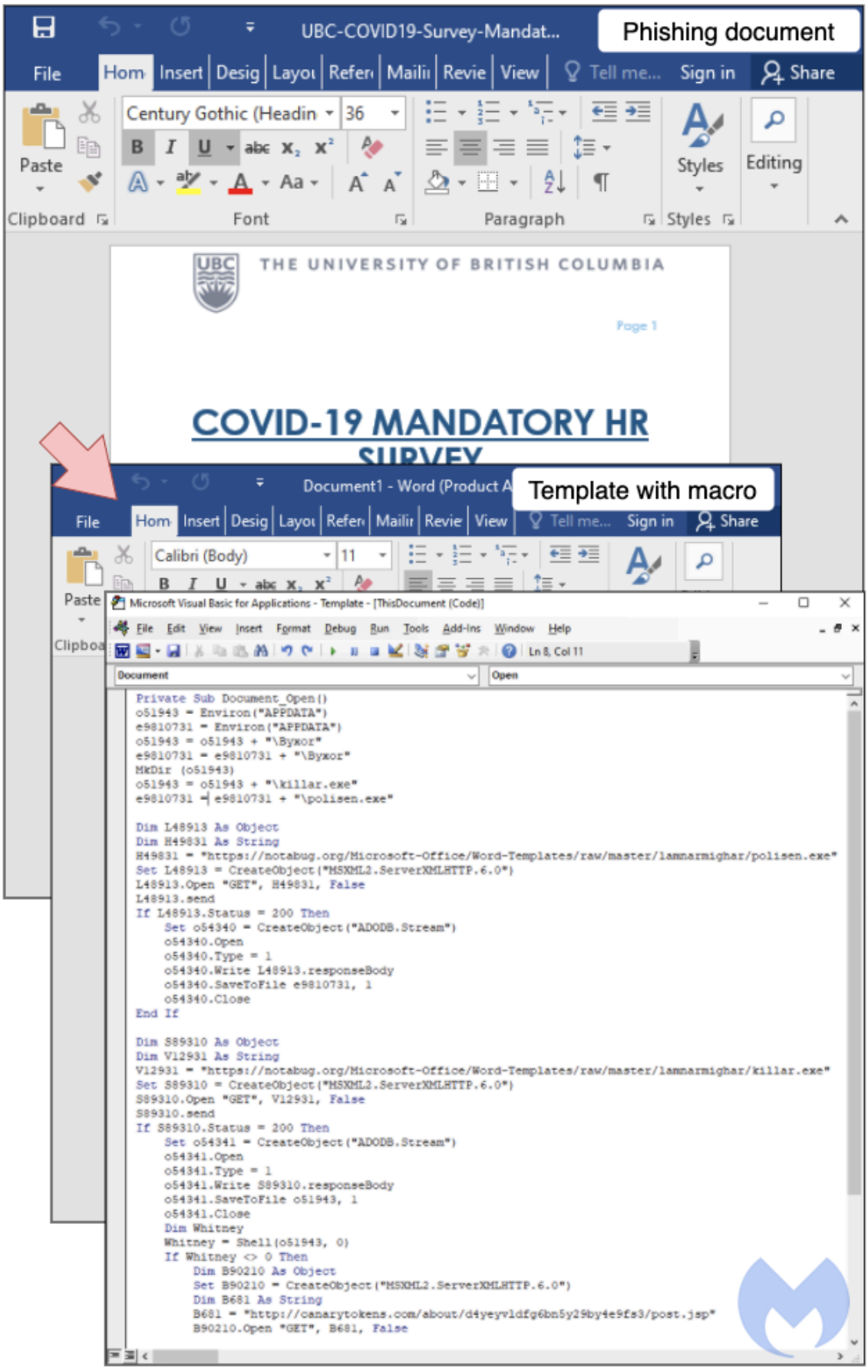
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

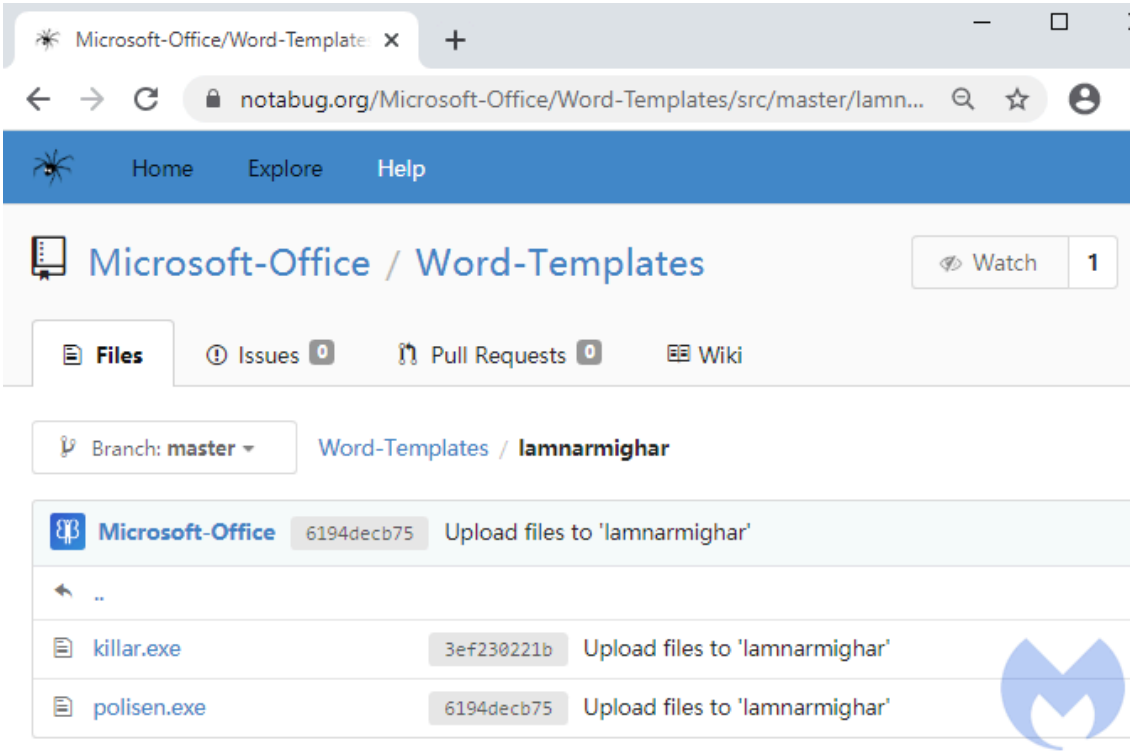
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



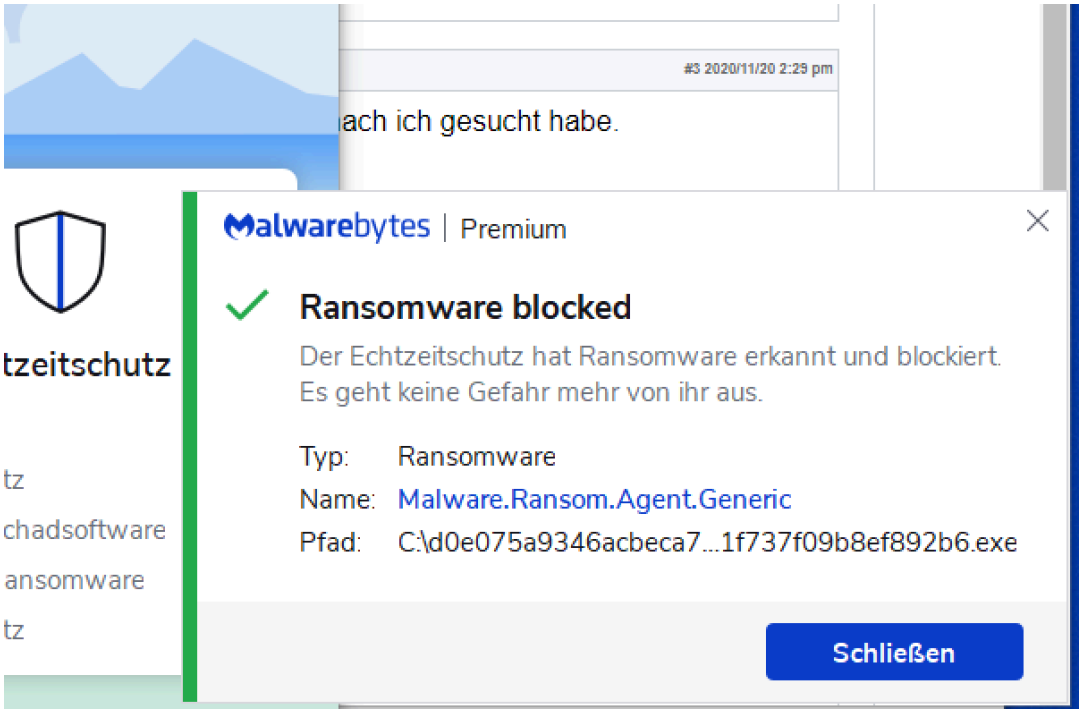
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

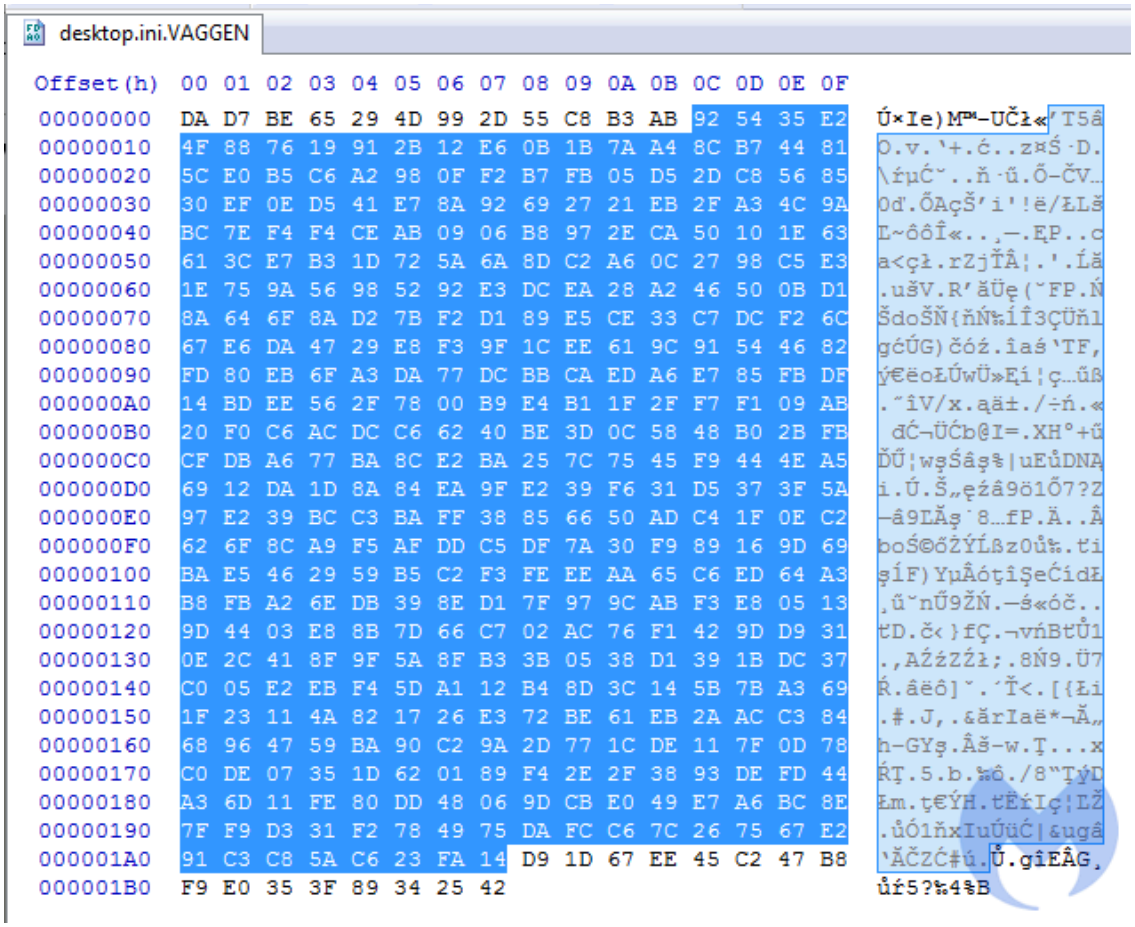
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

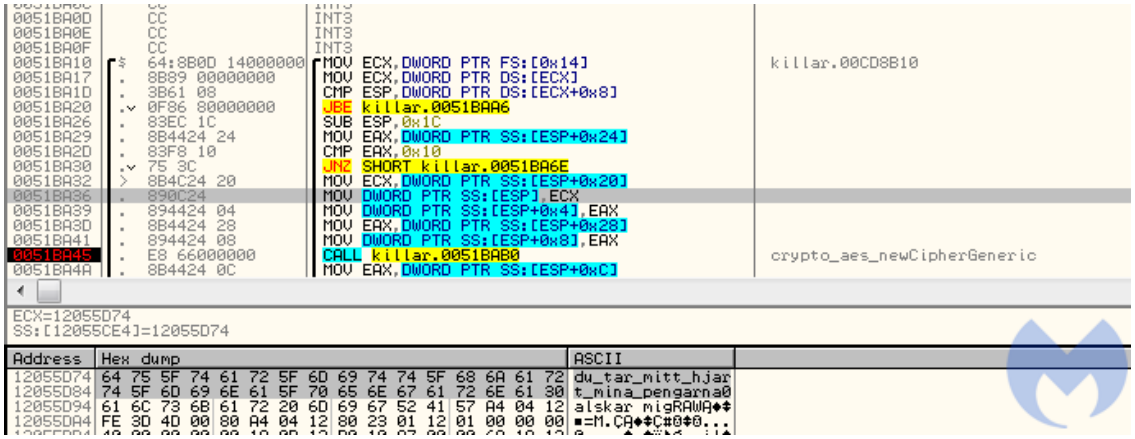


The content of the output file (with .VAGGEN extension) contains:

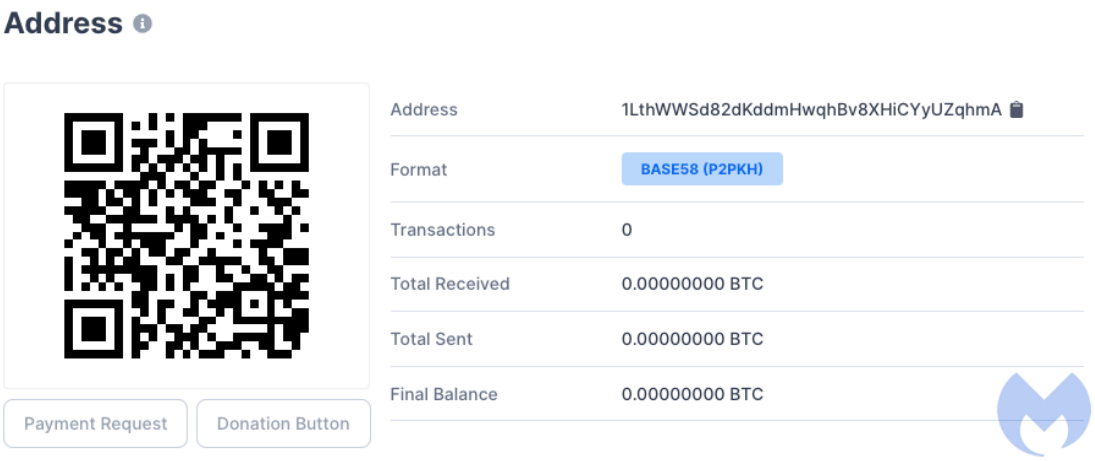
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

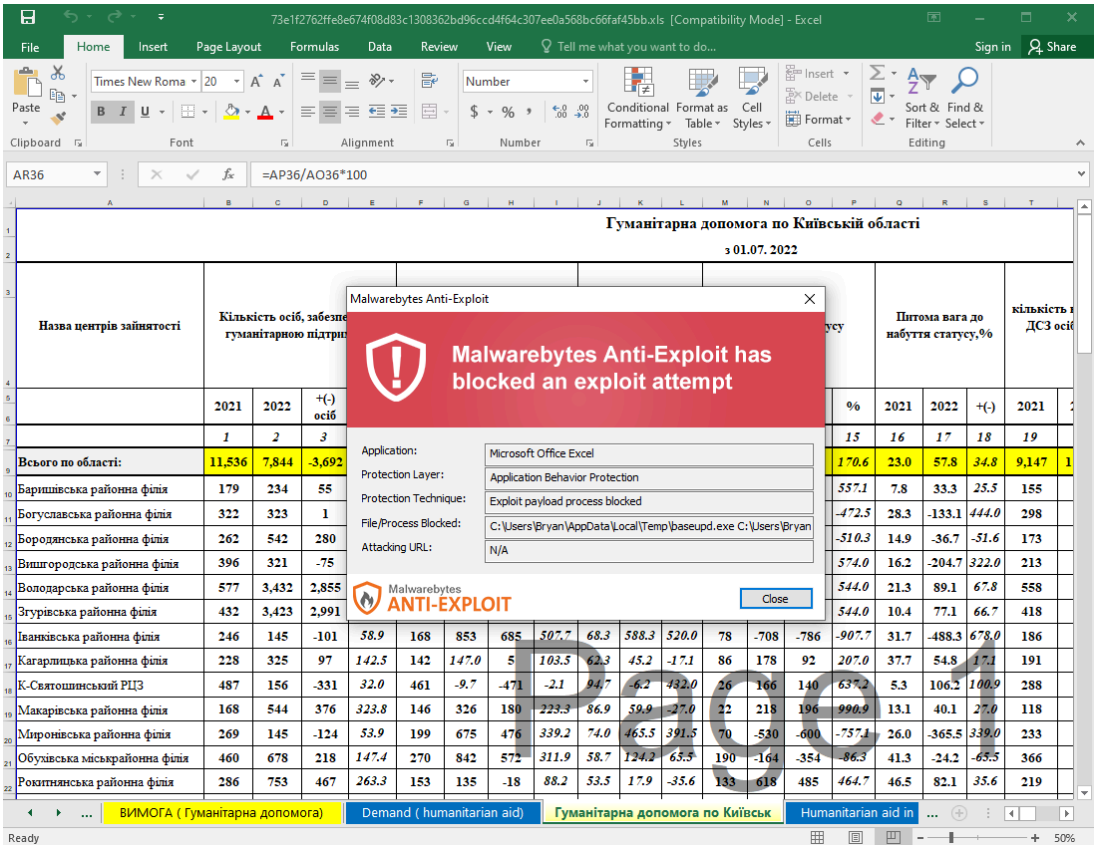
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

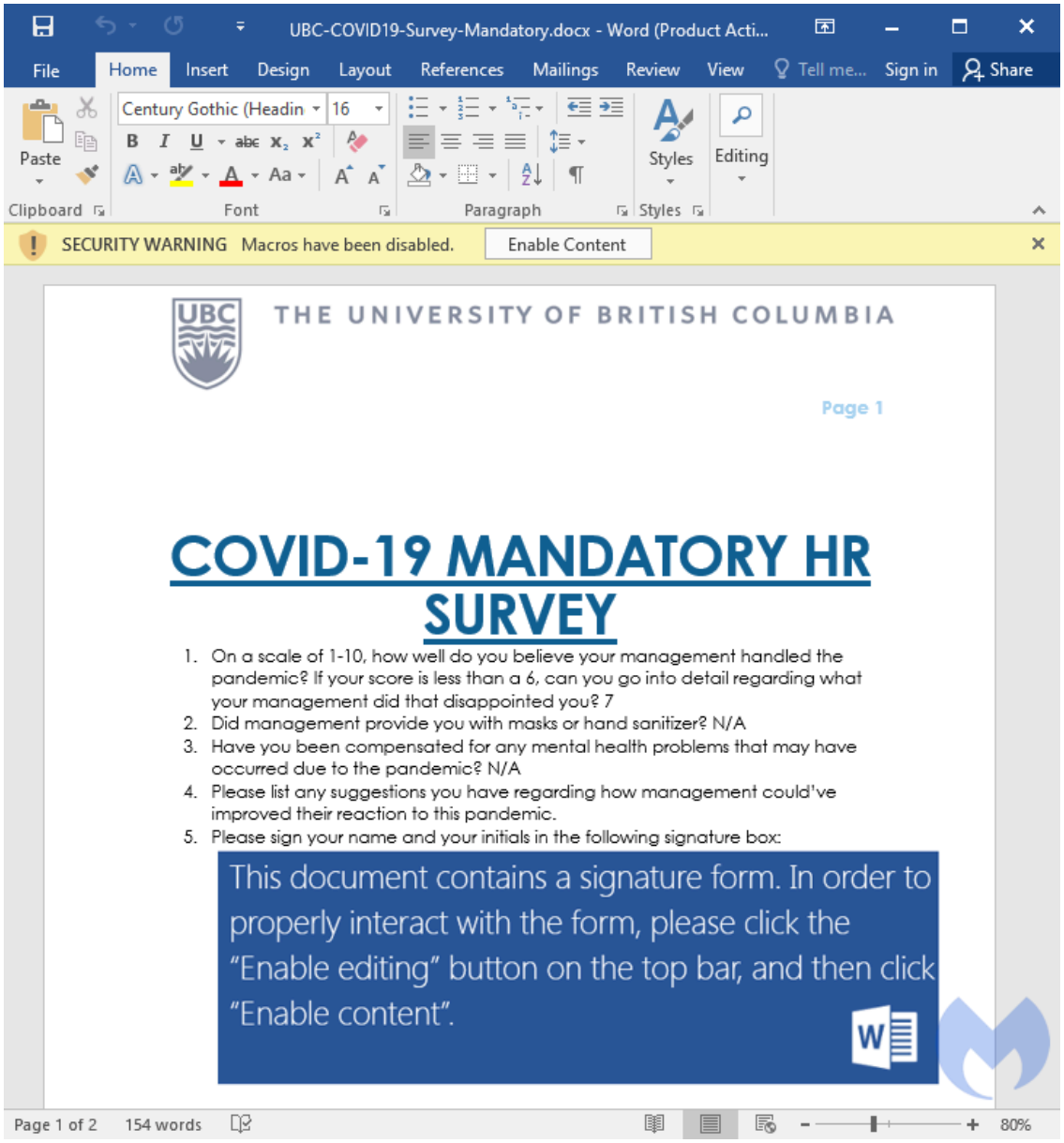
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

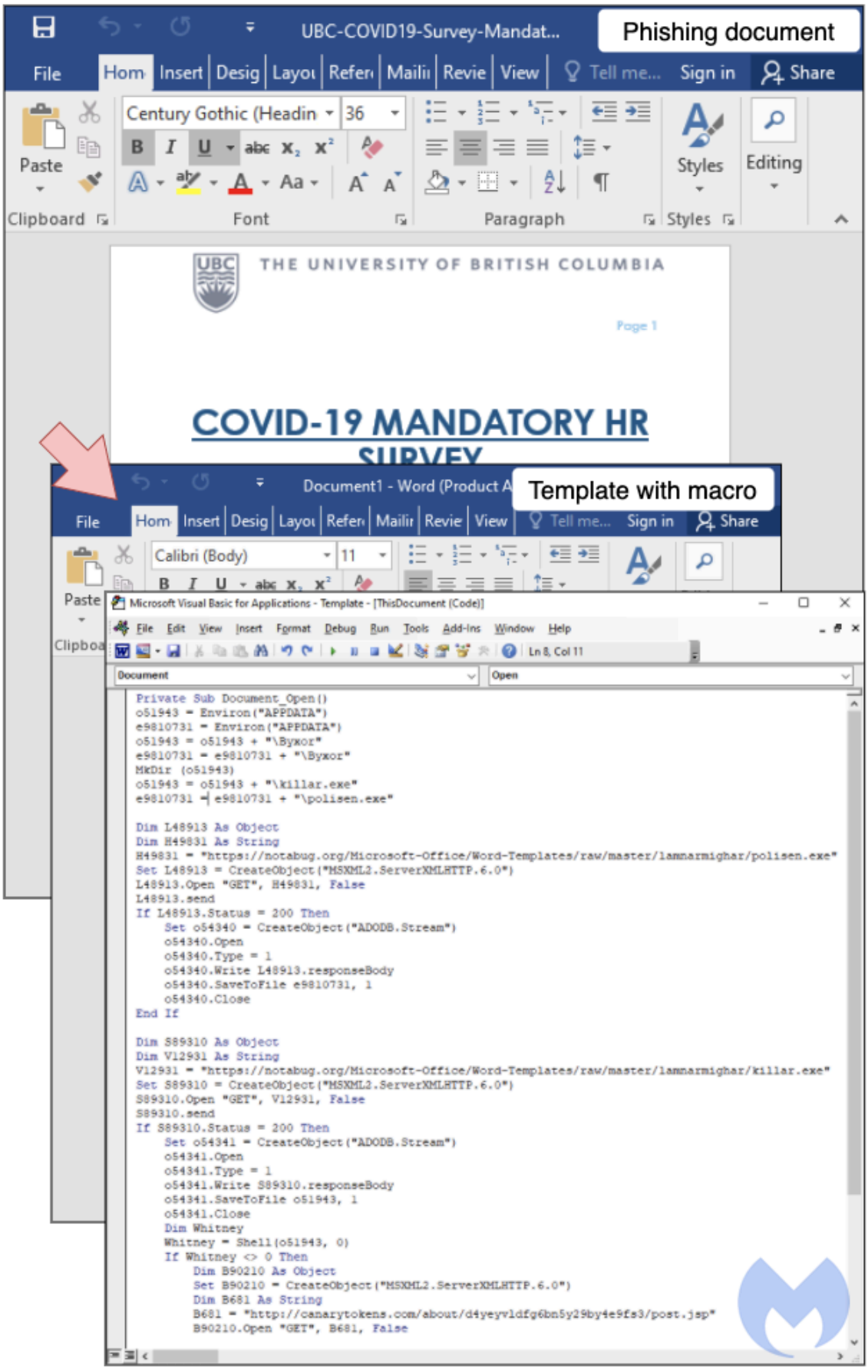
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

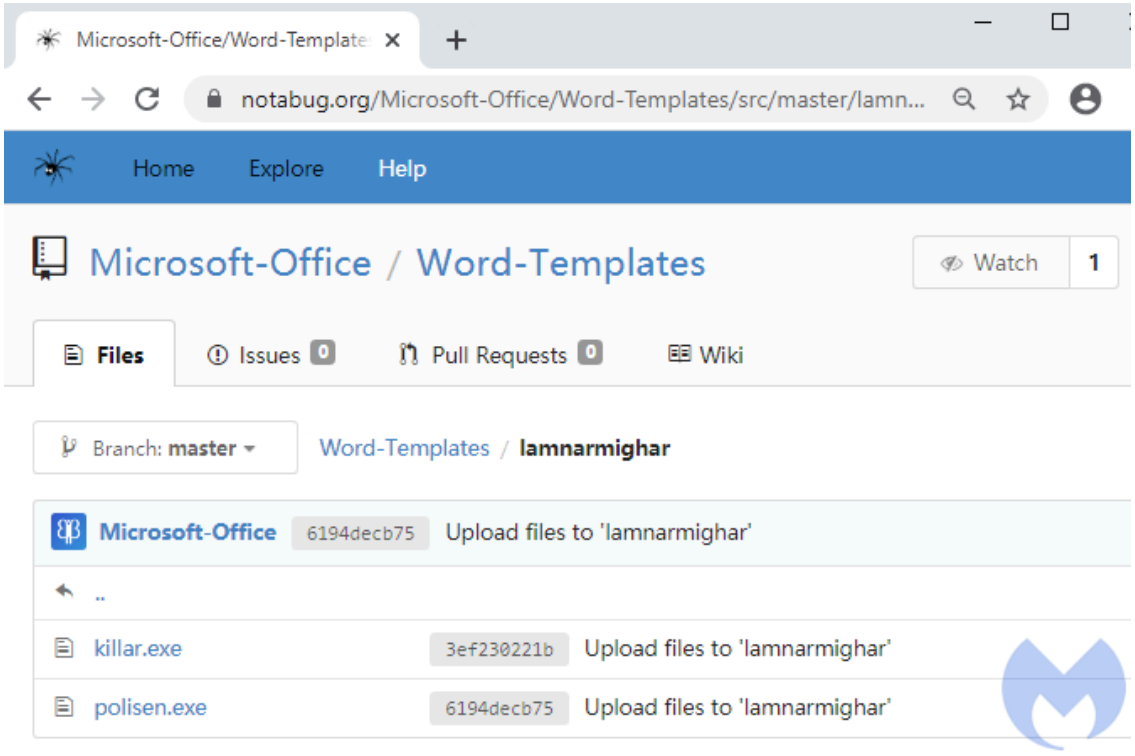
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



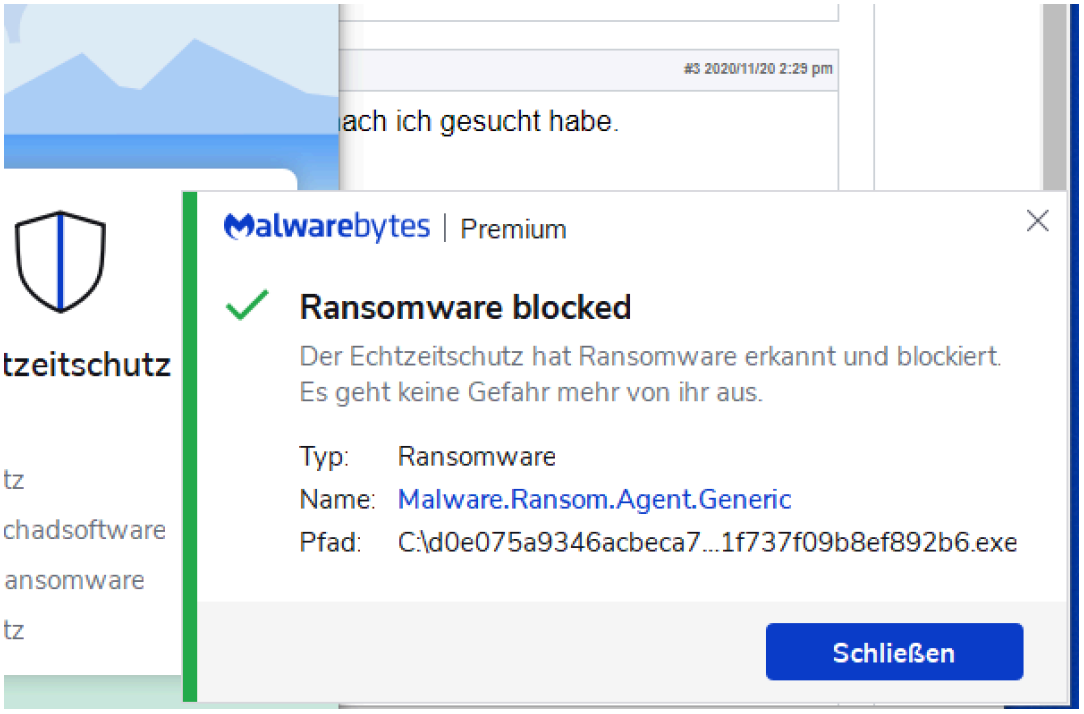
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

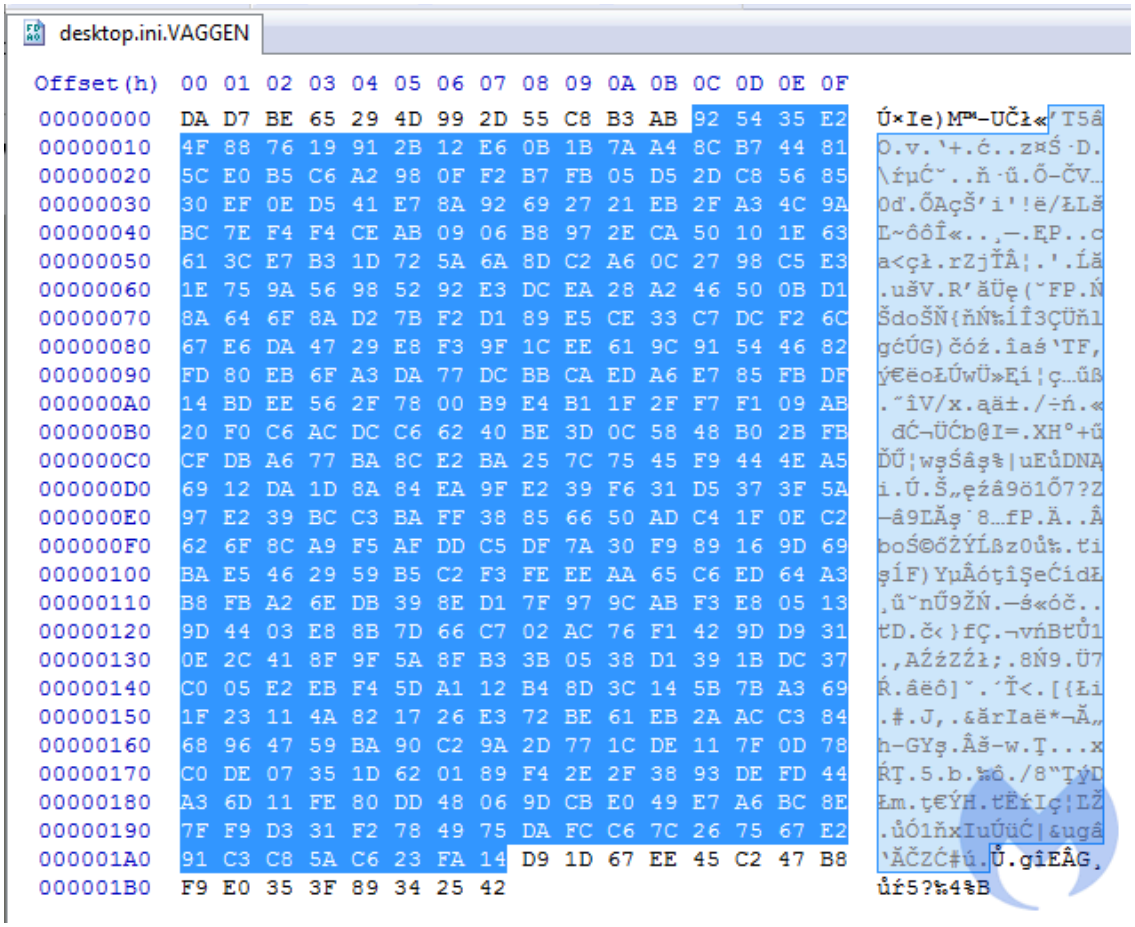
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

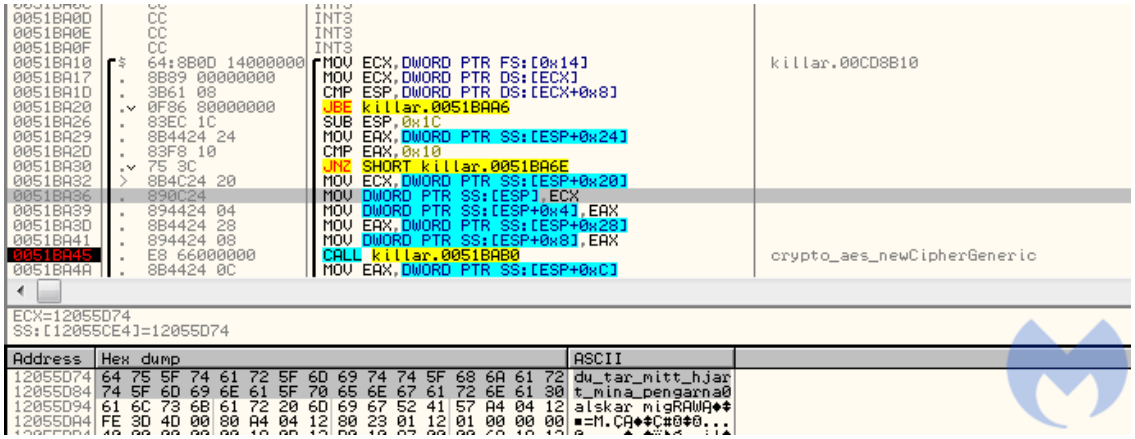


The content of the output file (with .VAGGEN extension) contains:

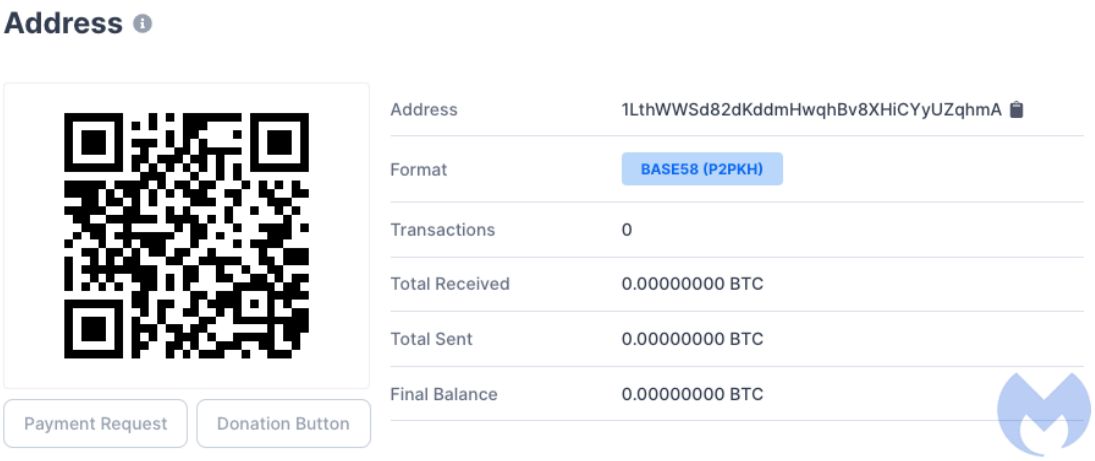
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

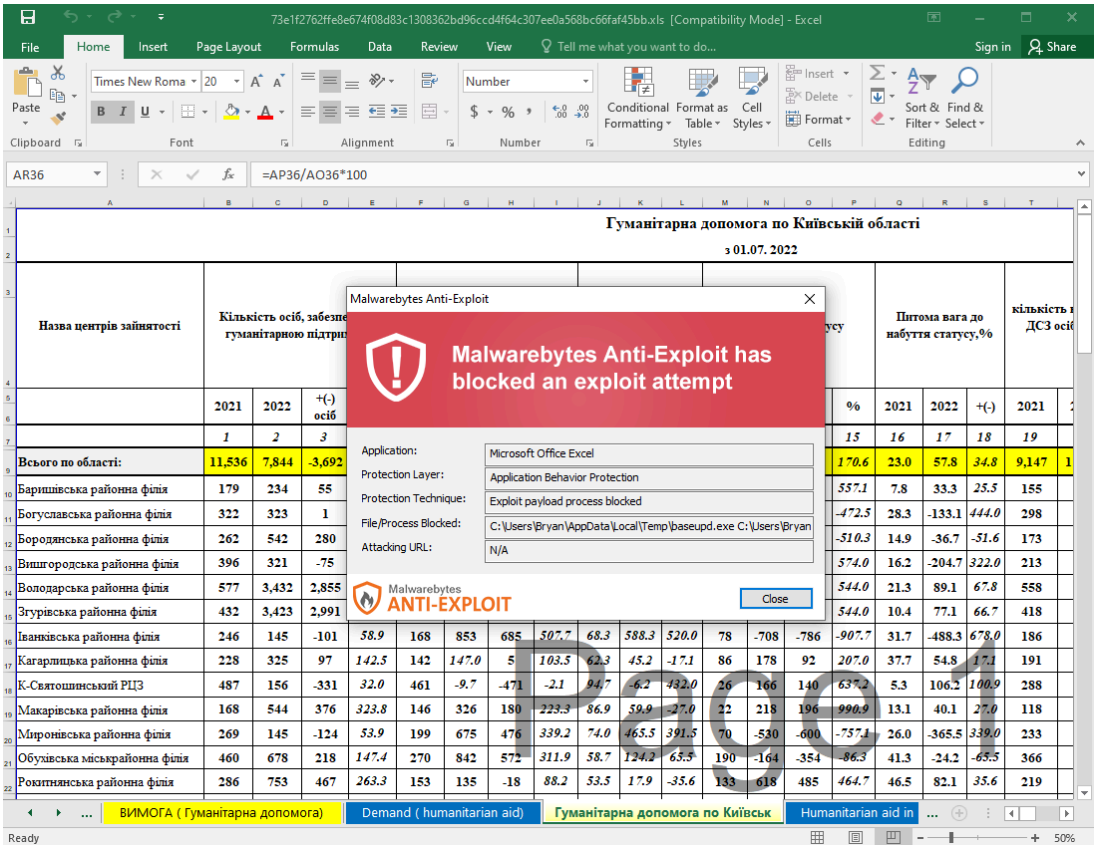
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

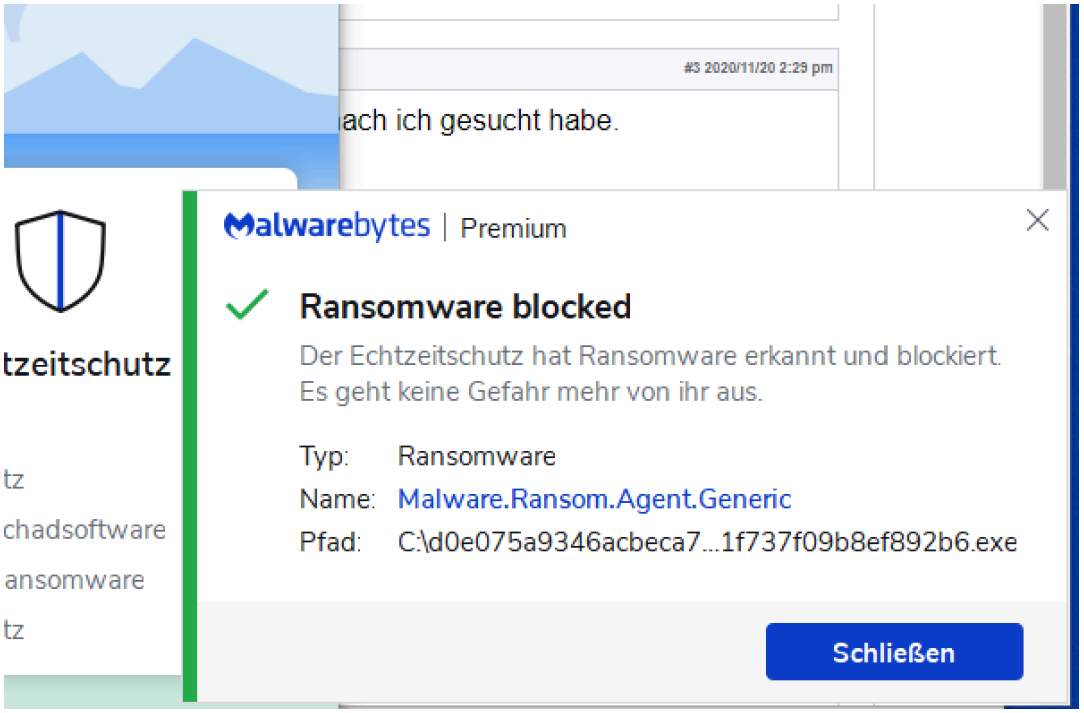
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

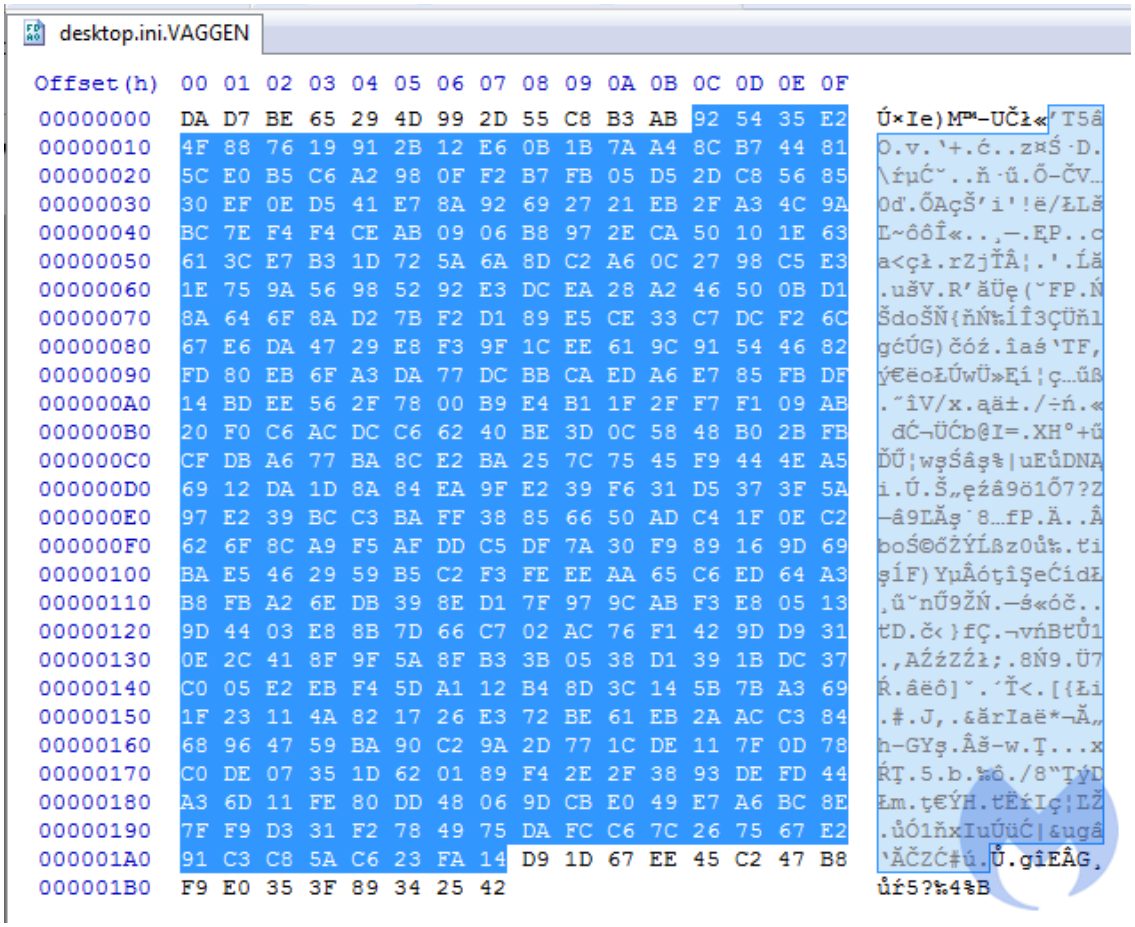
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

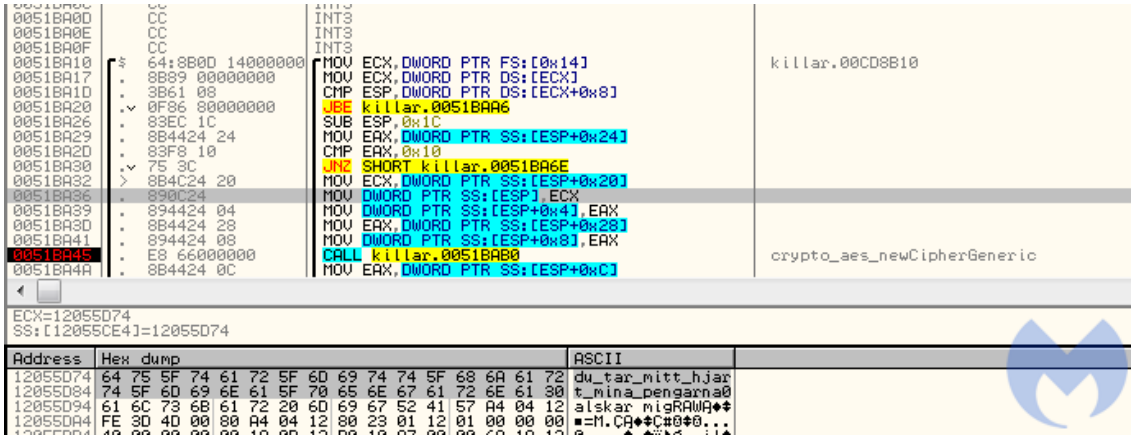


The content of the output file (with .VAGGEN extension) contains:

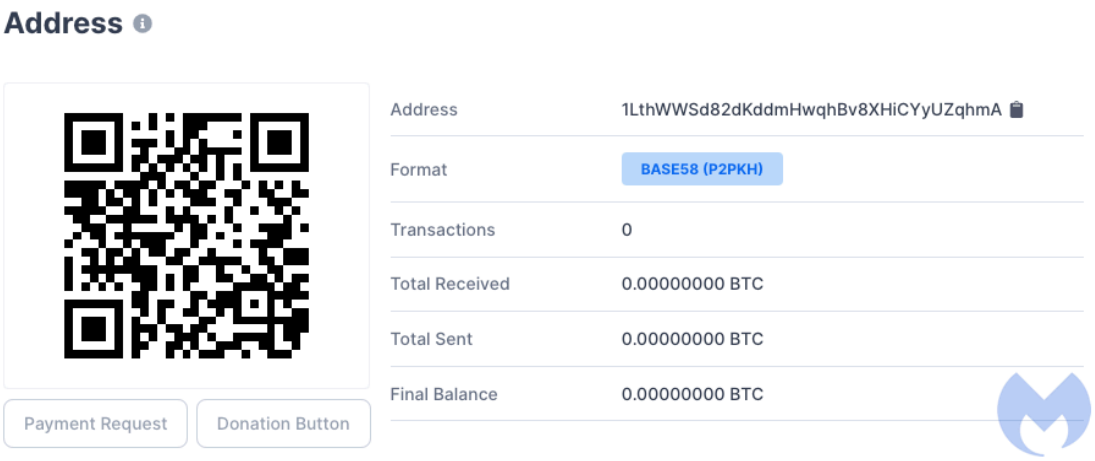
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

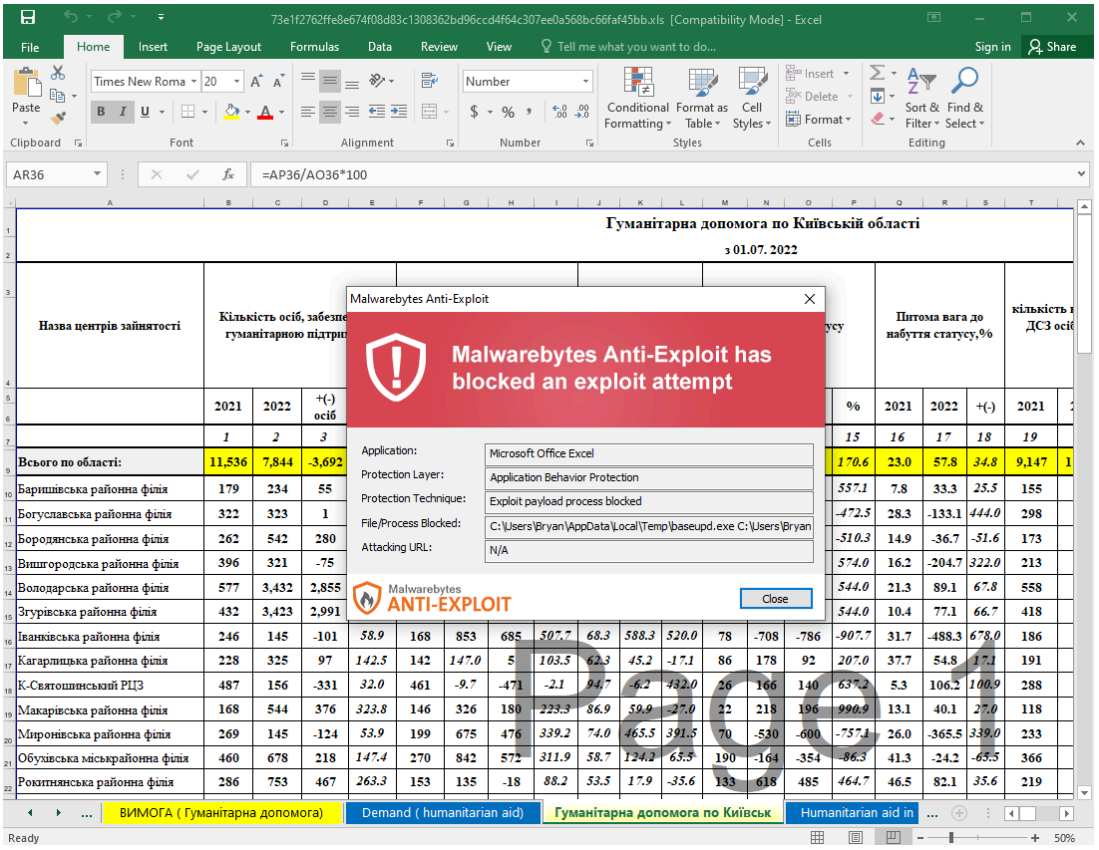
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

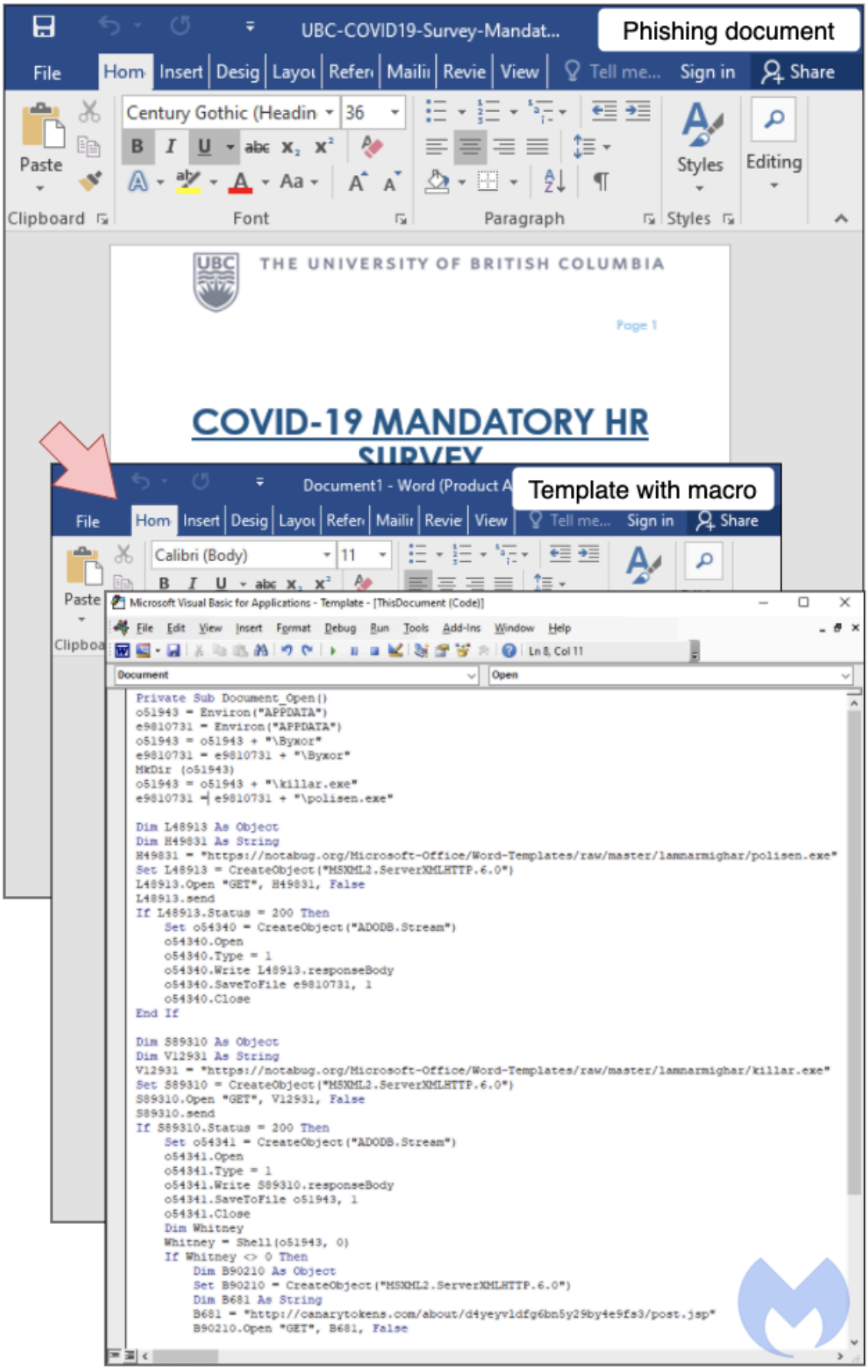
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

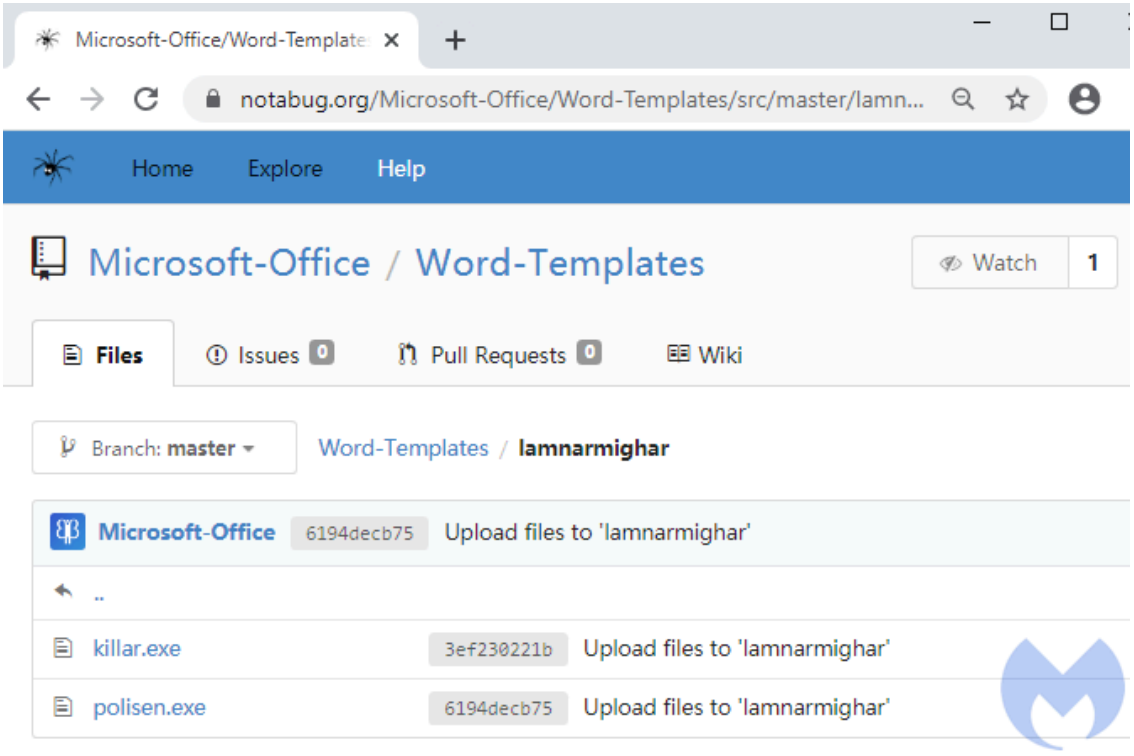
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



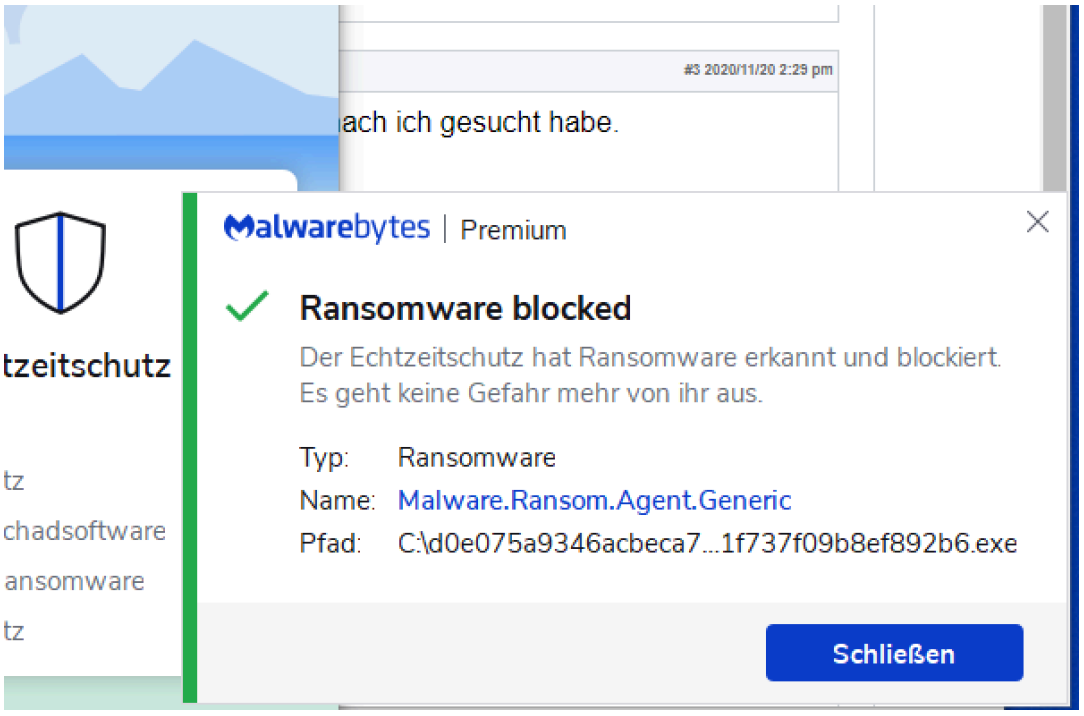
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

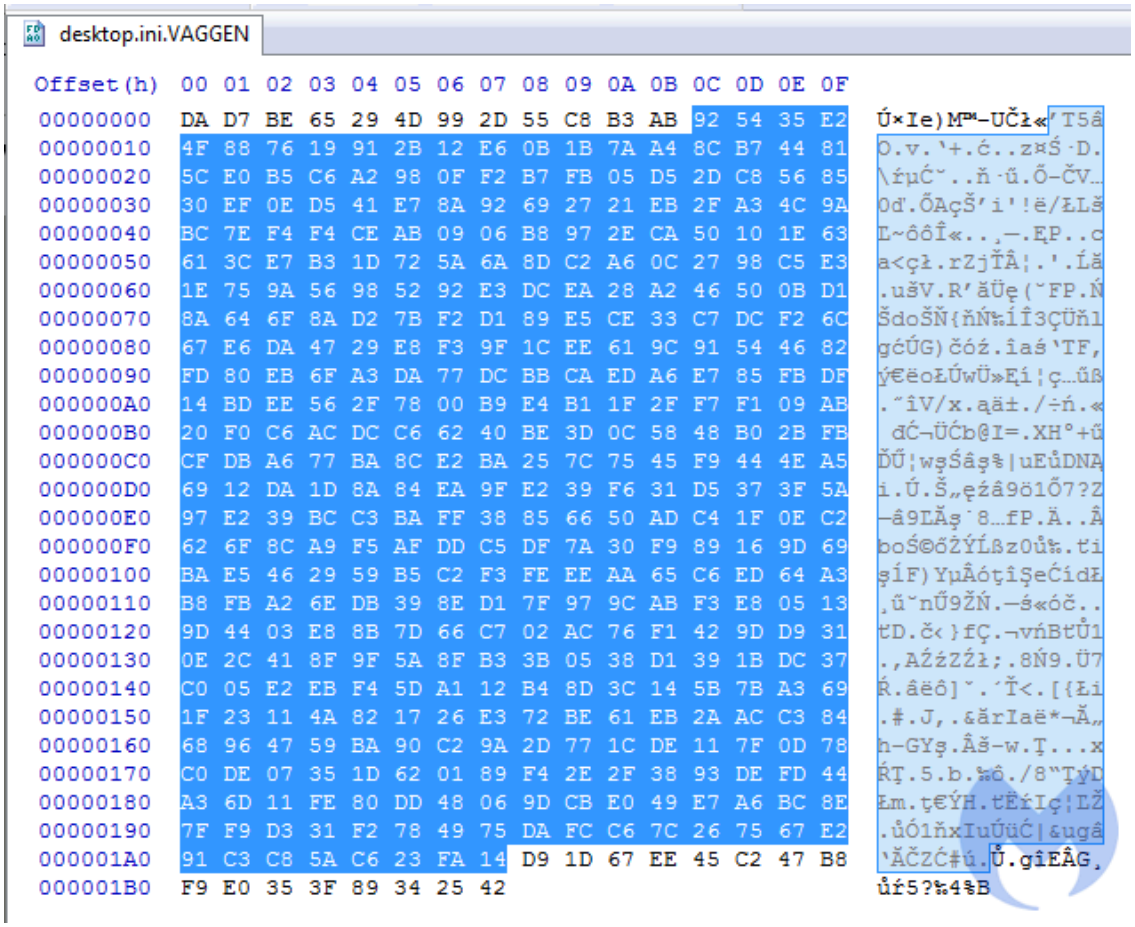
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

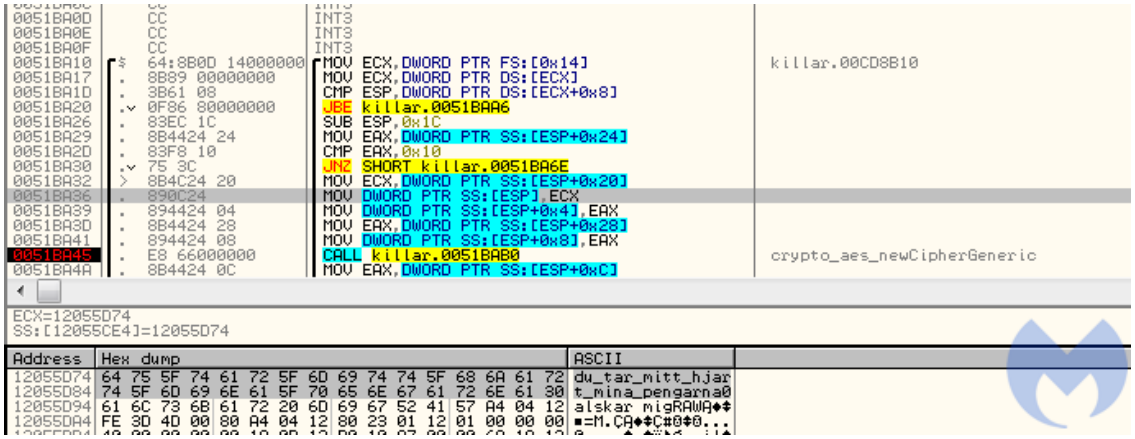


The content of the output file (with .VAGGEN extension) contains:

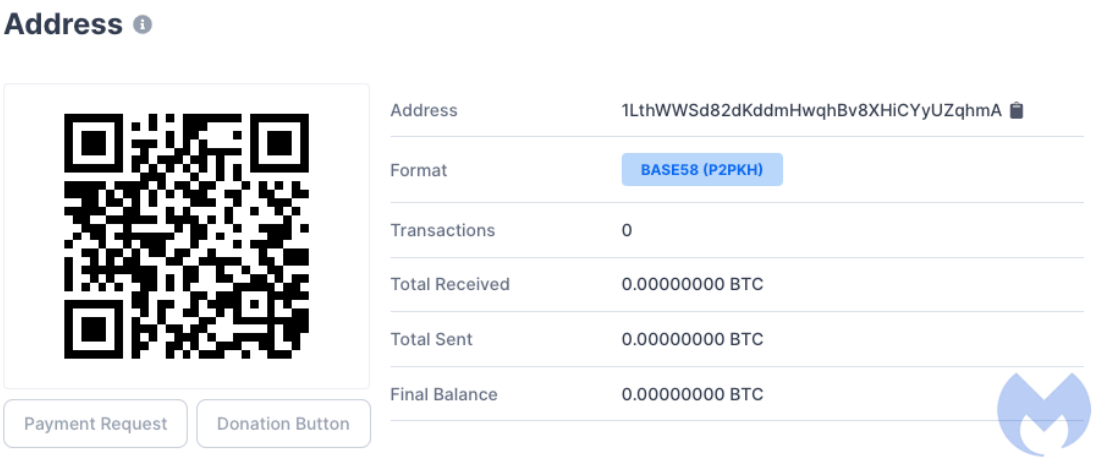
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

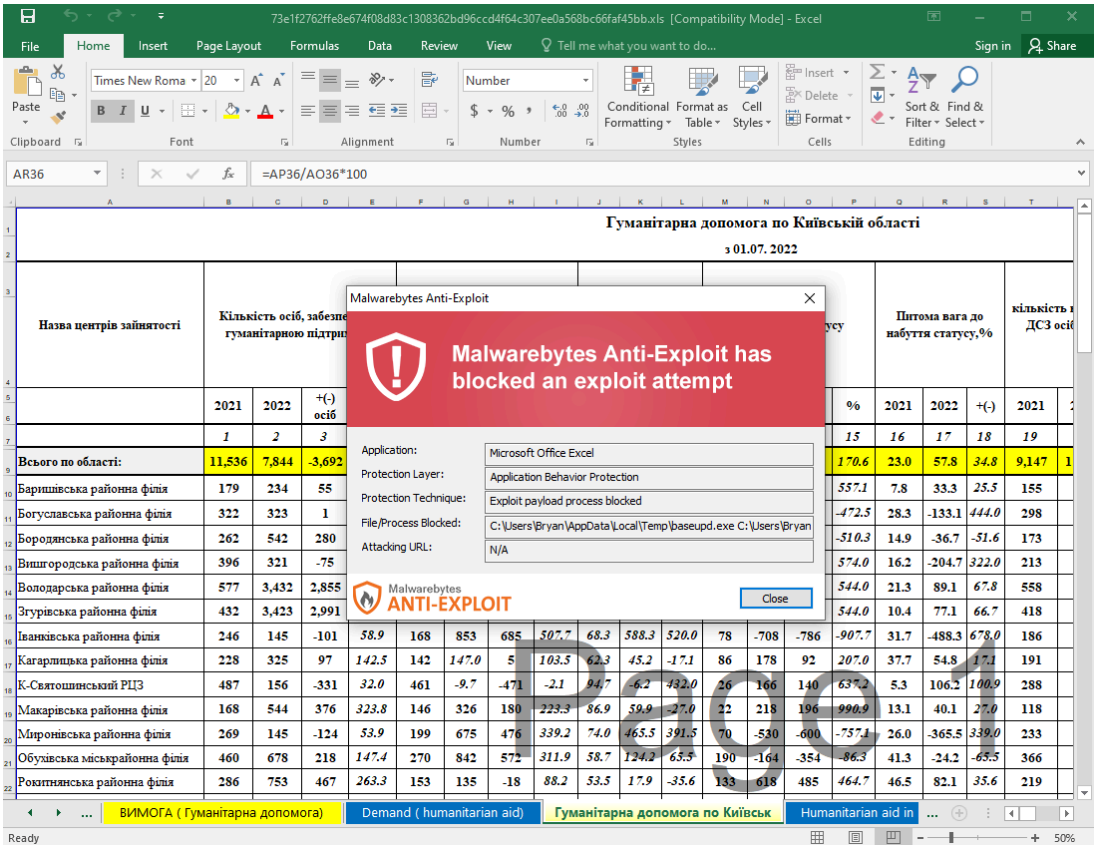
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

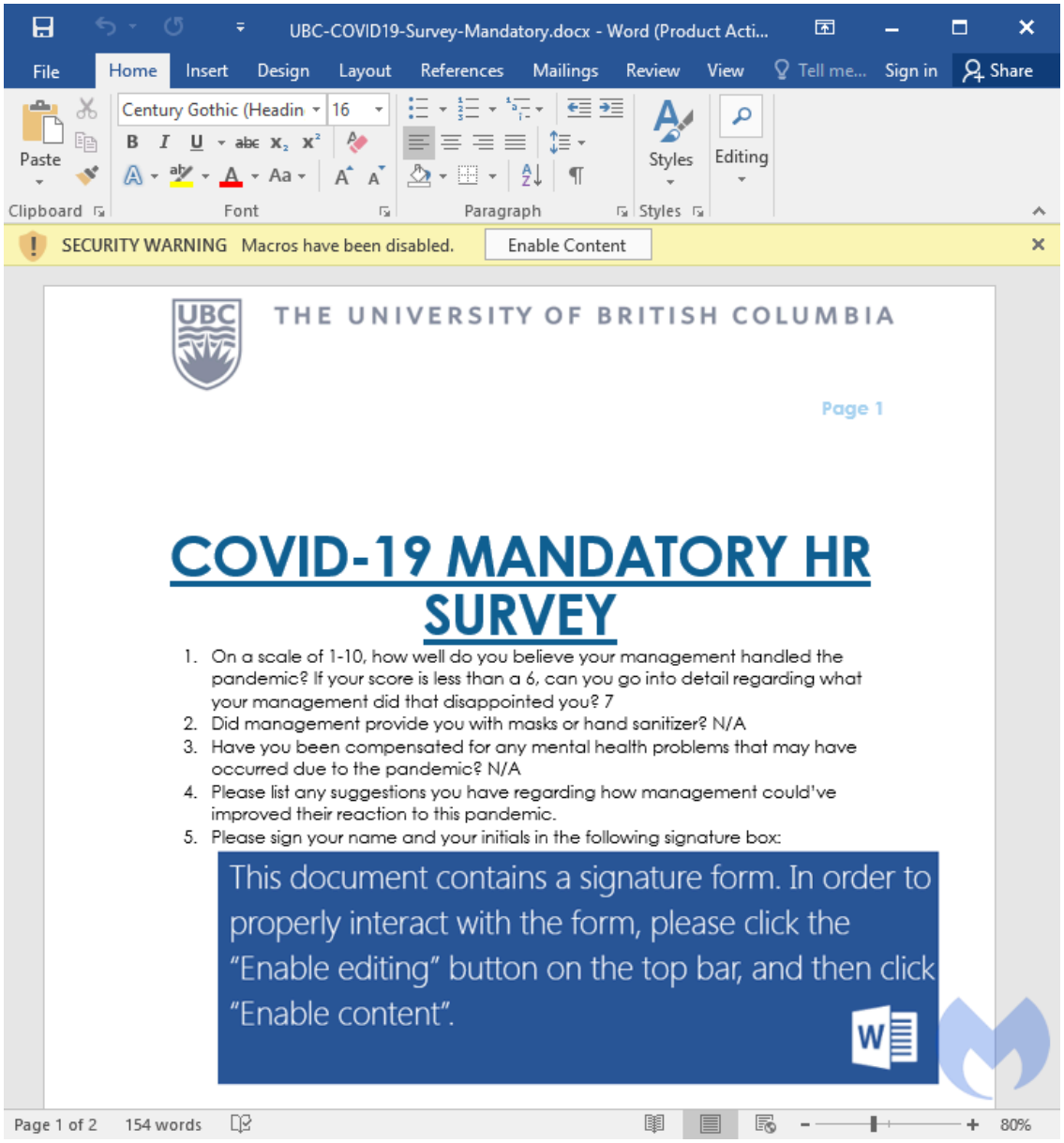
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

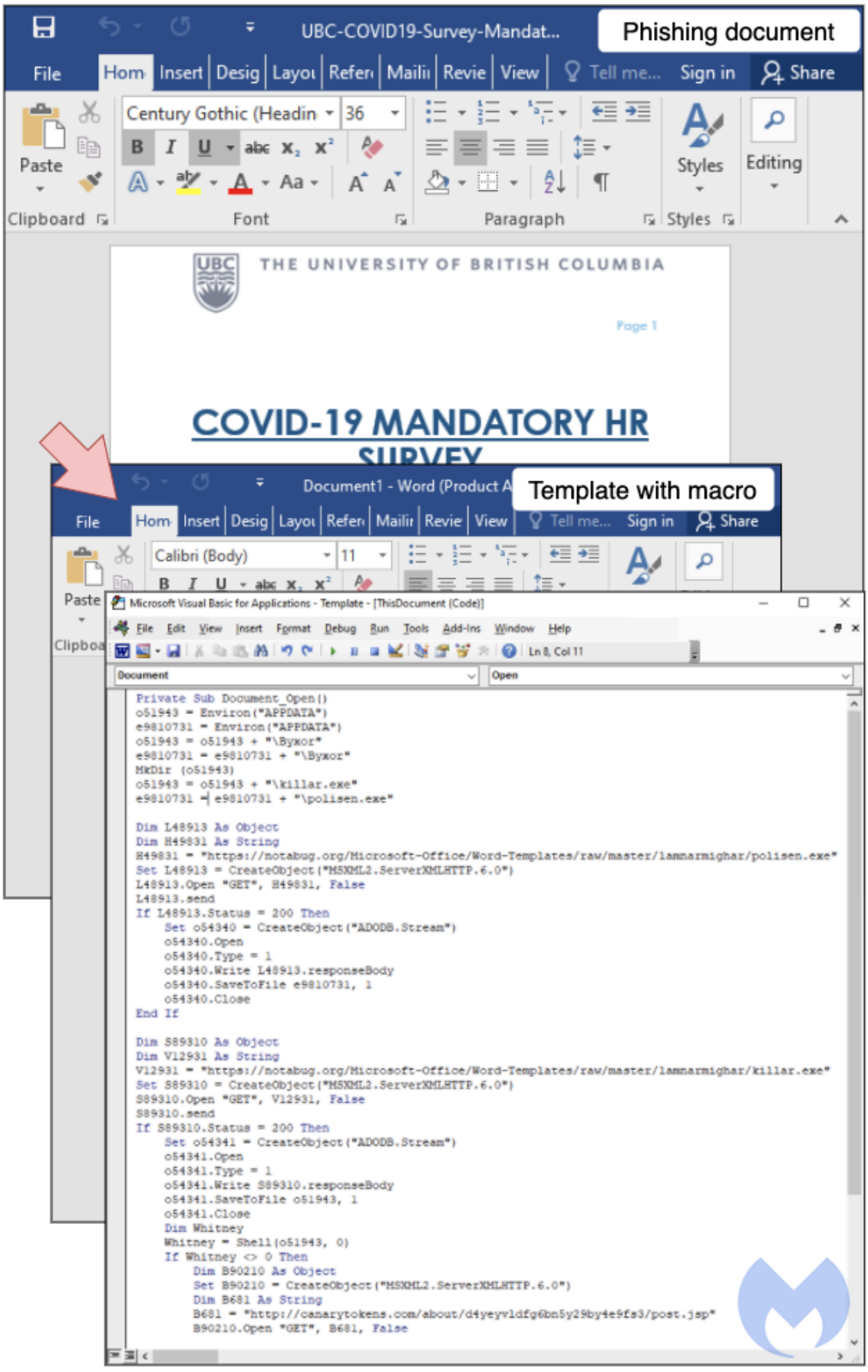
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

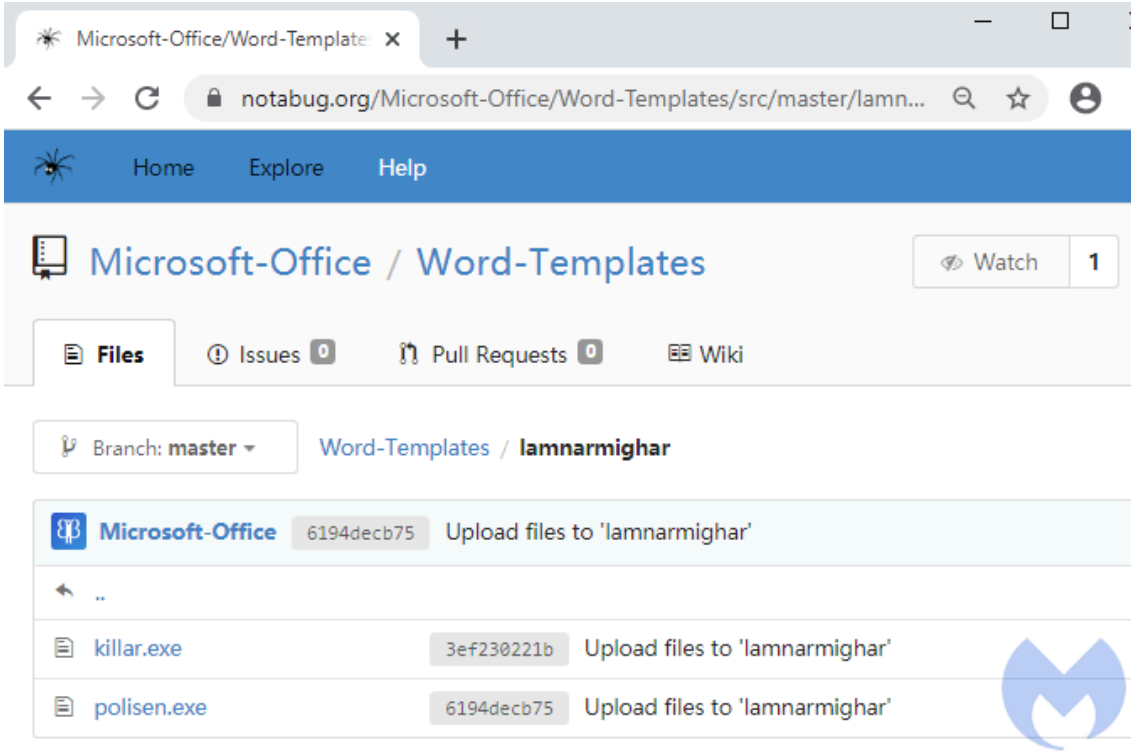
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



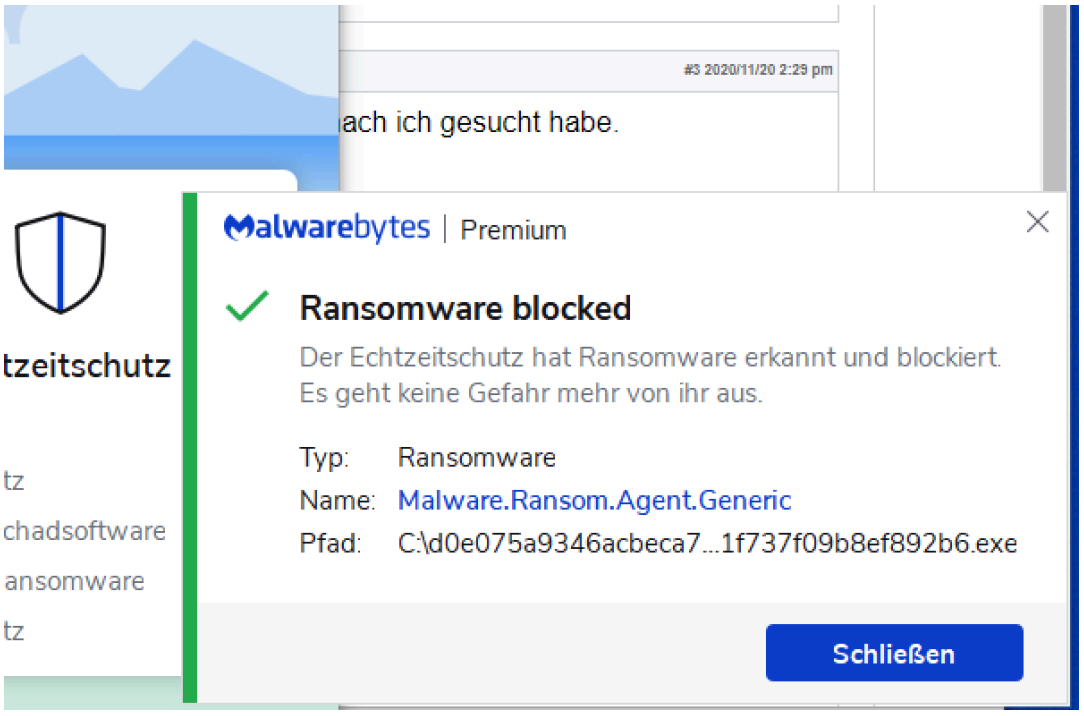
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

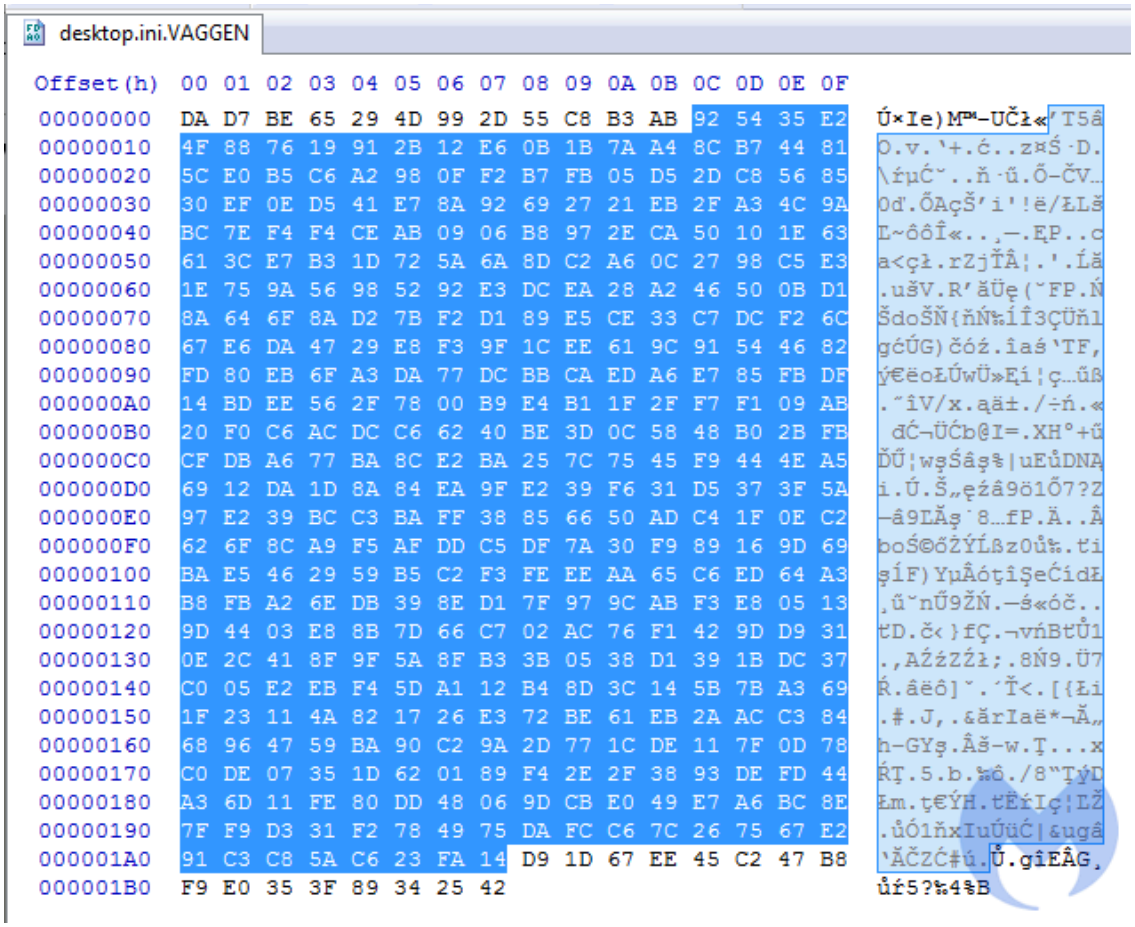
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

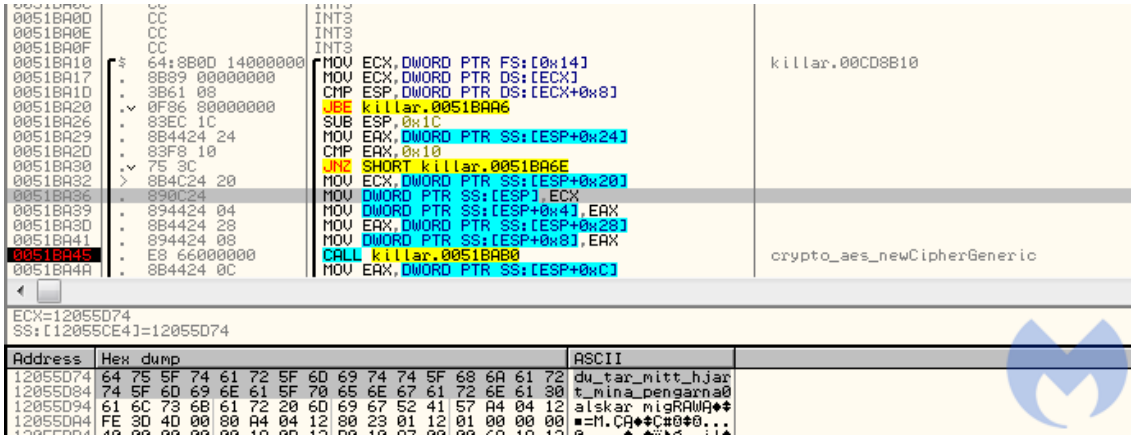


The content of the output file (with .VAGGEN extension) contains:

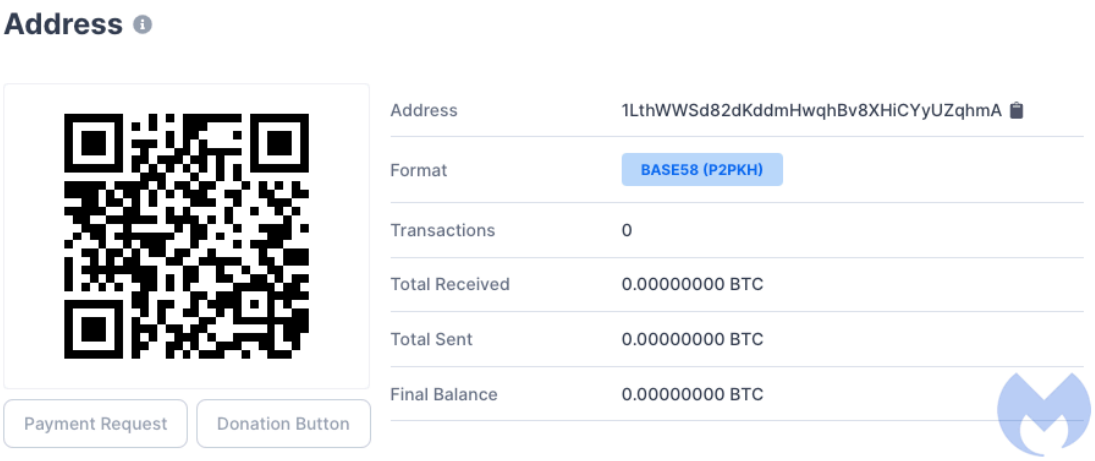
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

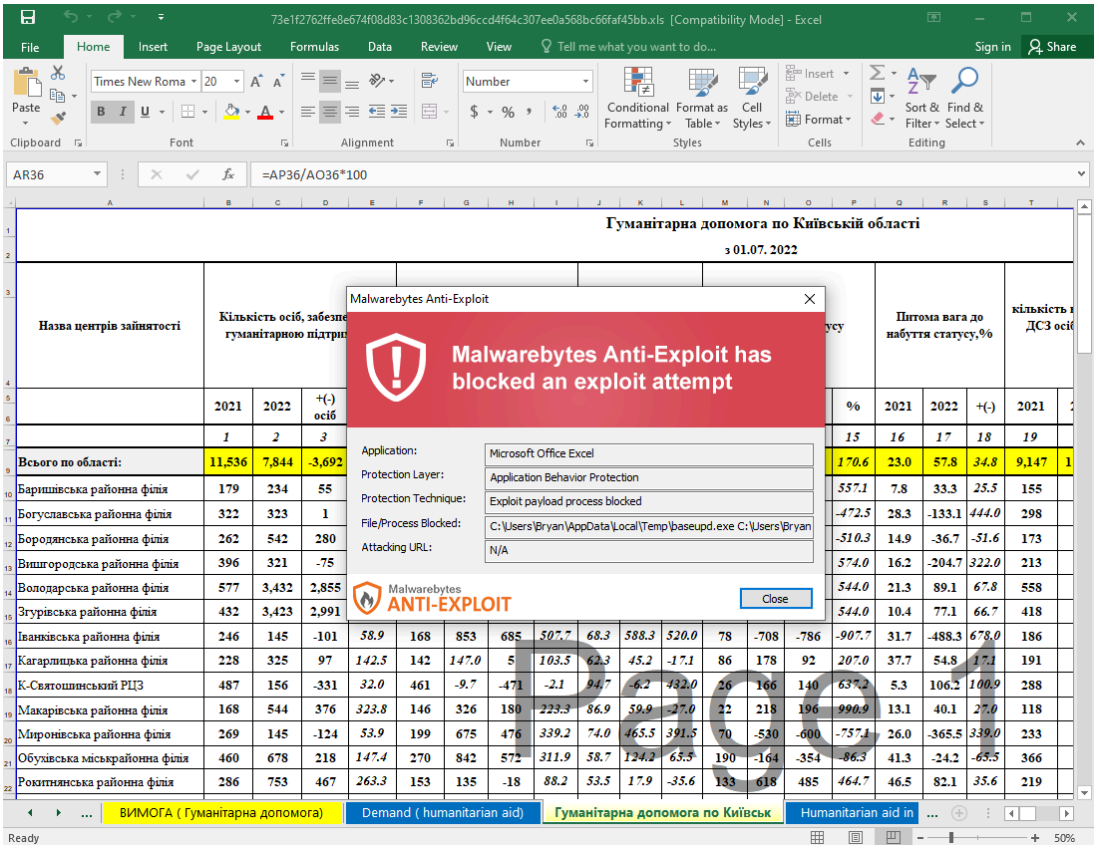
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

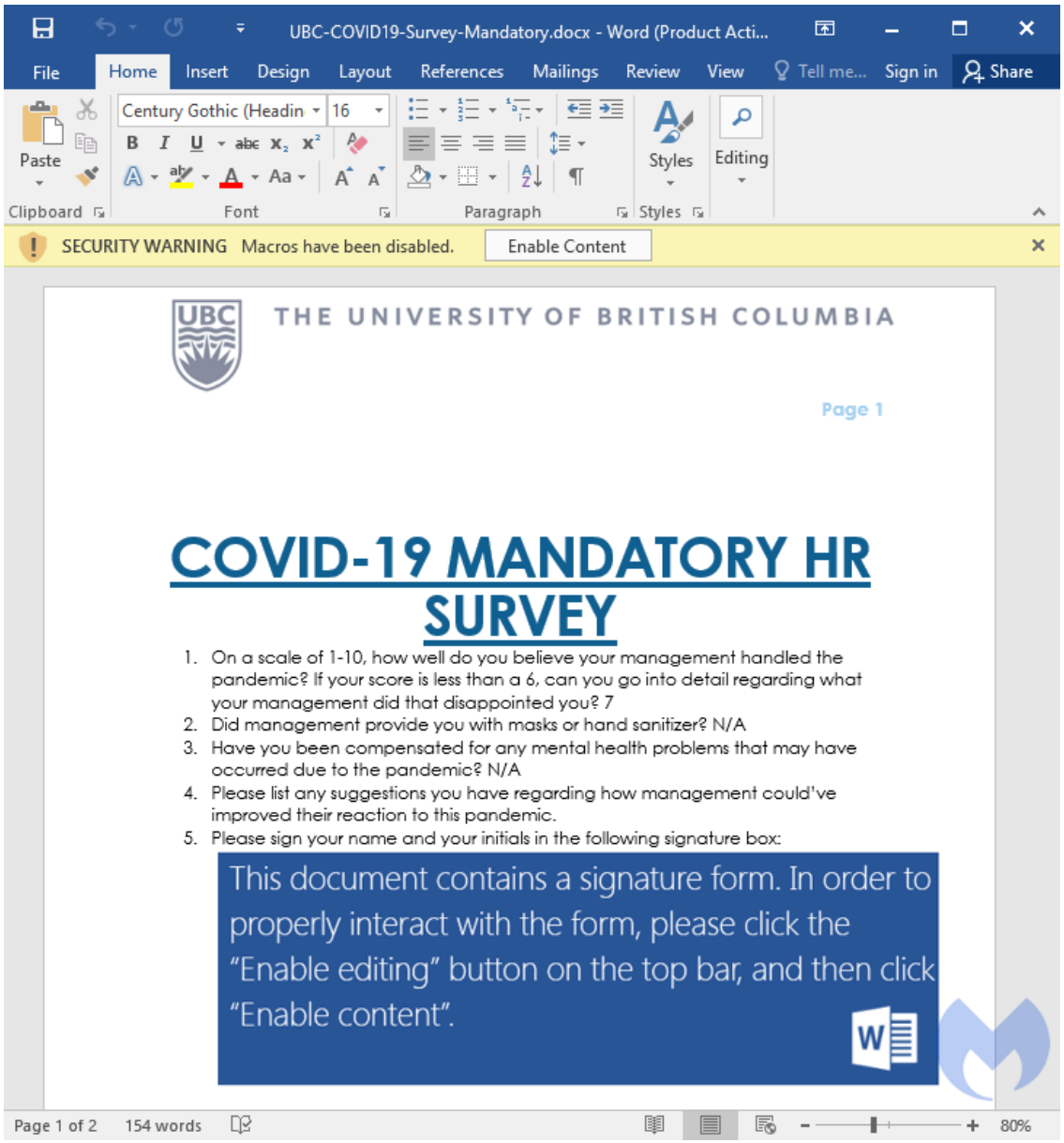
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

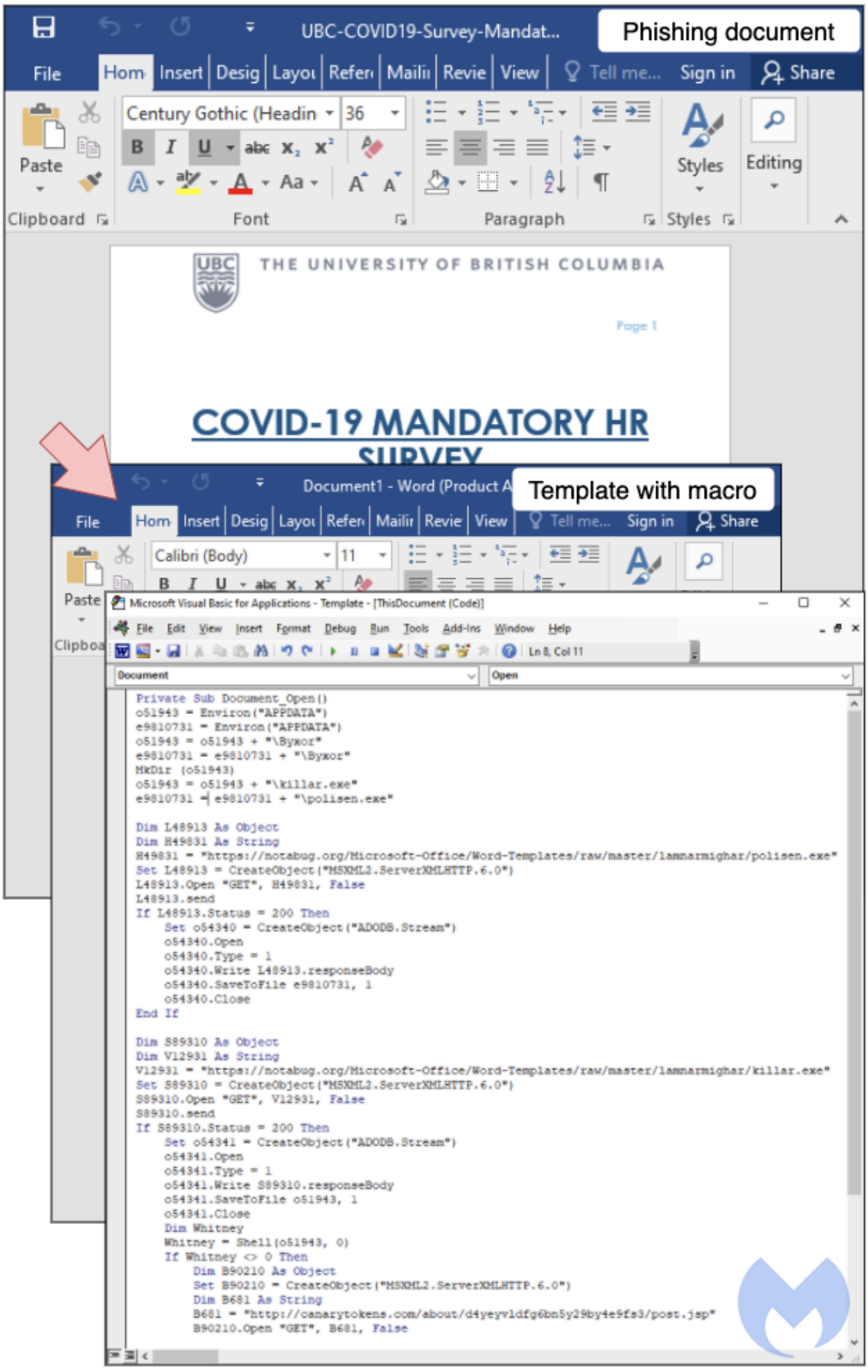
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

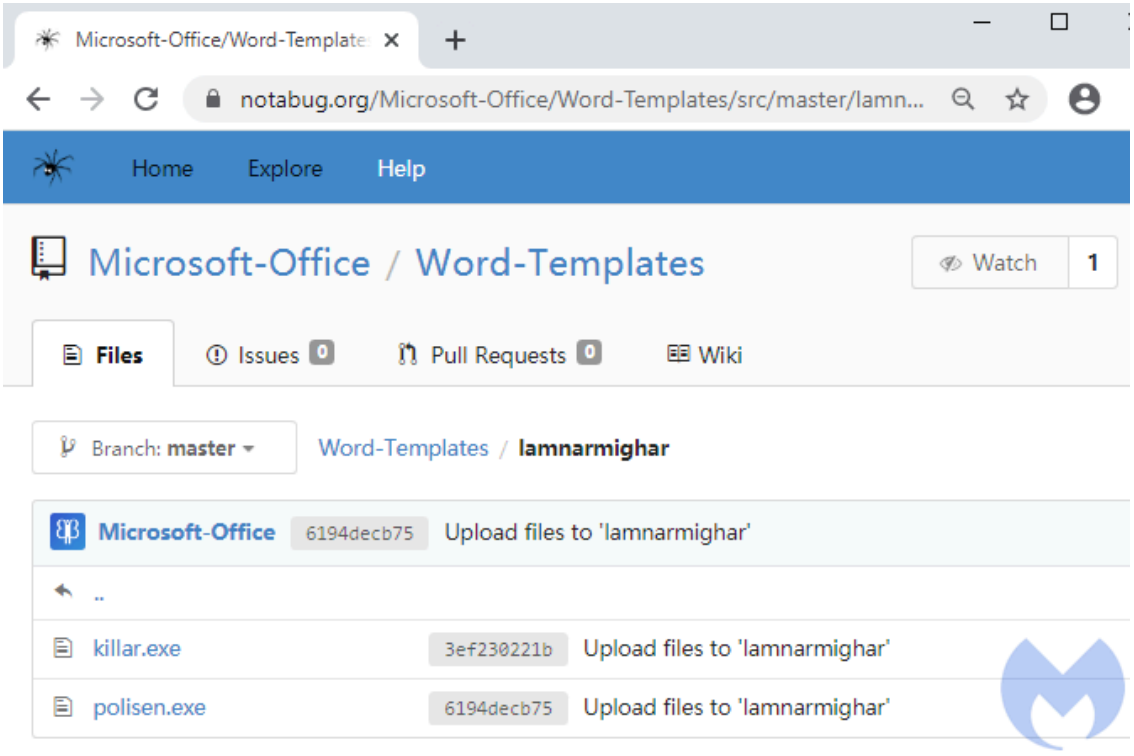
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



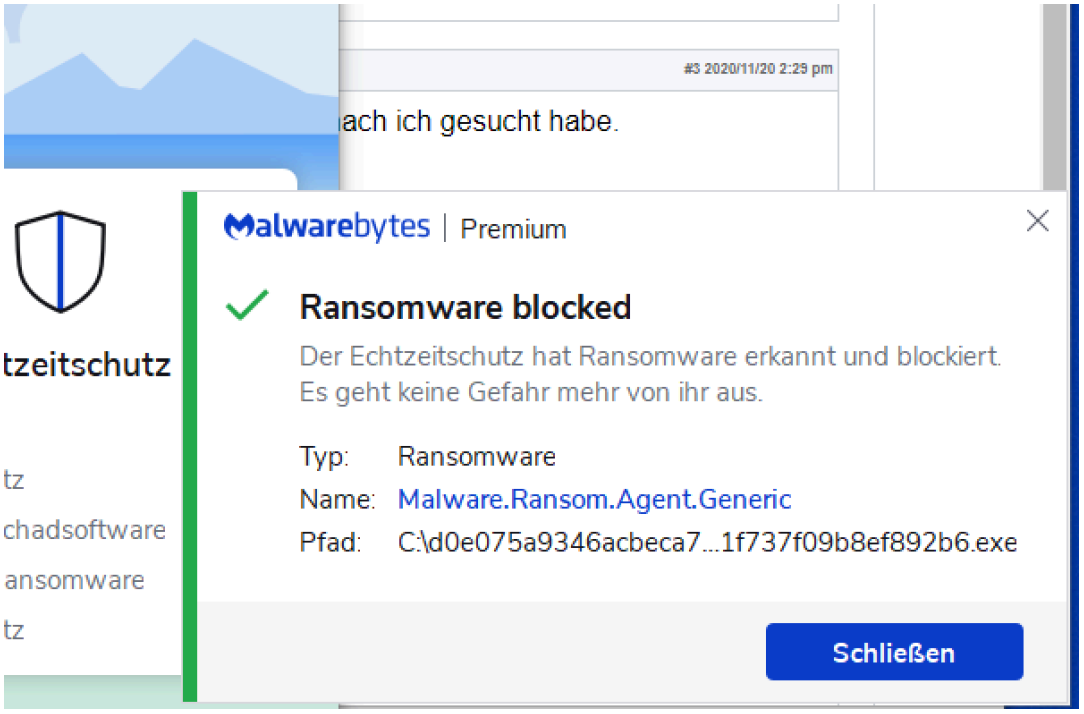
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

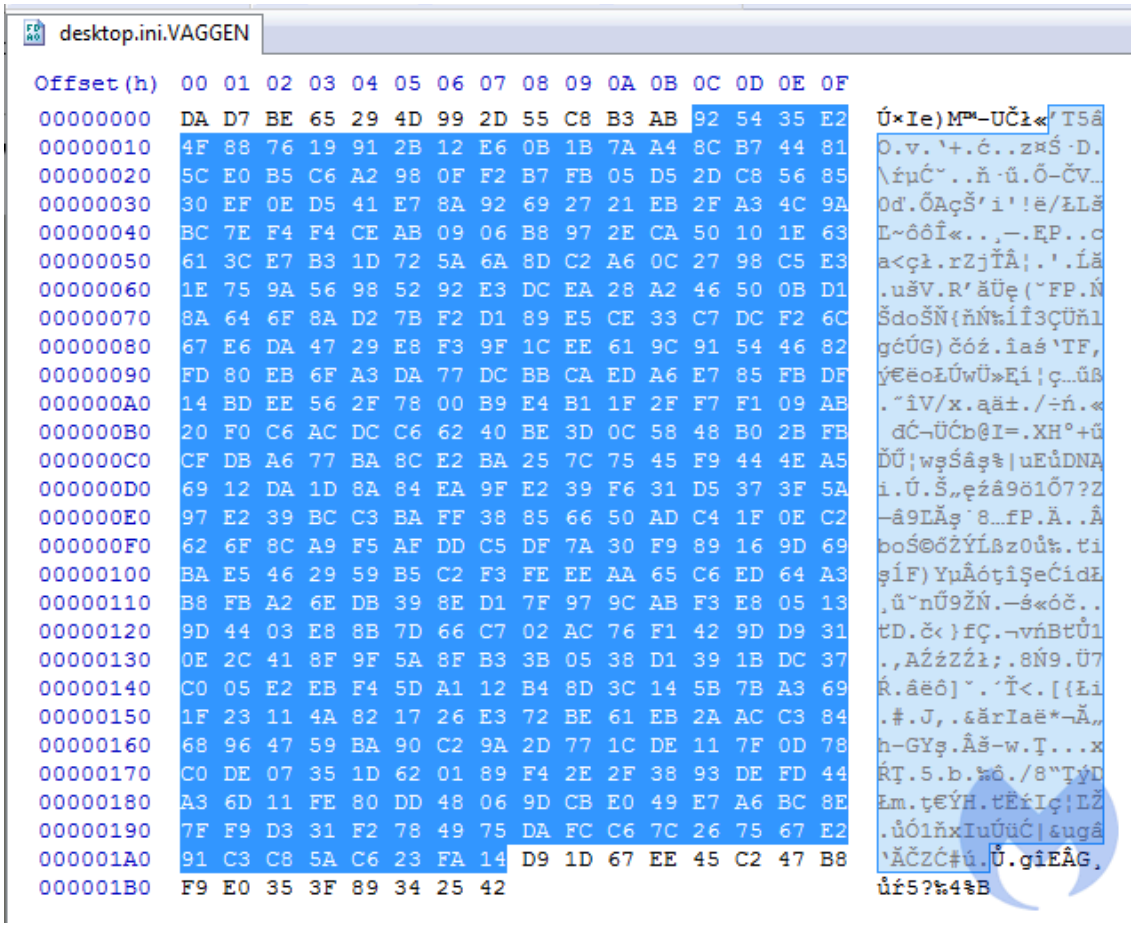
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

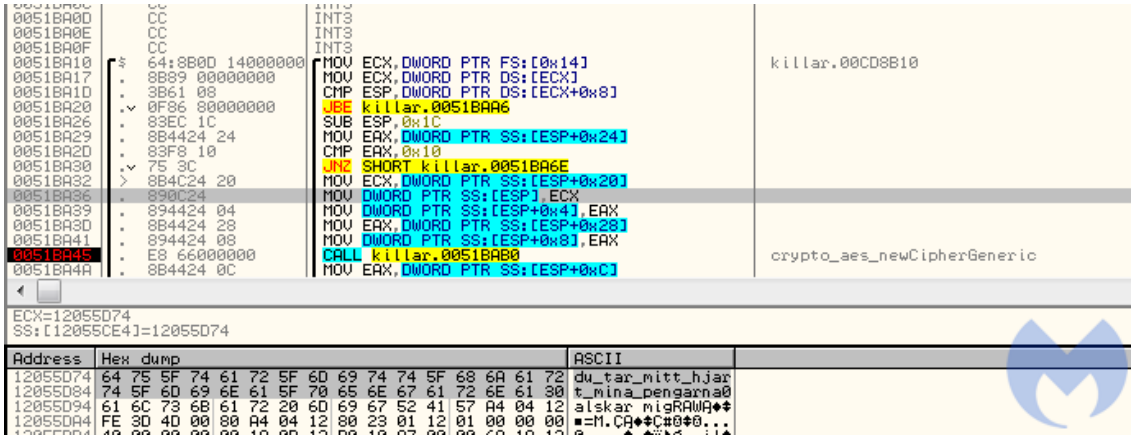


The content of the output file (with .VAGGEN extension) contains:

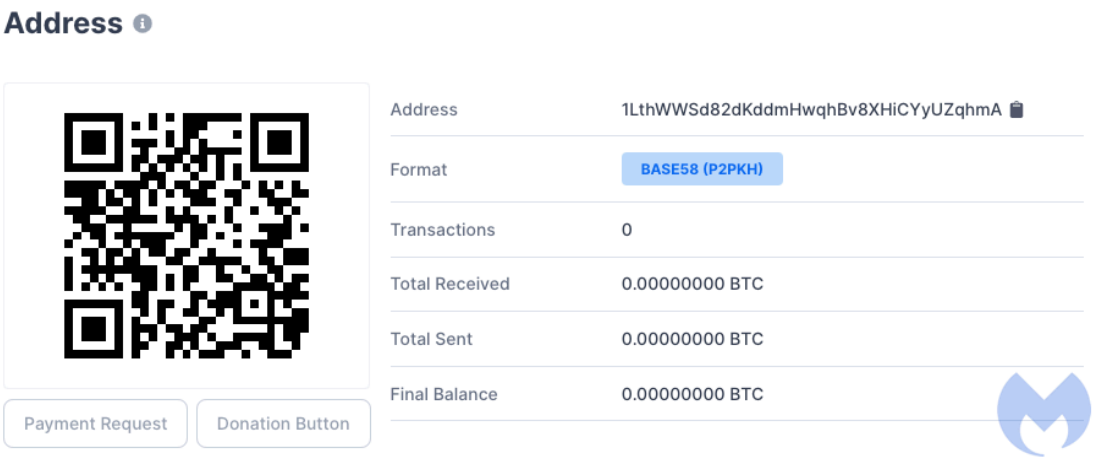
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

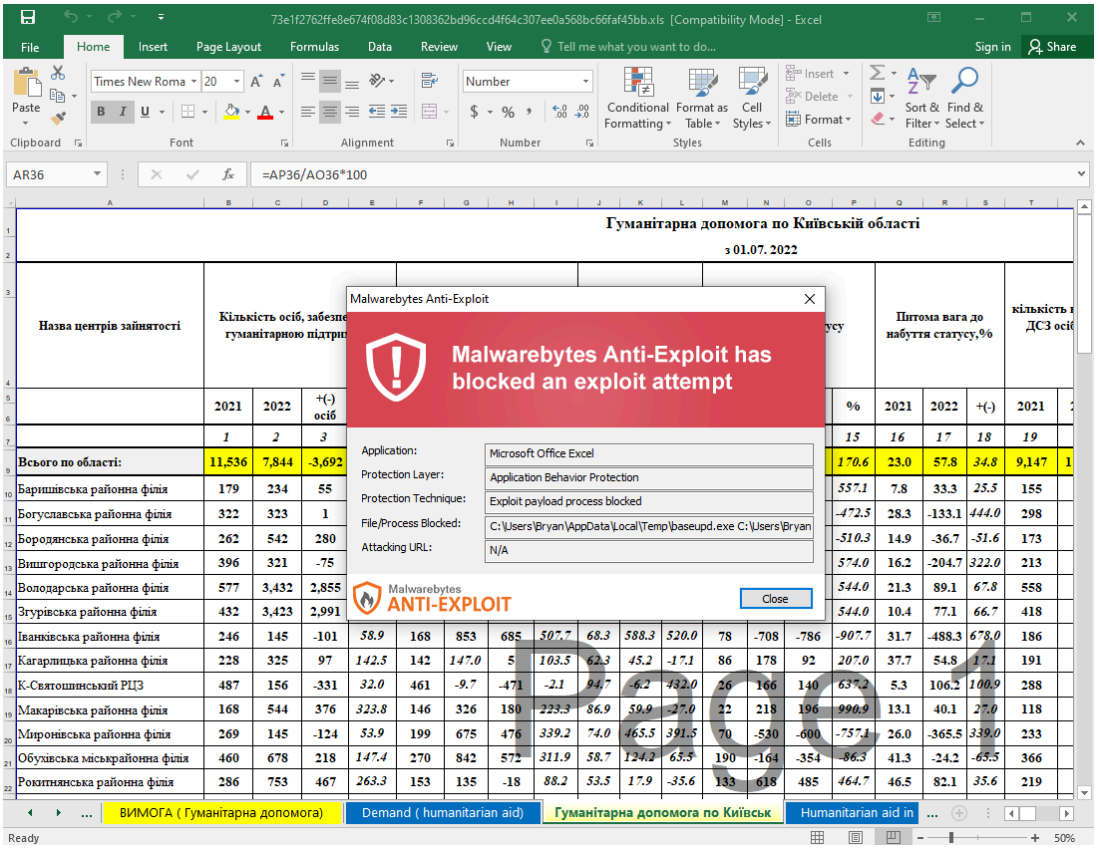
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

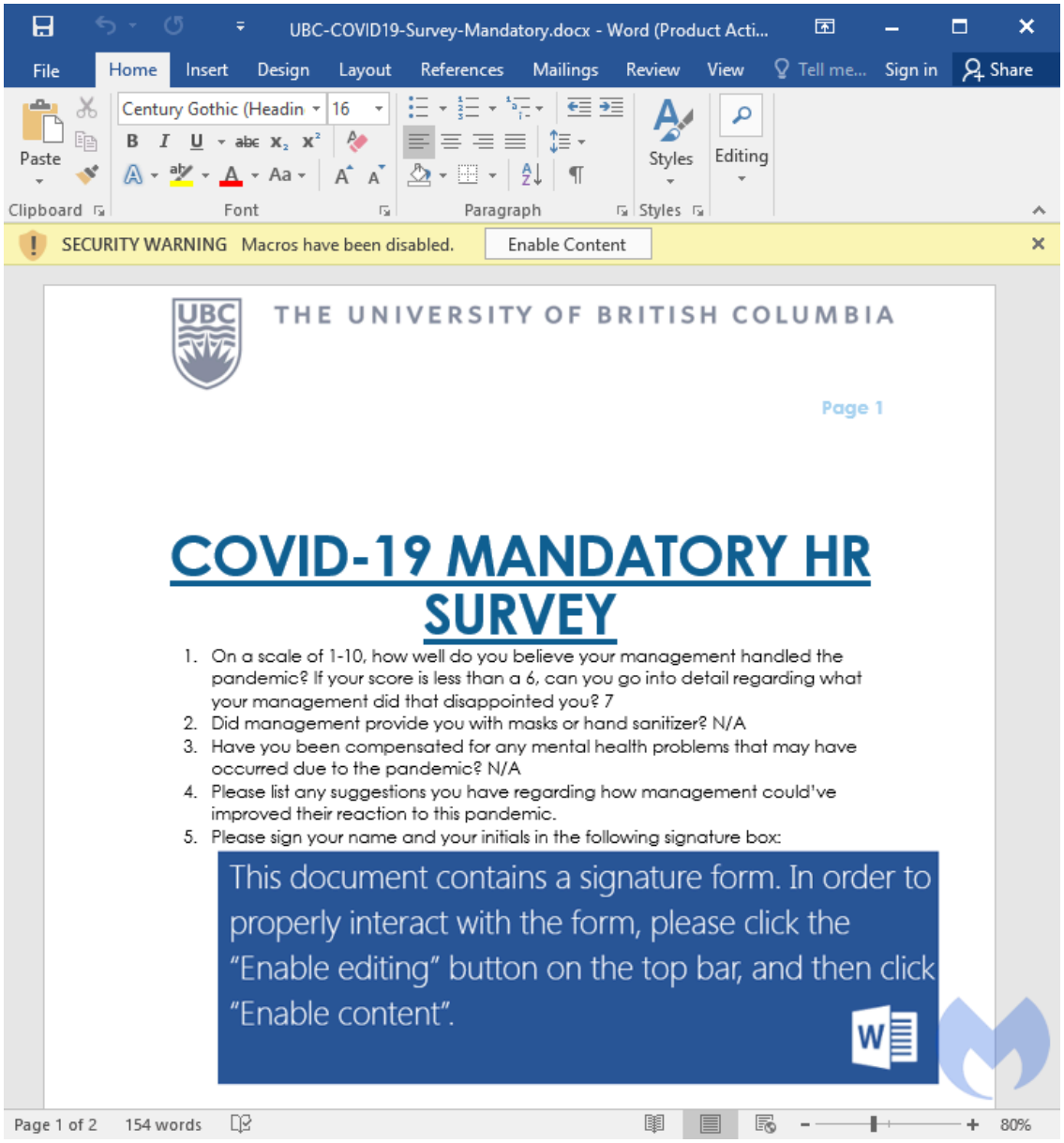
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

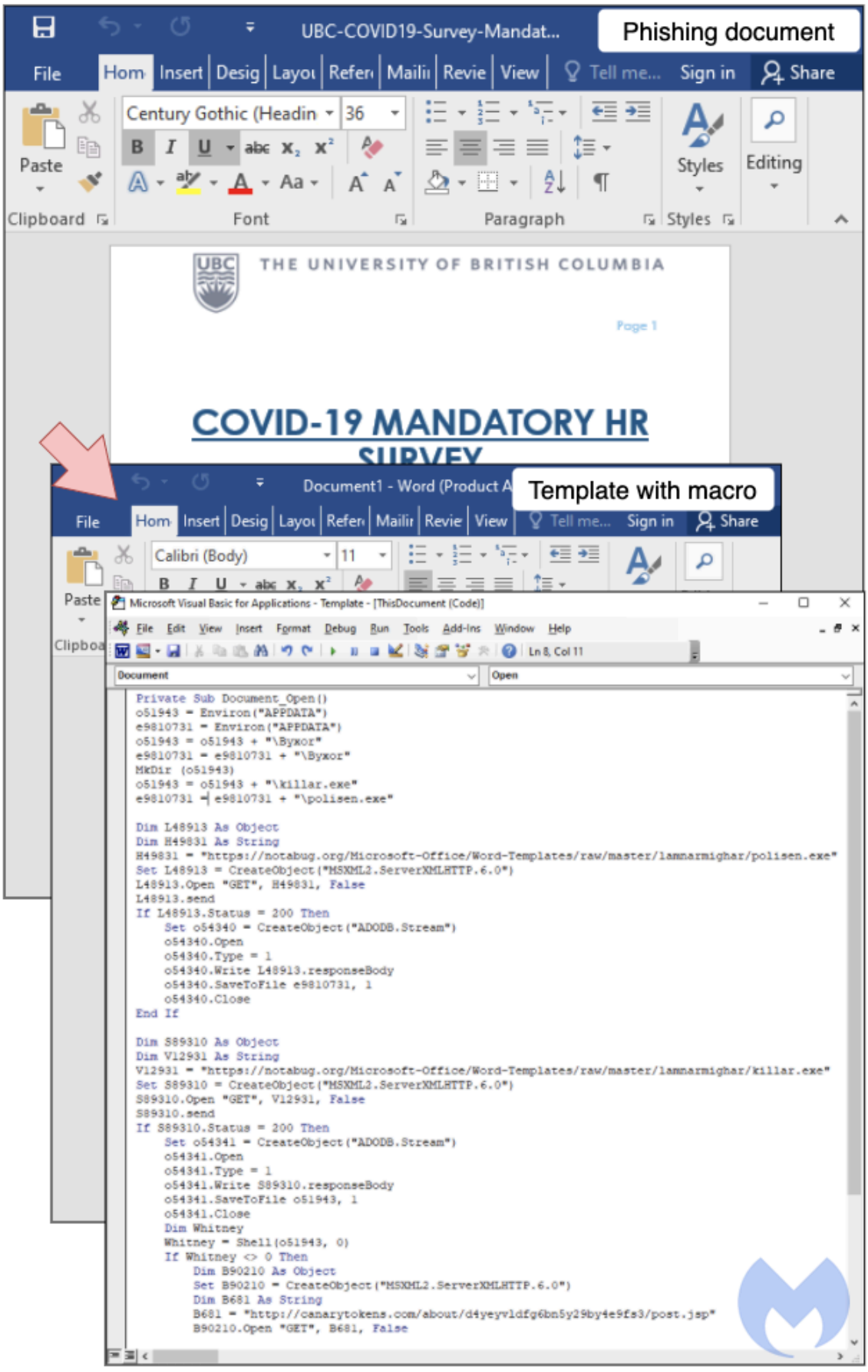
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

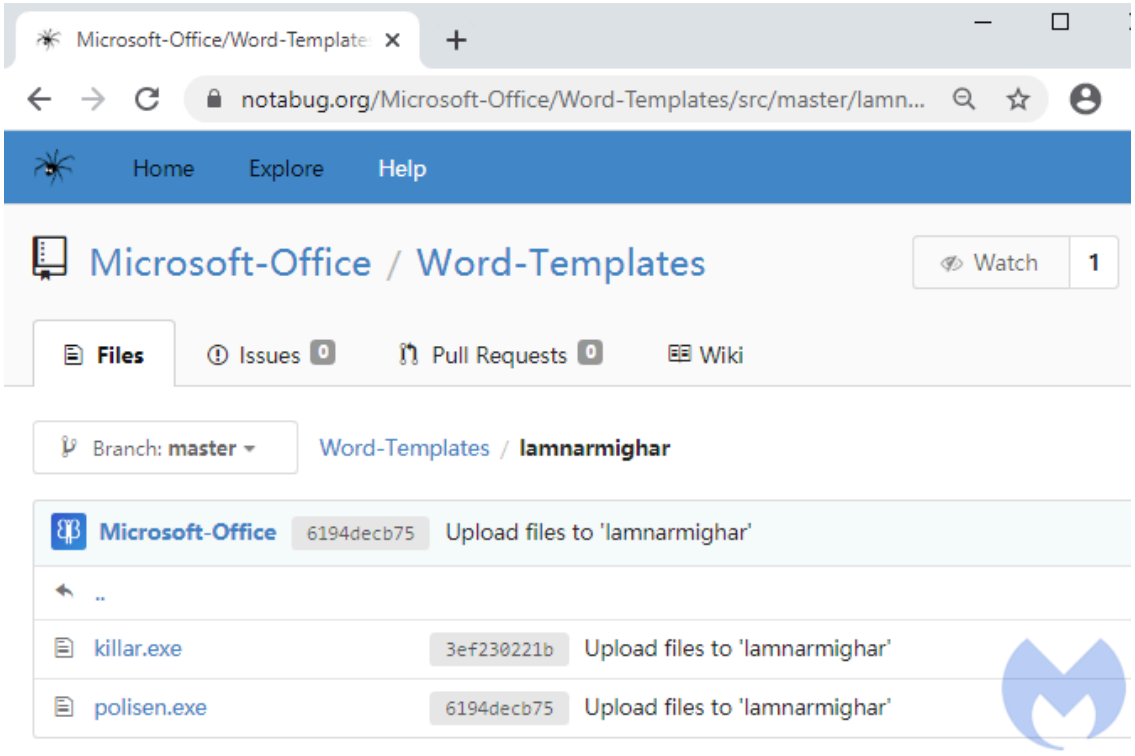
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



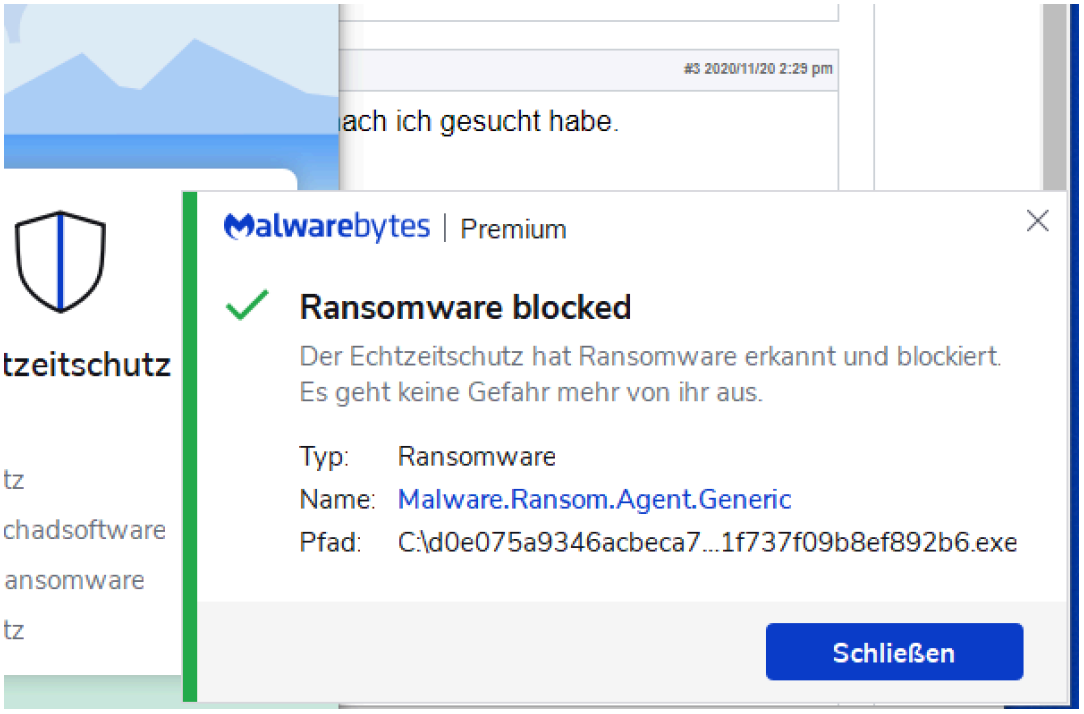
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

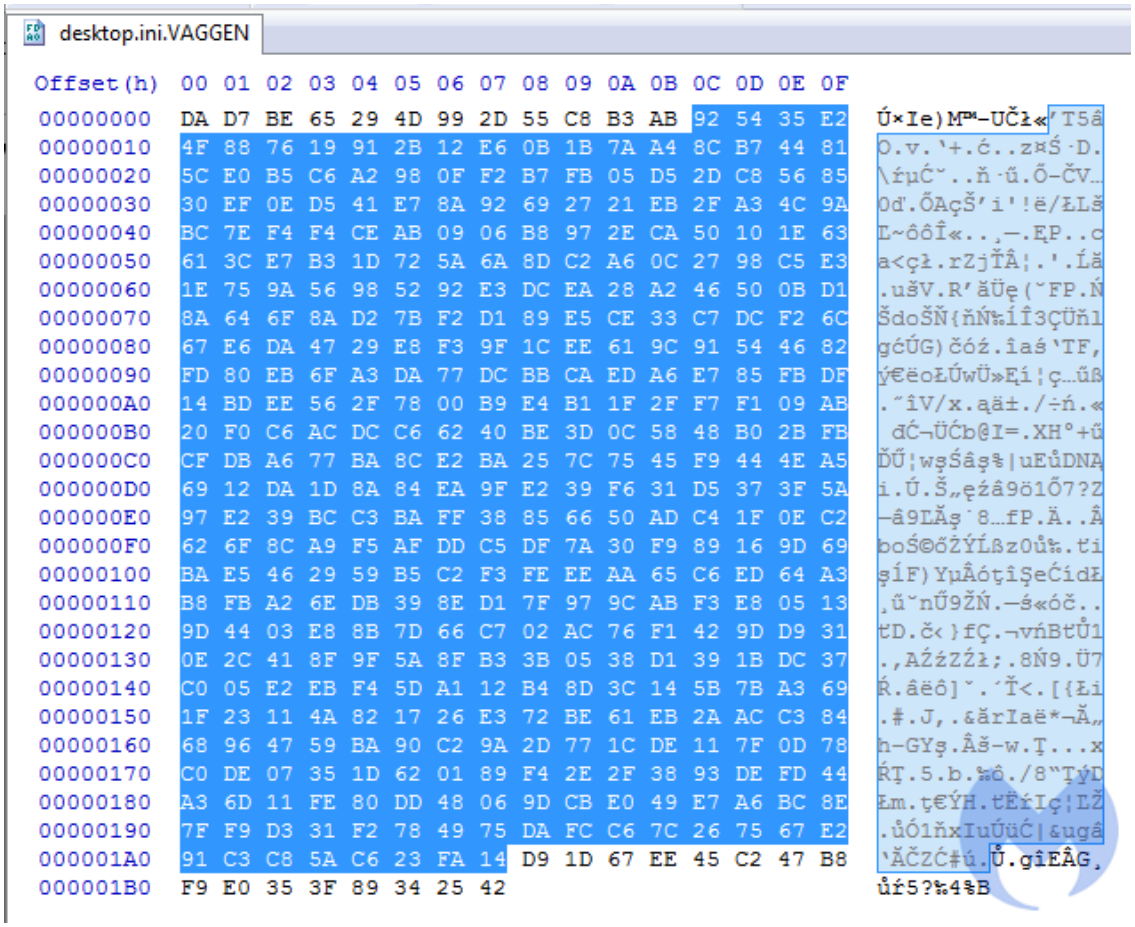
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

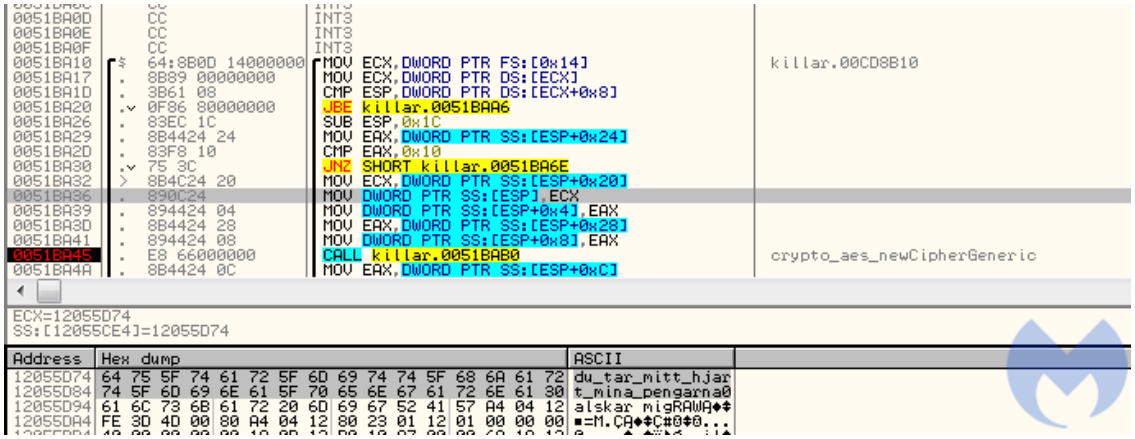


The content of the output file (with .VAGGEN extension) contains:

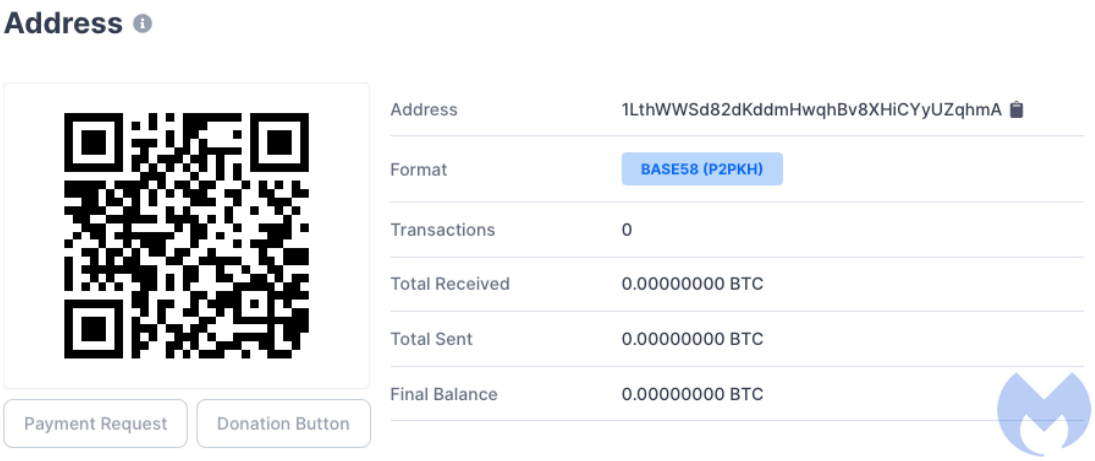
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

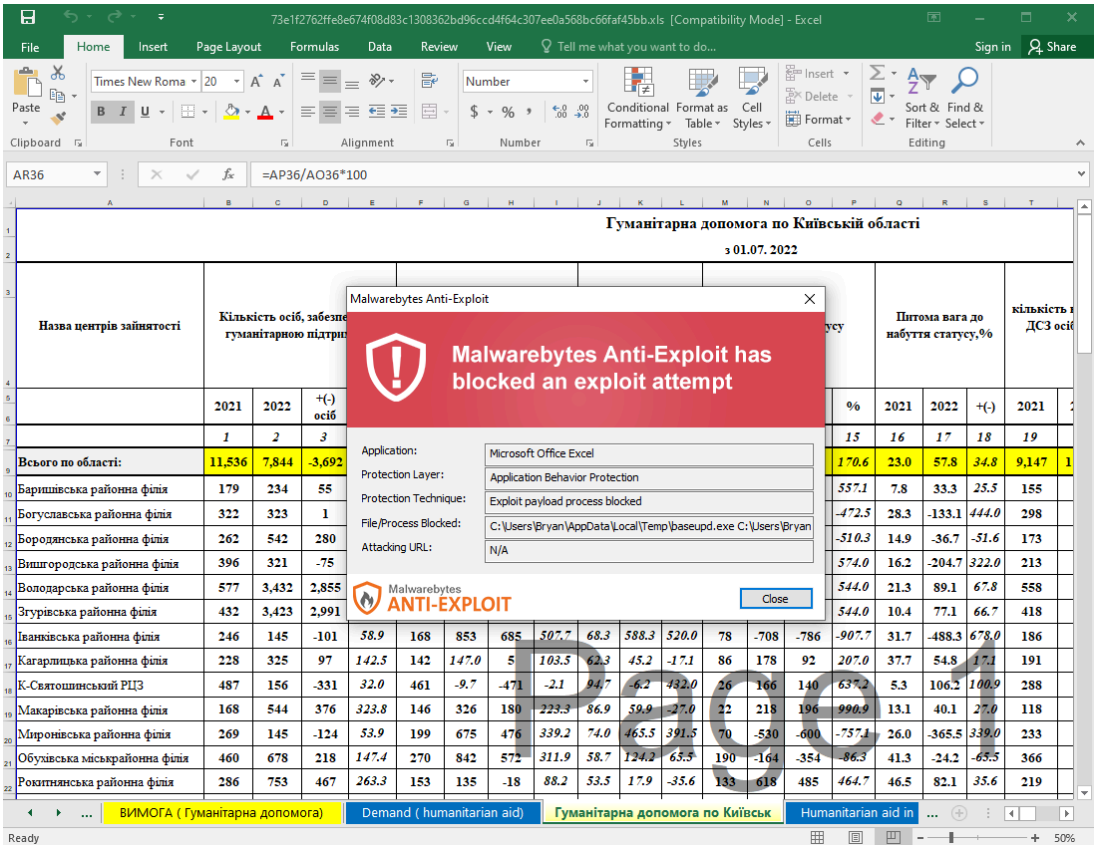
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

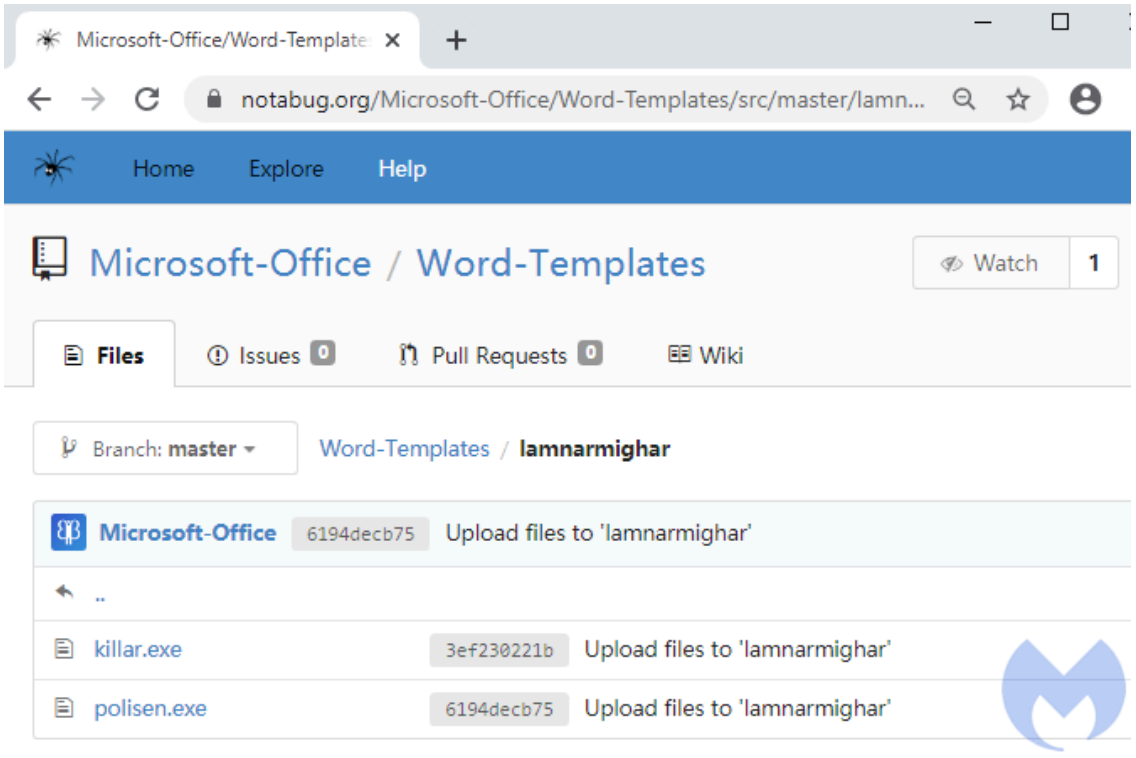
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



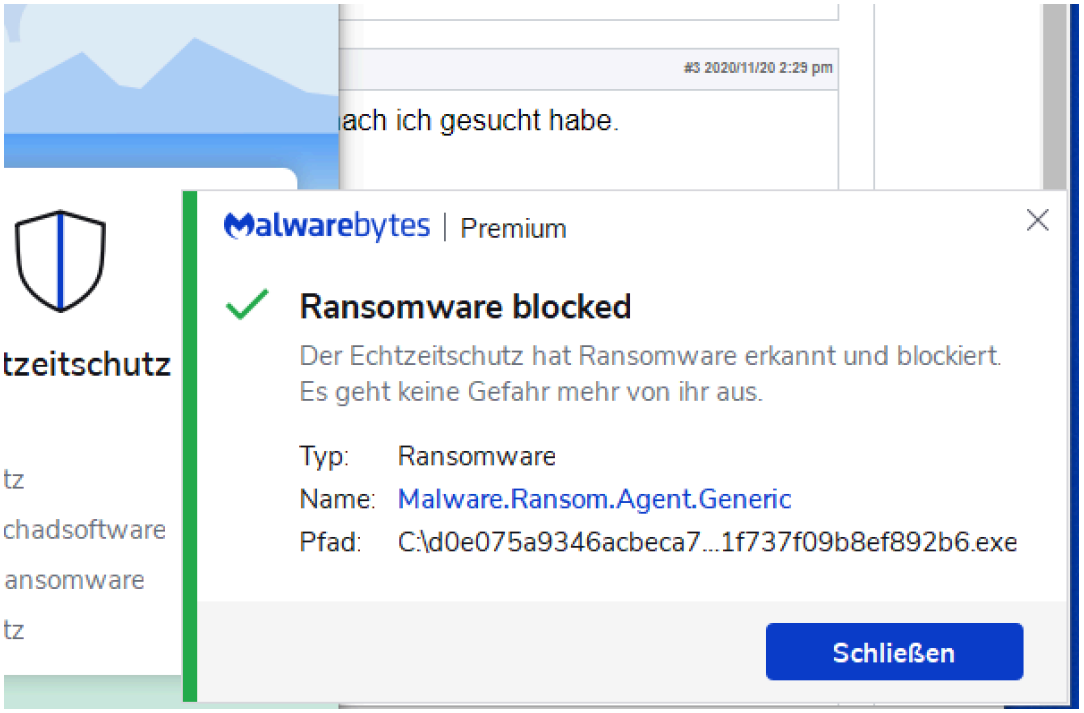
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

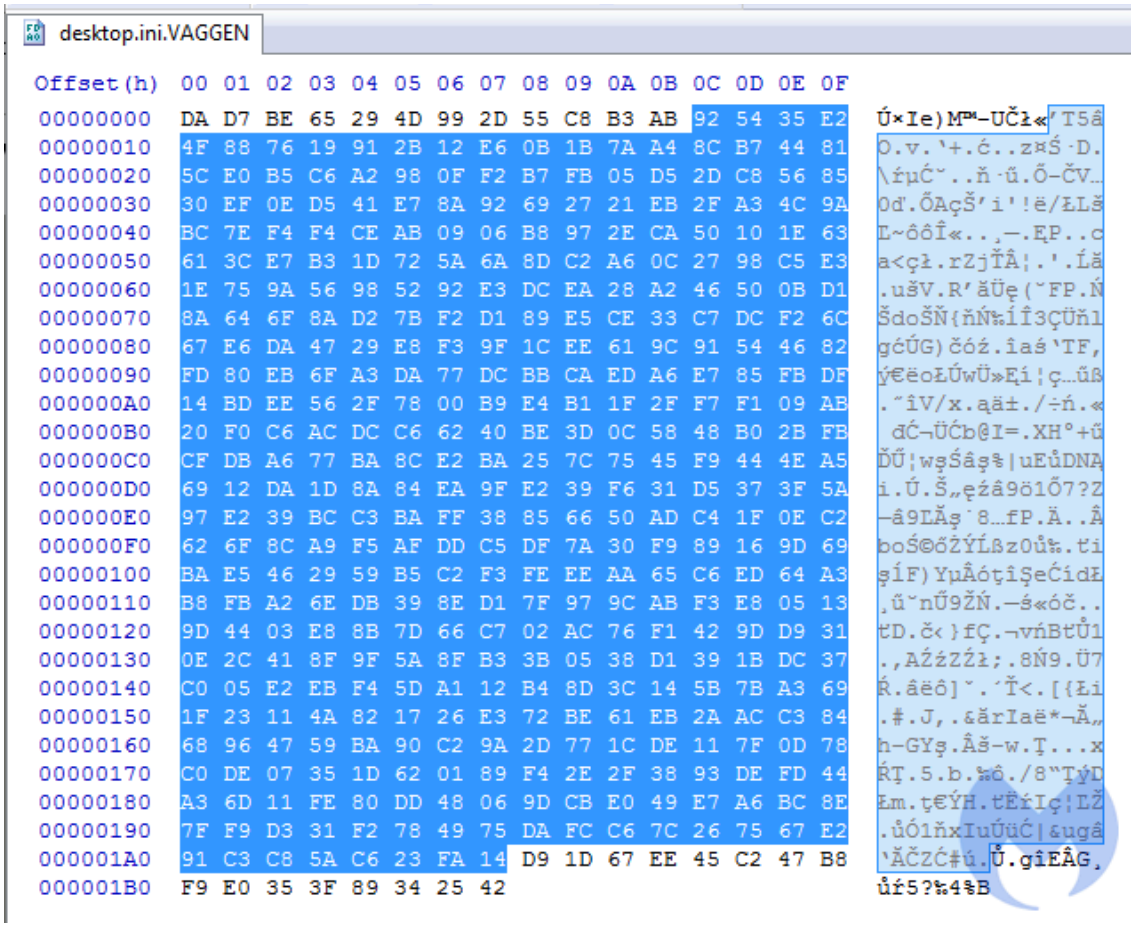
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

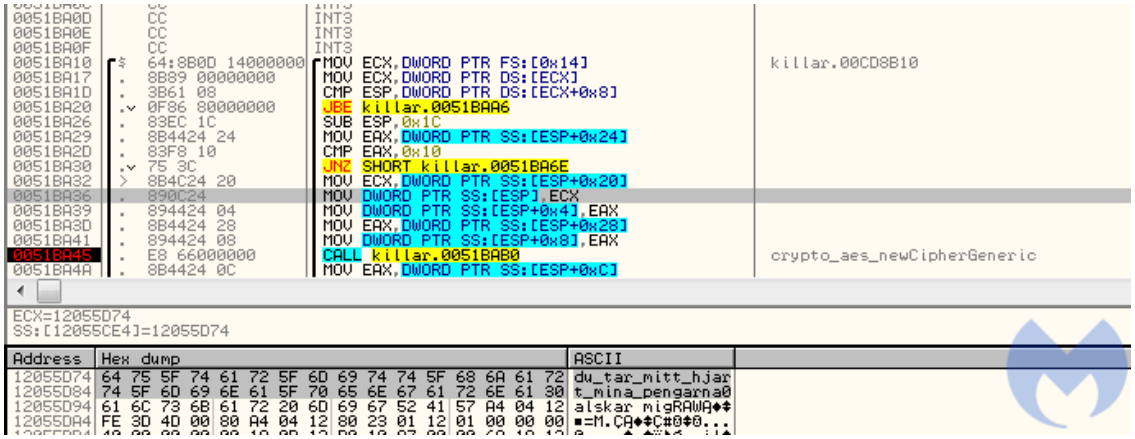


The content of the output file (with .VAGGEN extension) contains:

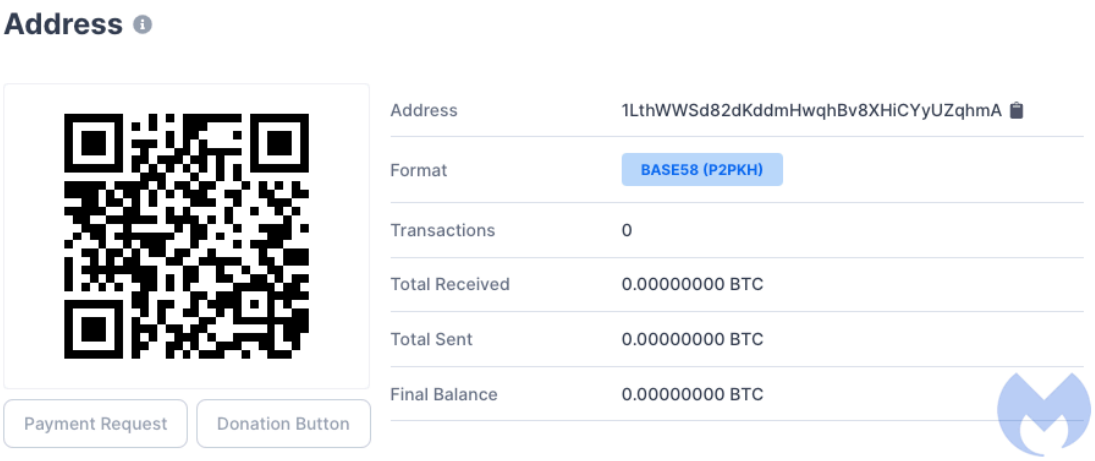
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

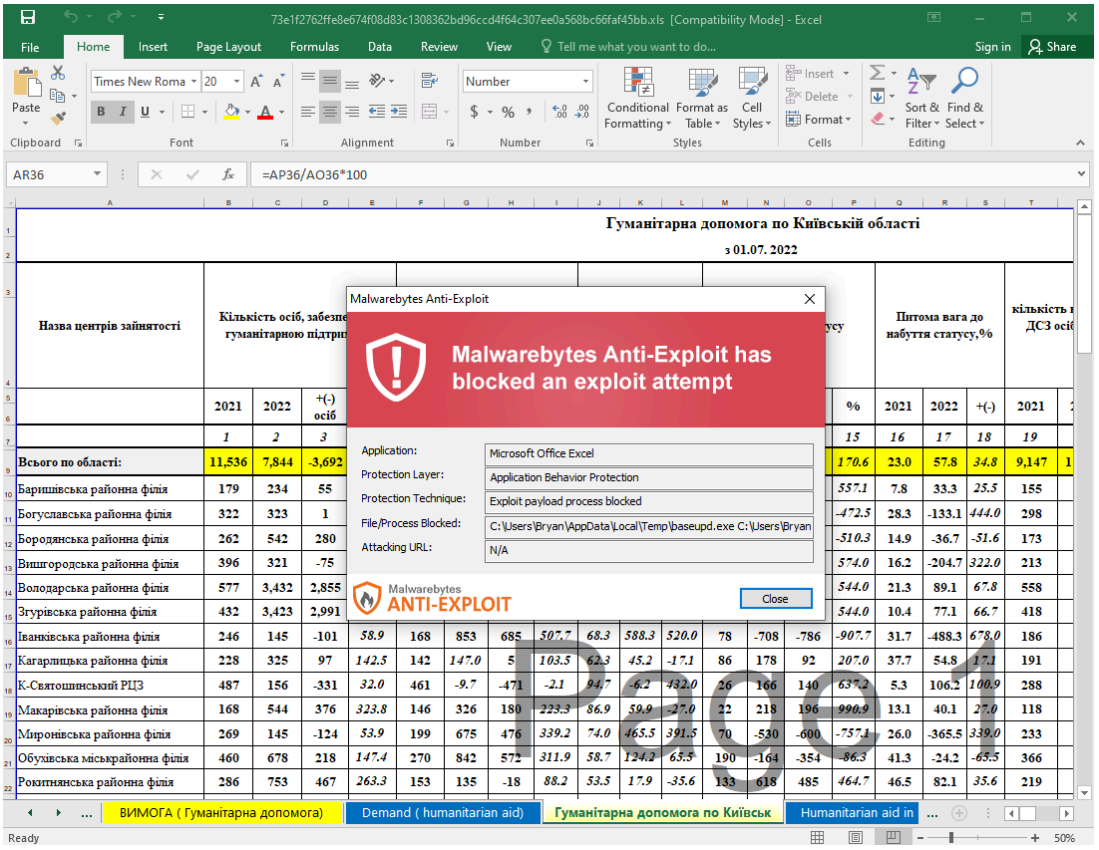
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

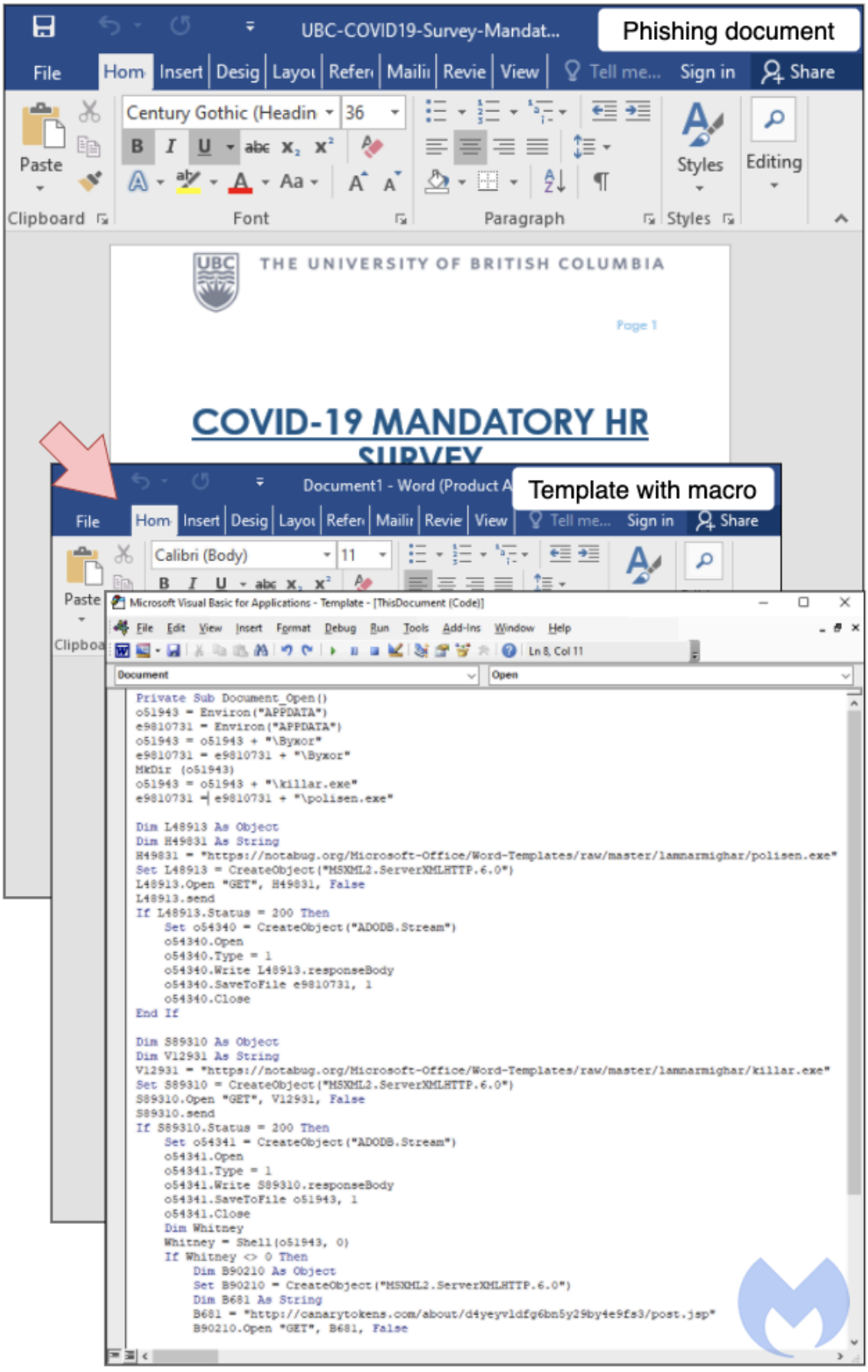
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

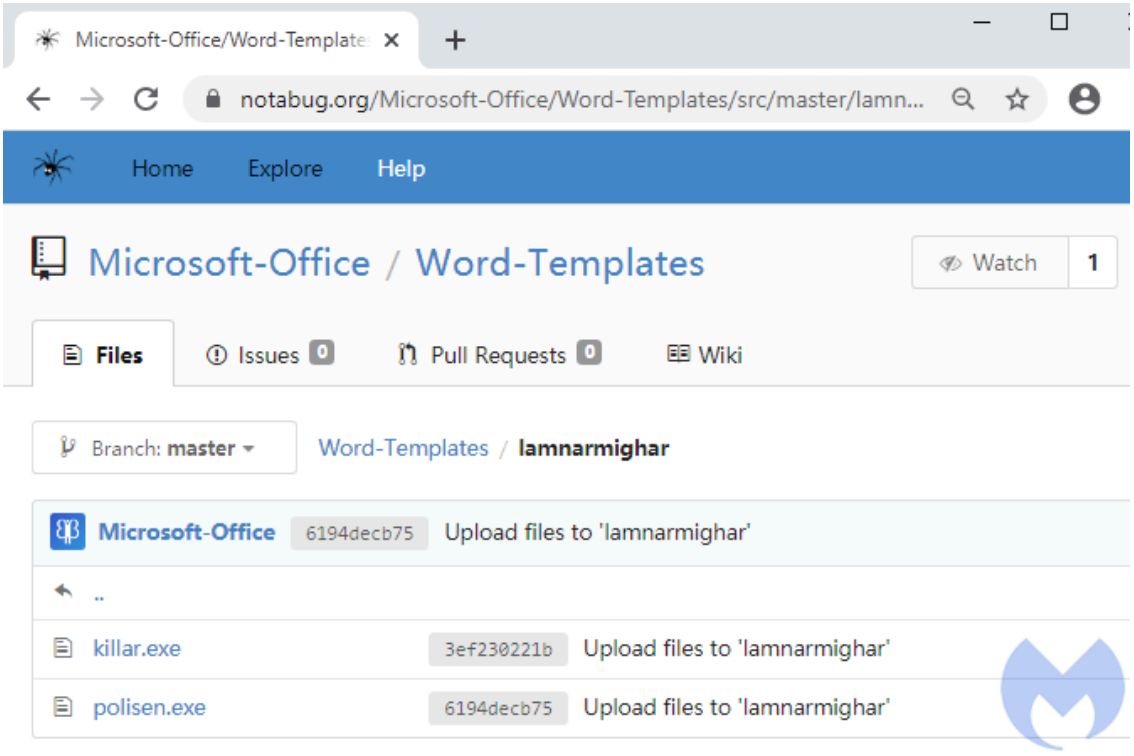
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



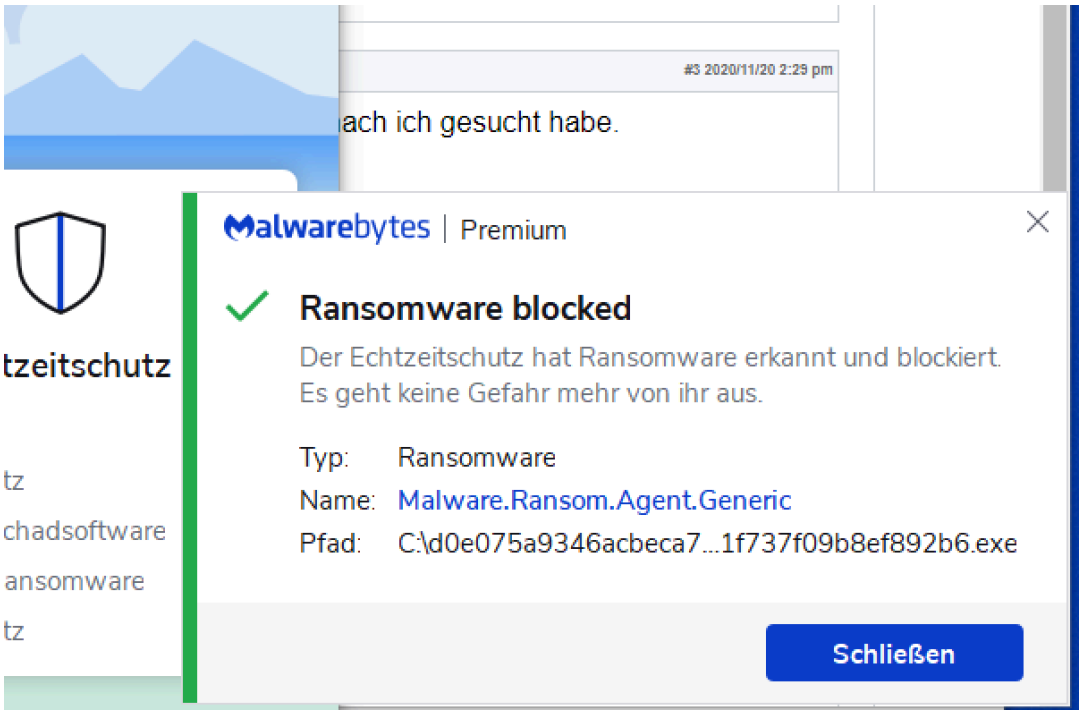
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

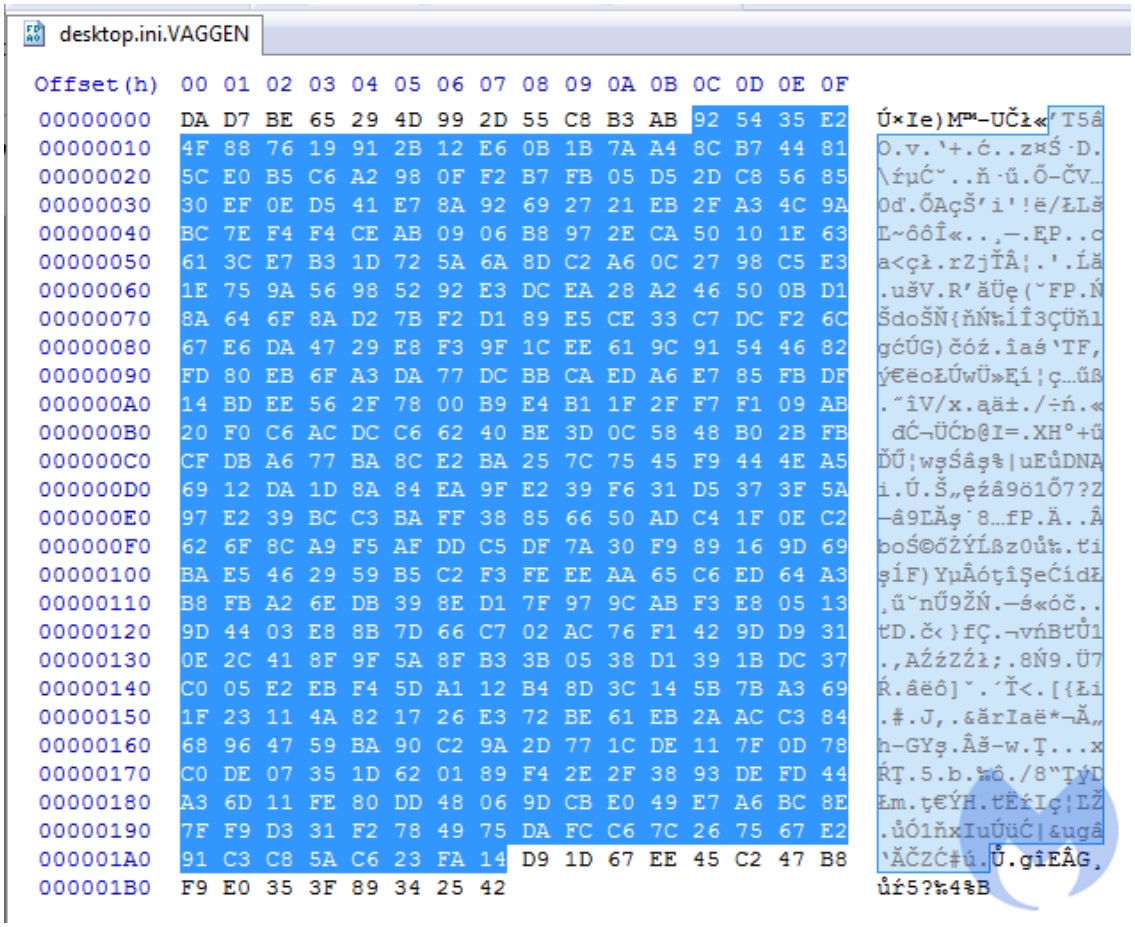
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

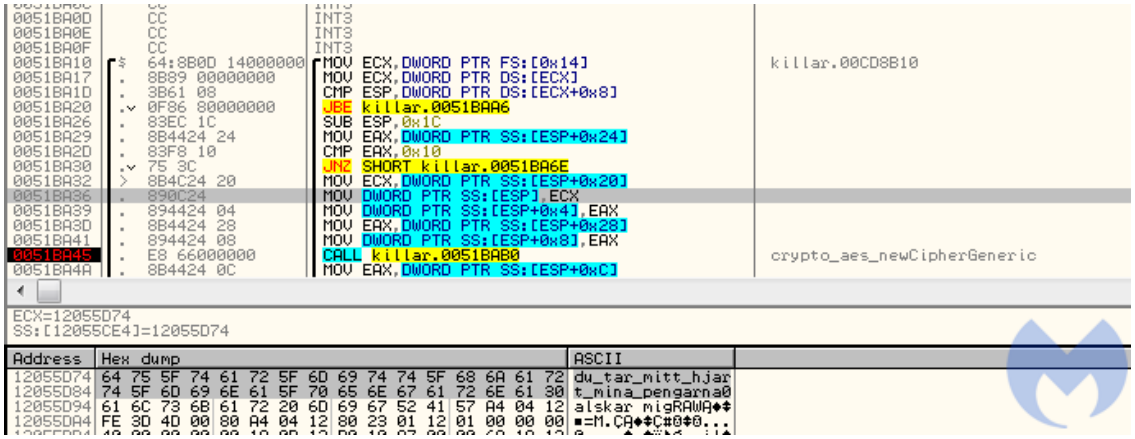


The content of the output file (with .VAGGEN extension) contains:

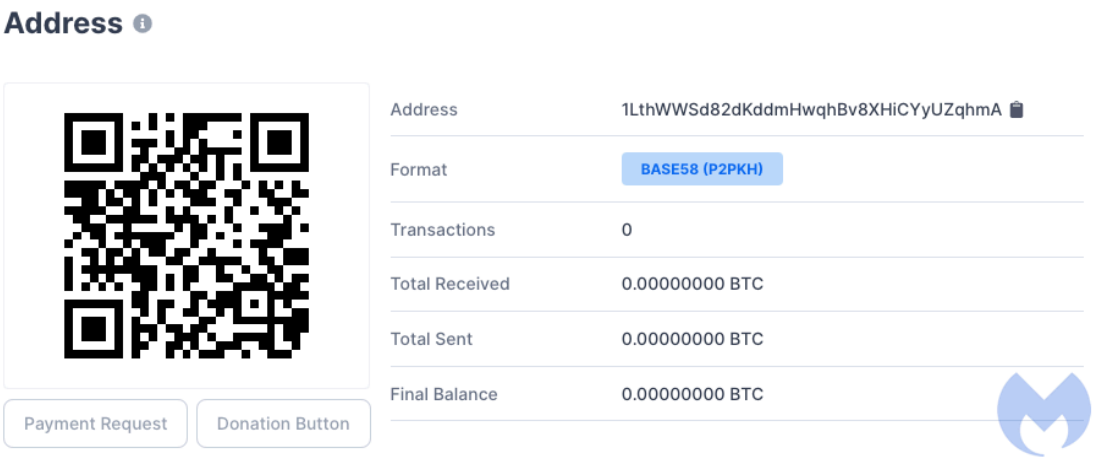
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

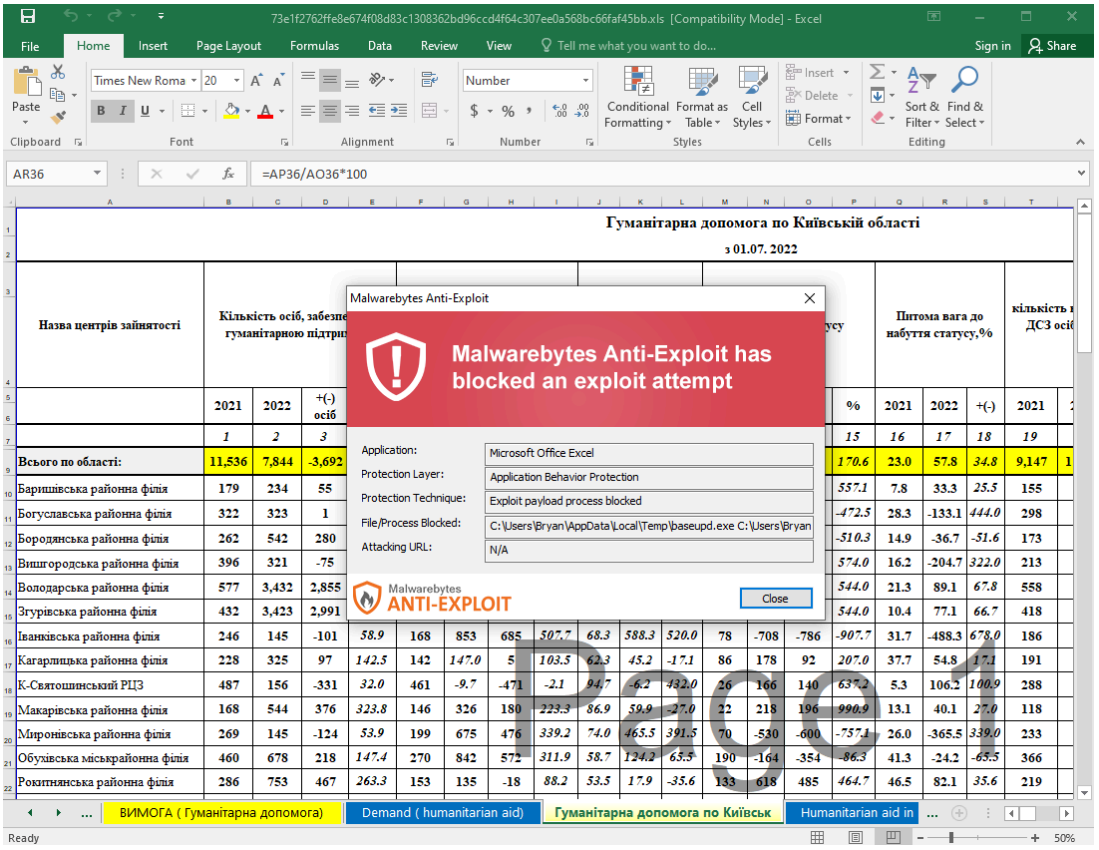
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

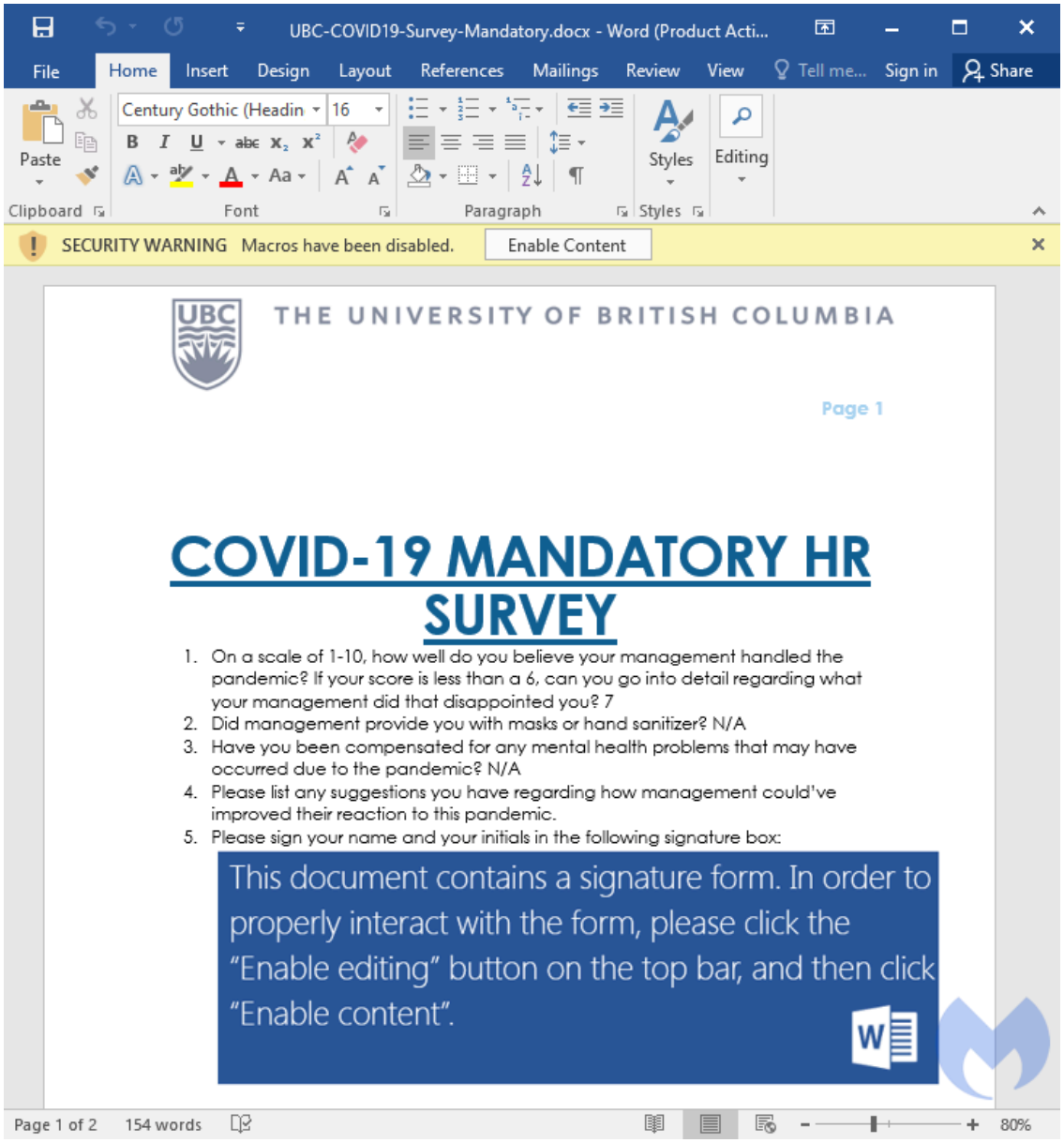
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

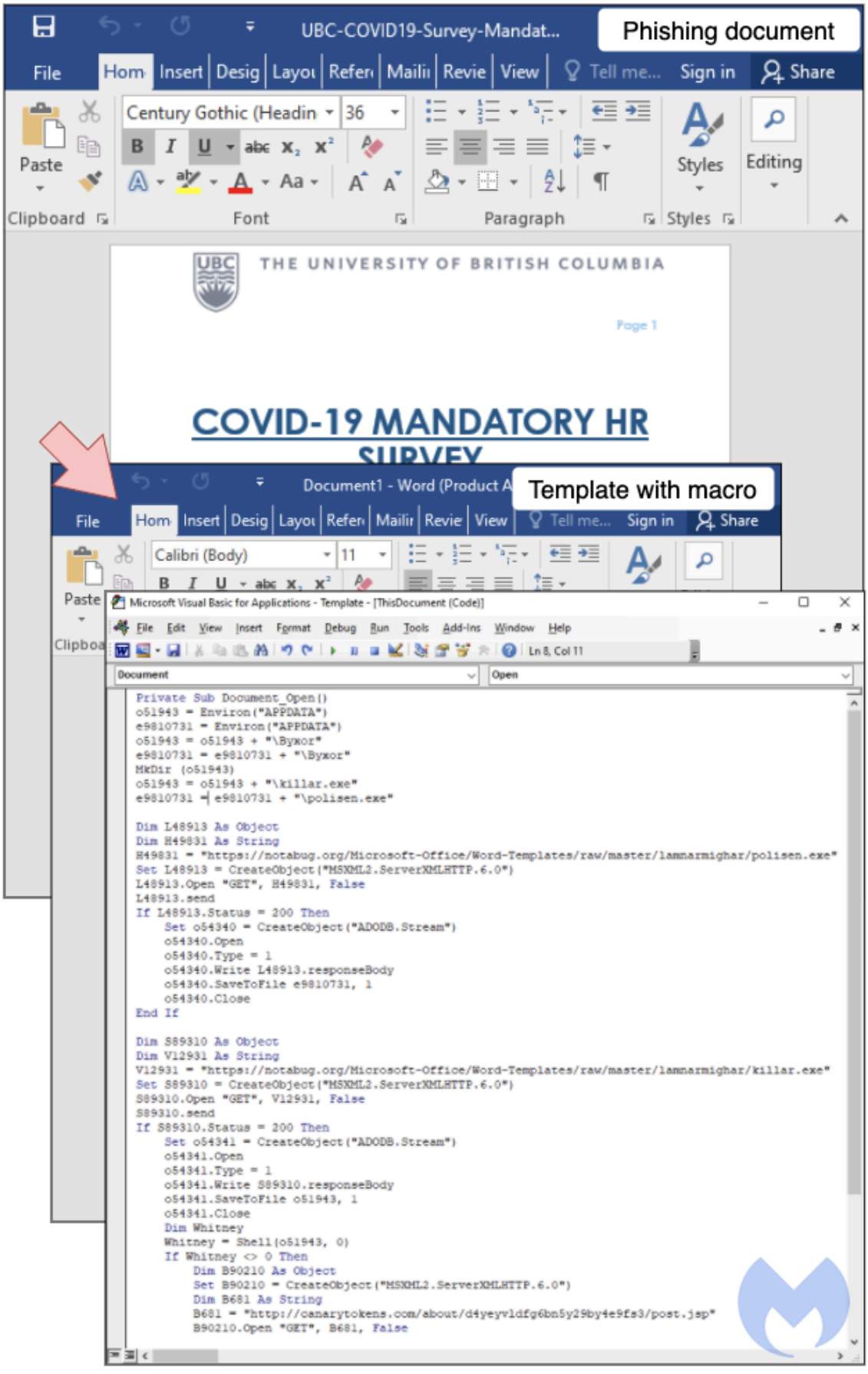
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

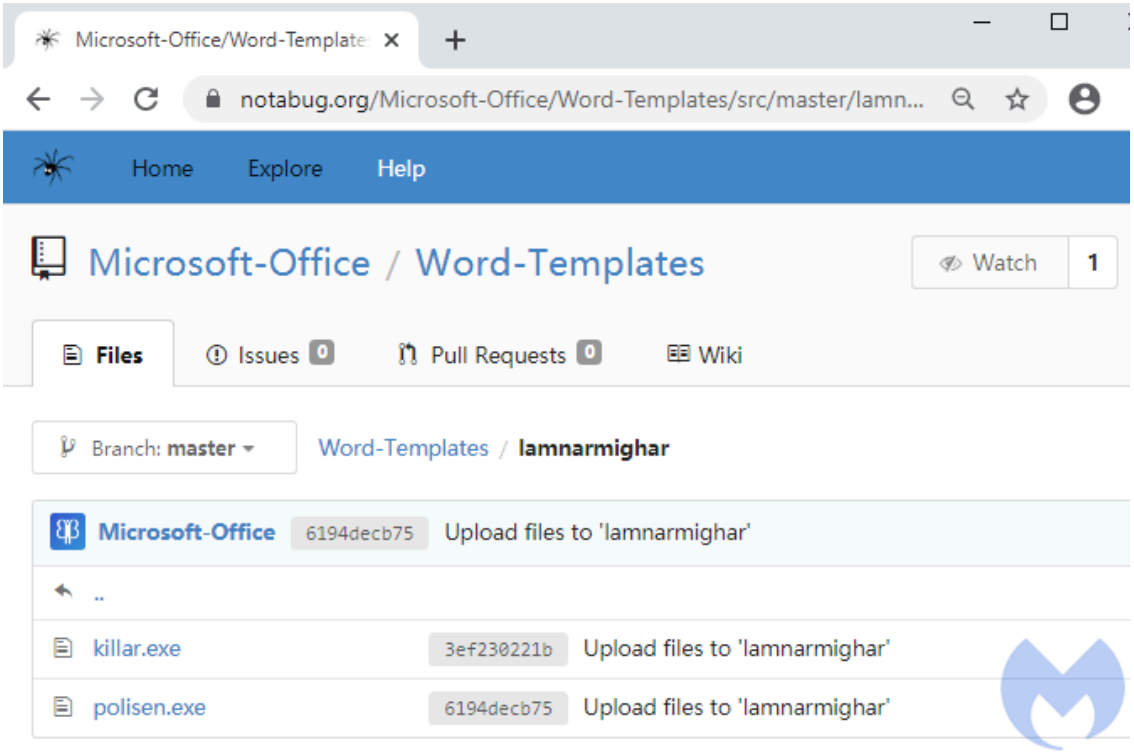
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



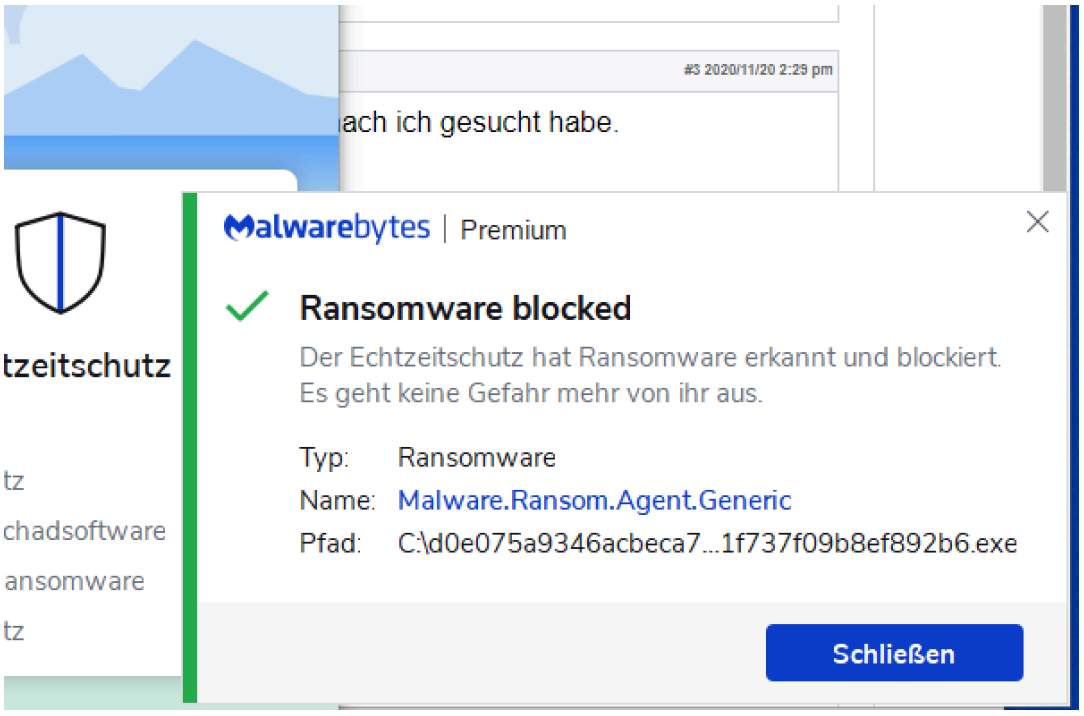
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

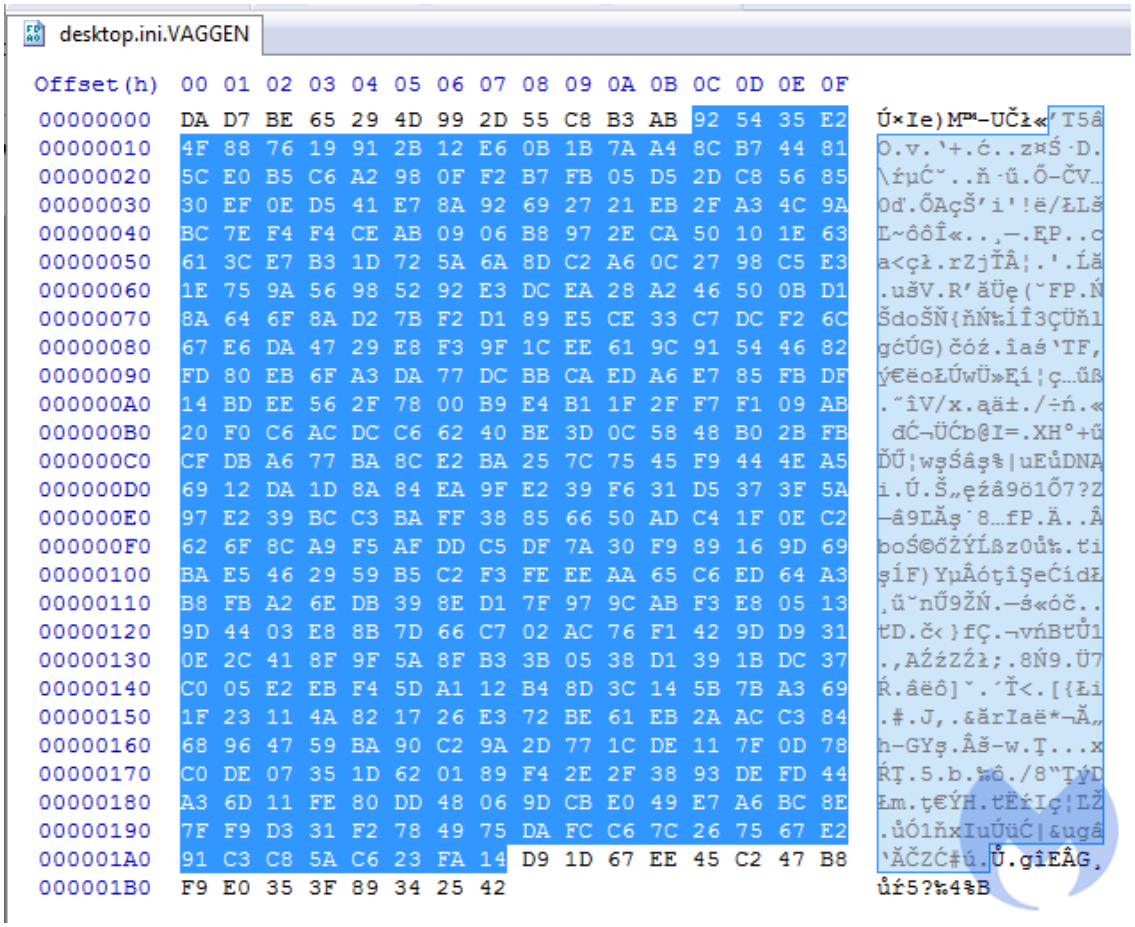
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

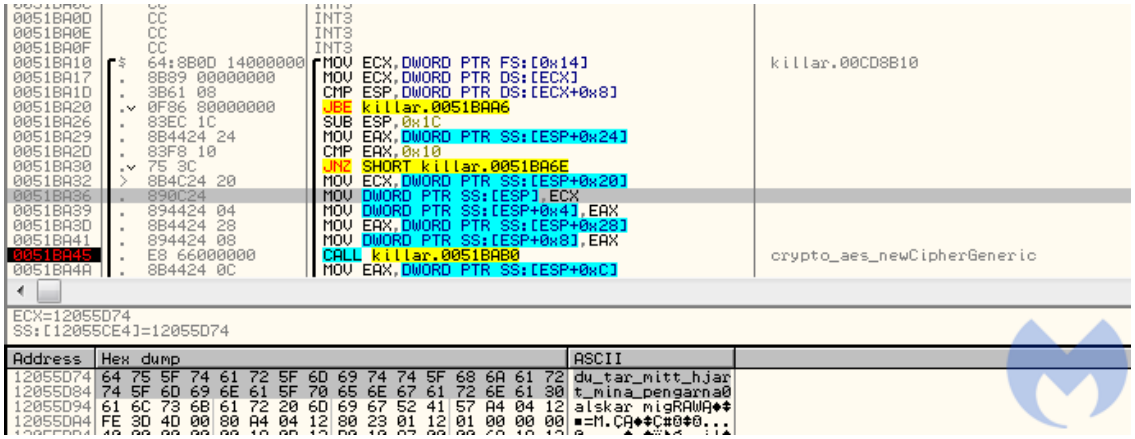


The content of the output file (with .VAGGEN extension) contains:

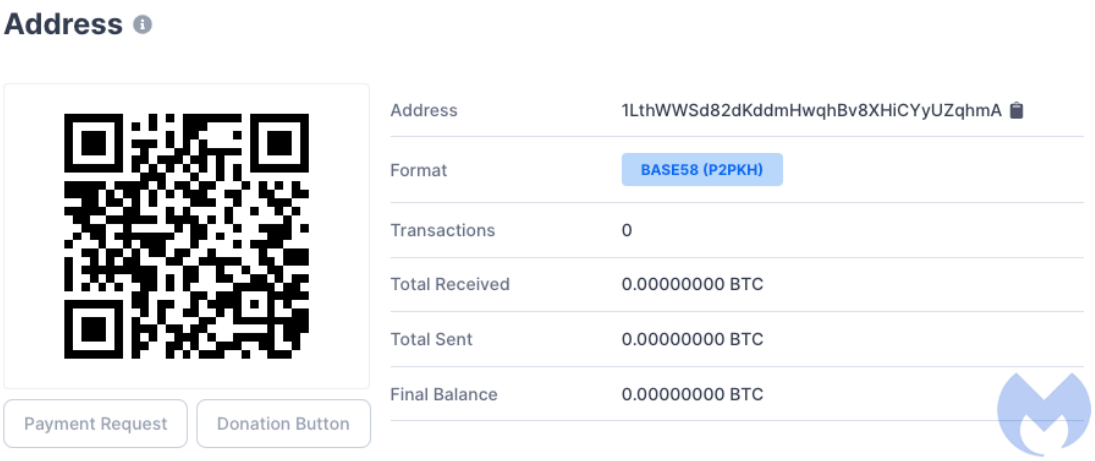
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

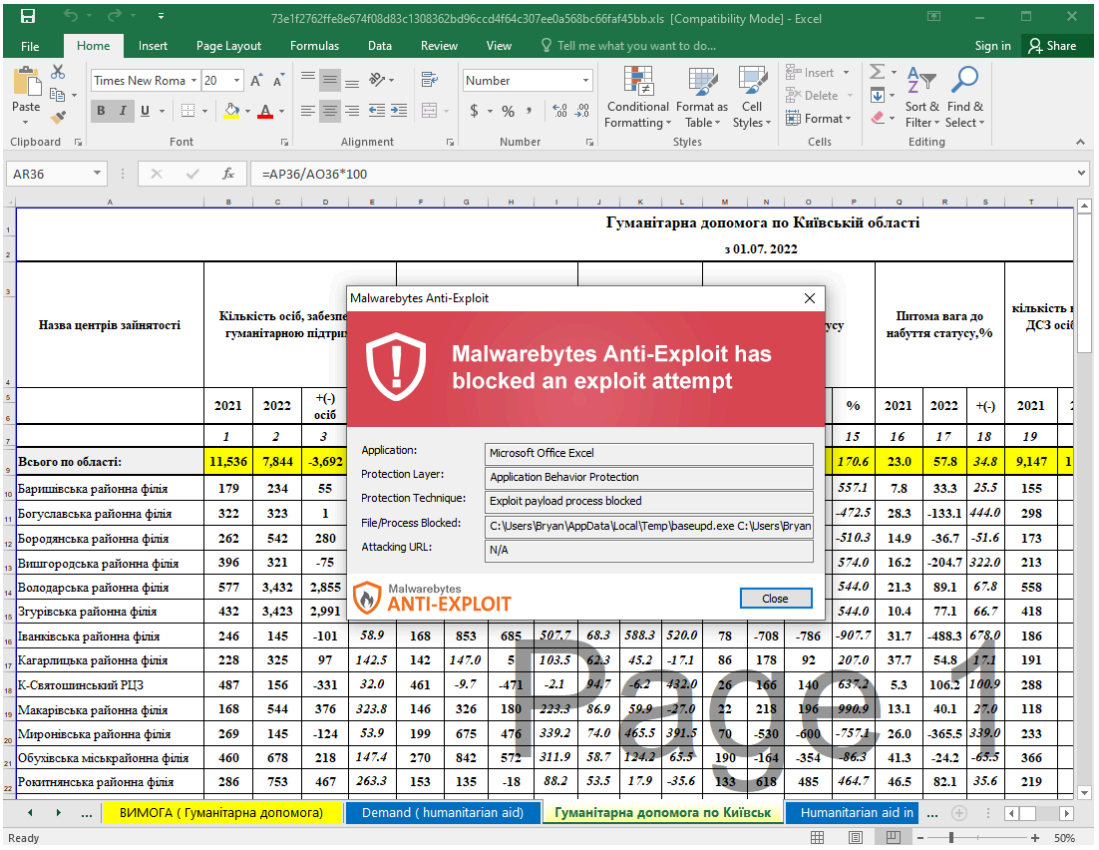
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

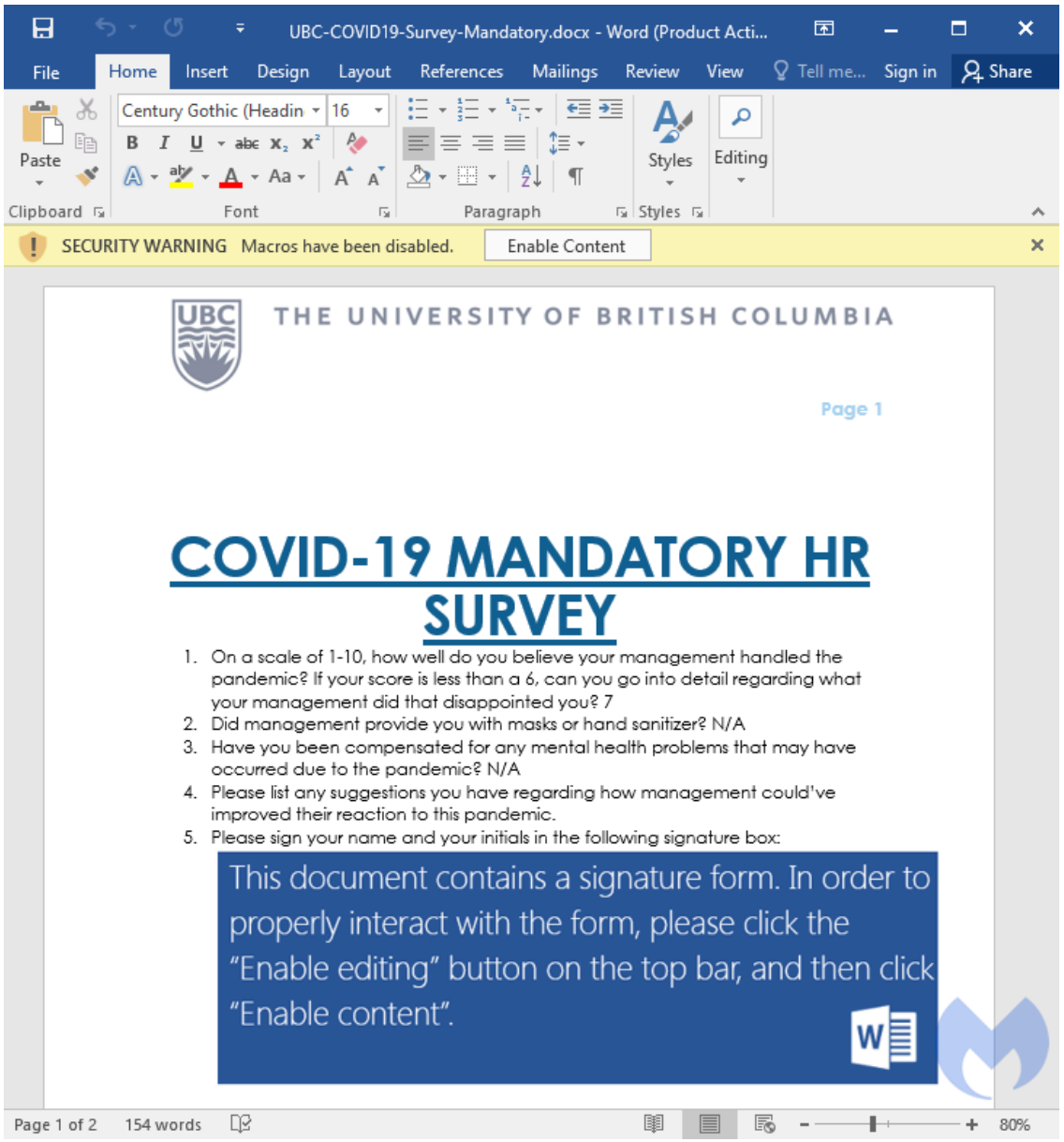
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

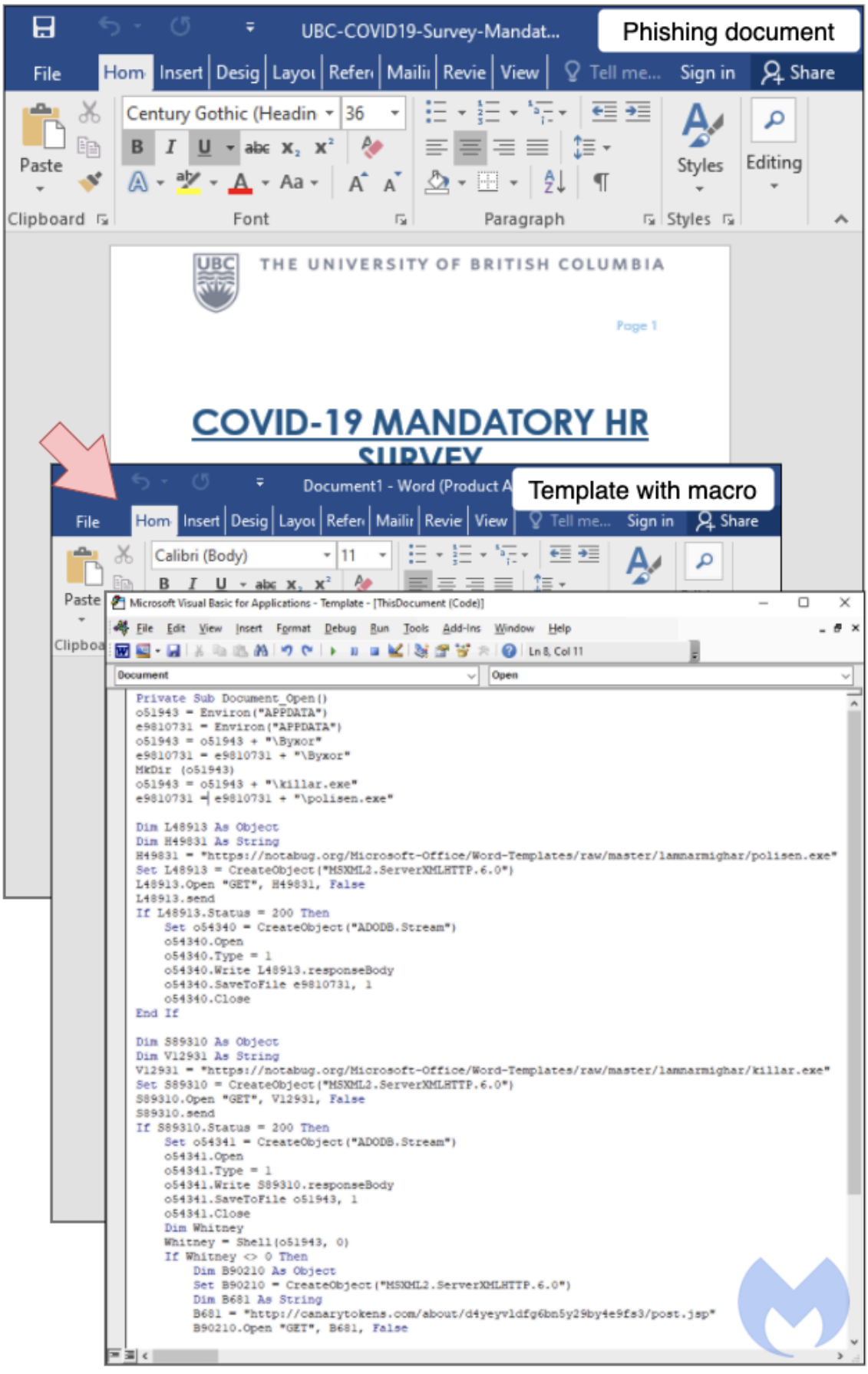
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

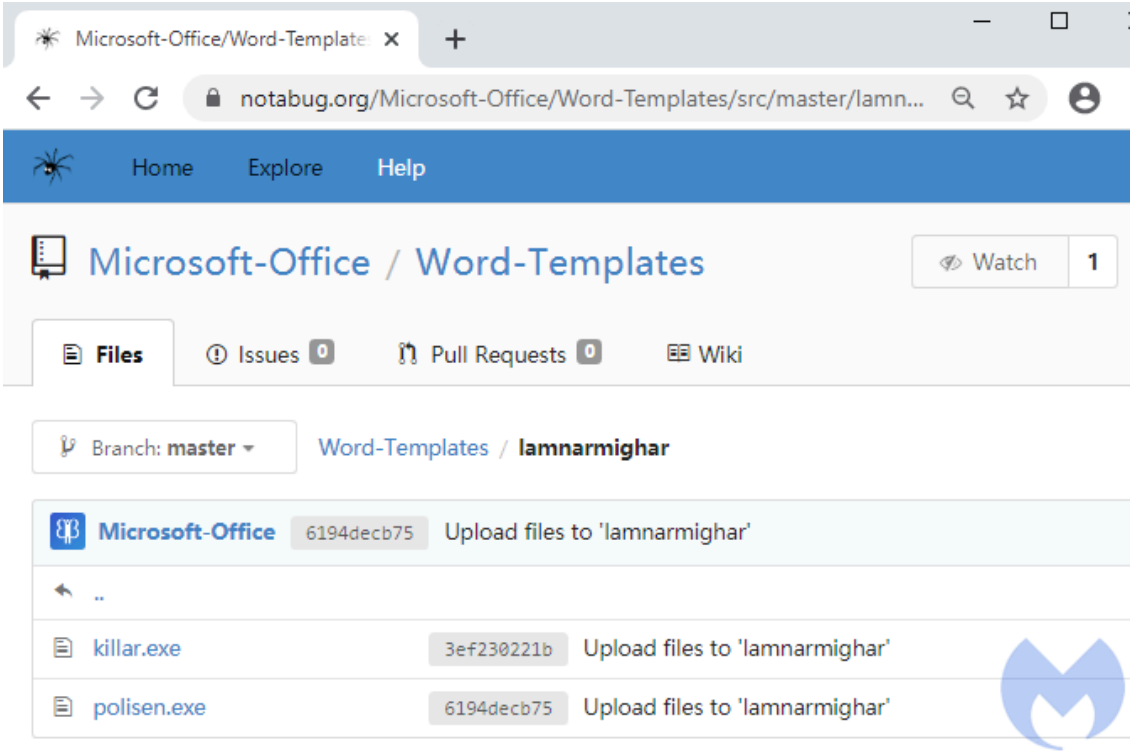
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



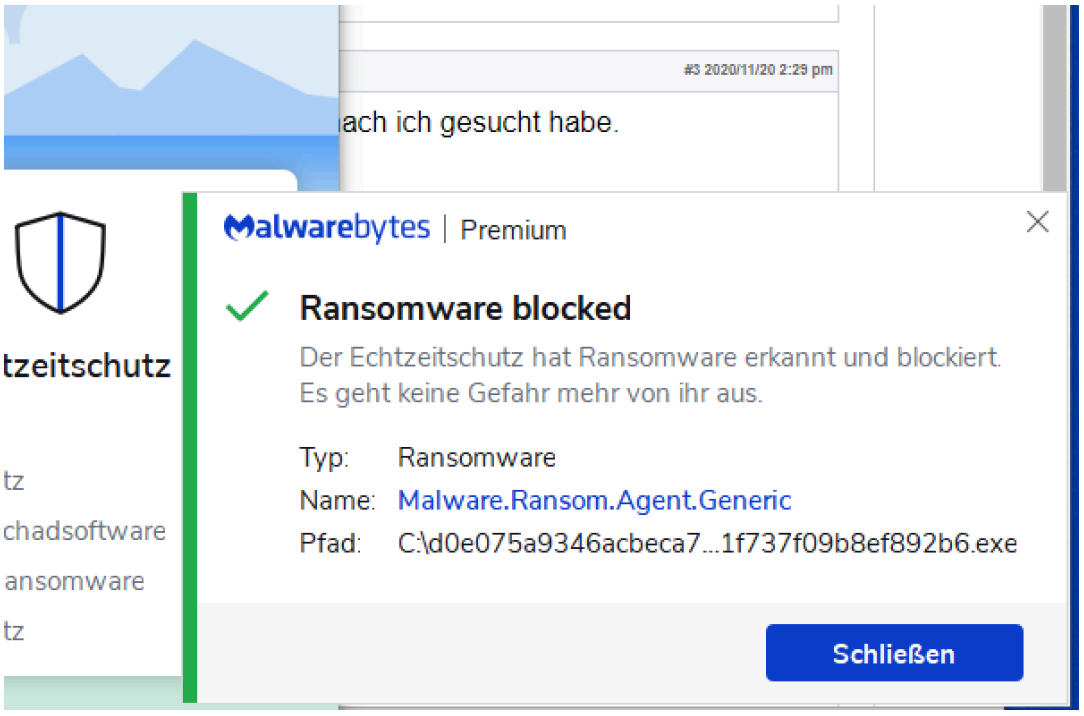
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

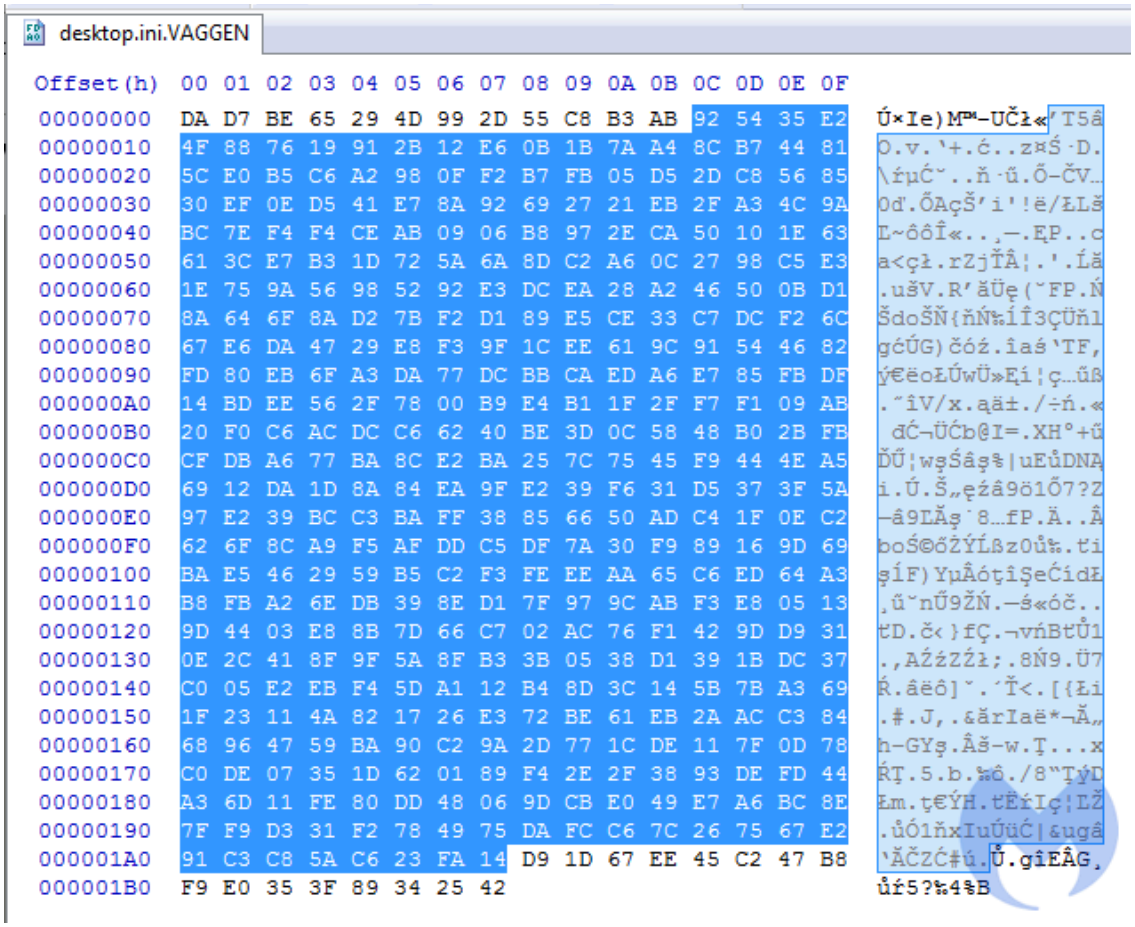
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



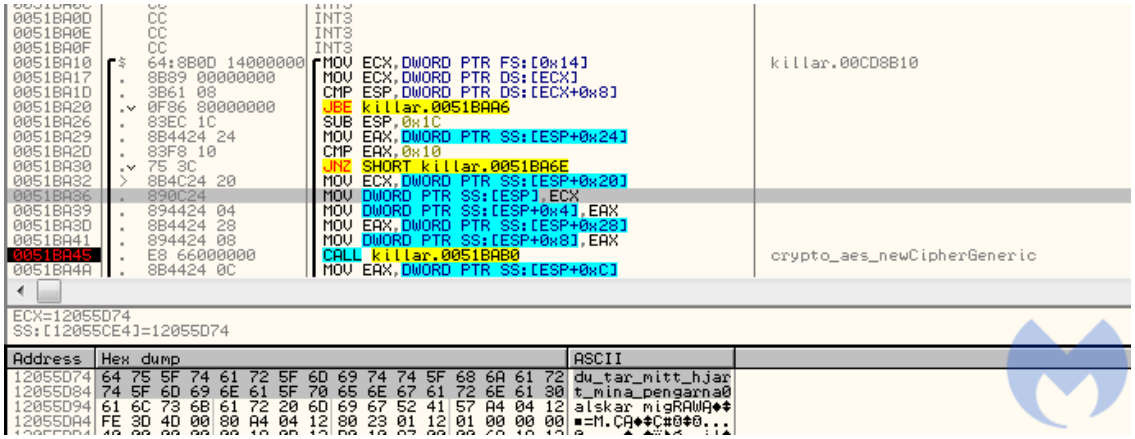
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

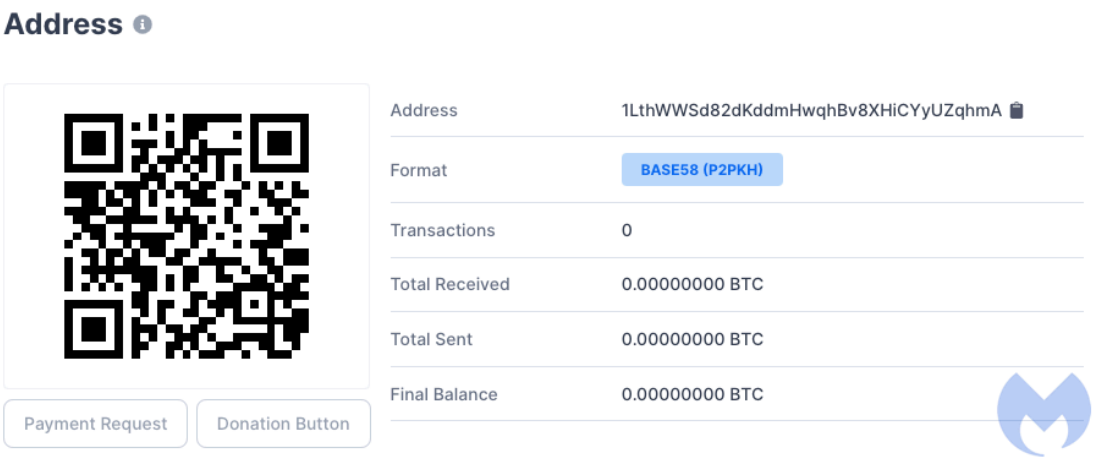
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

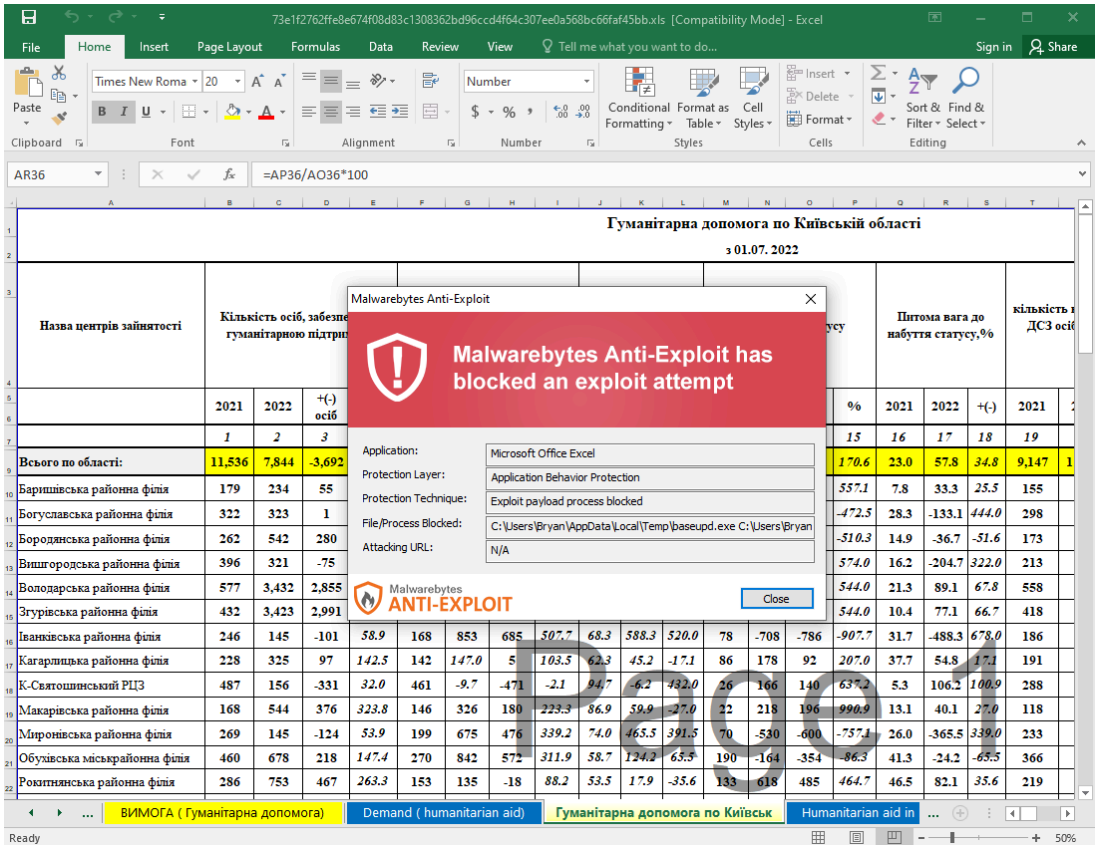
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

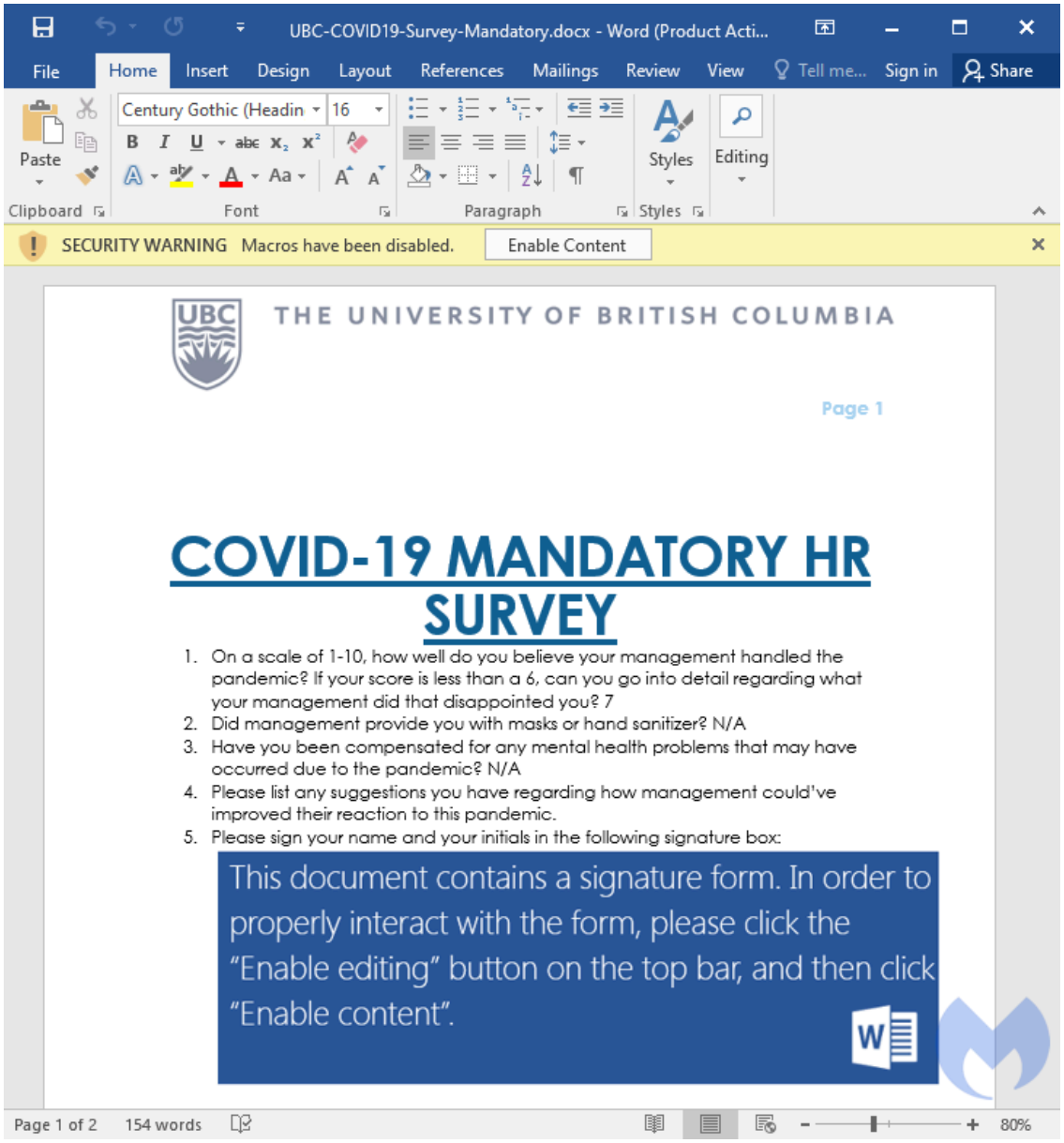
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

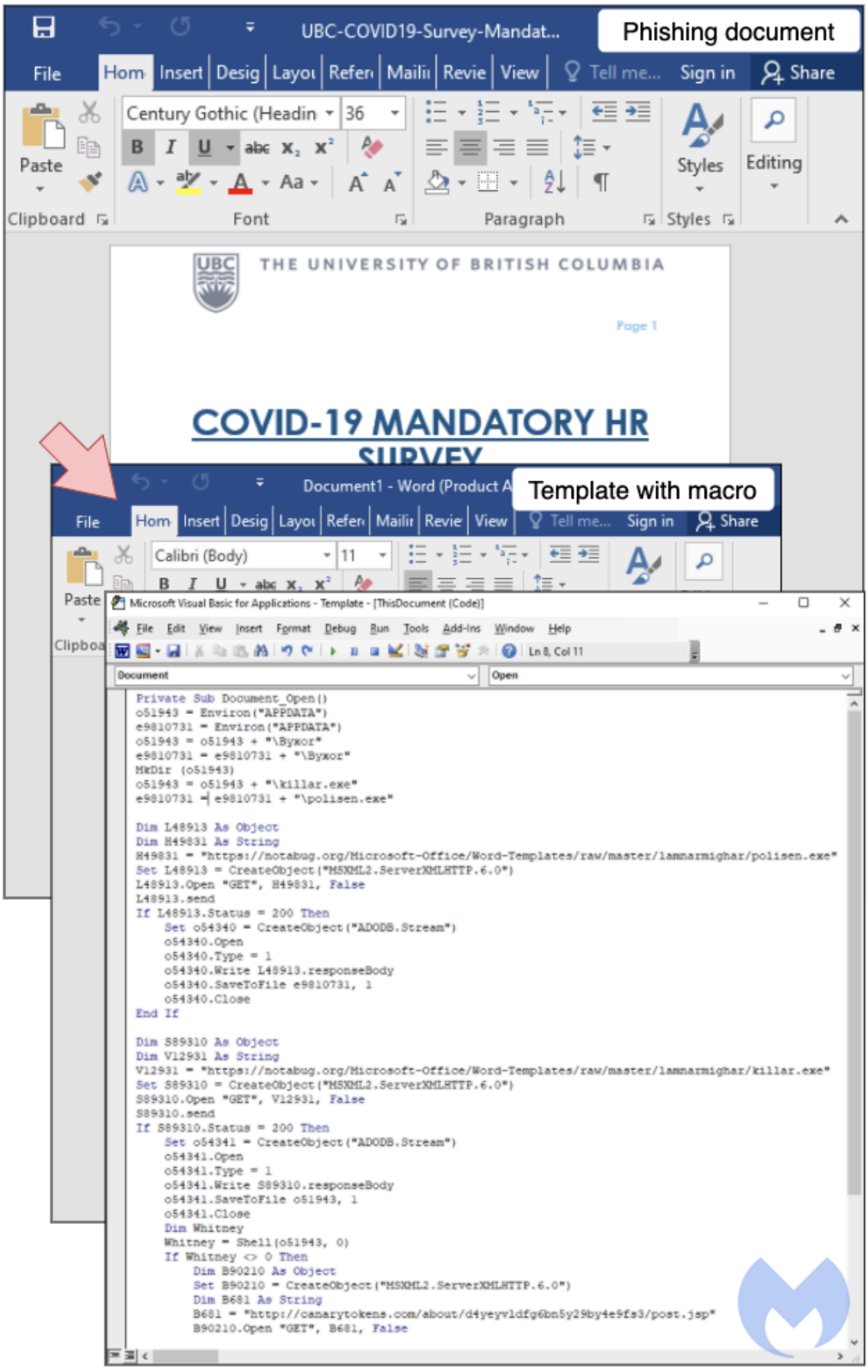
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

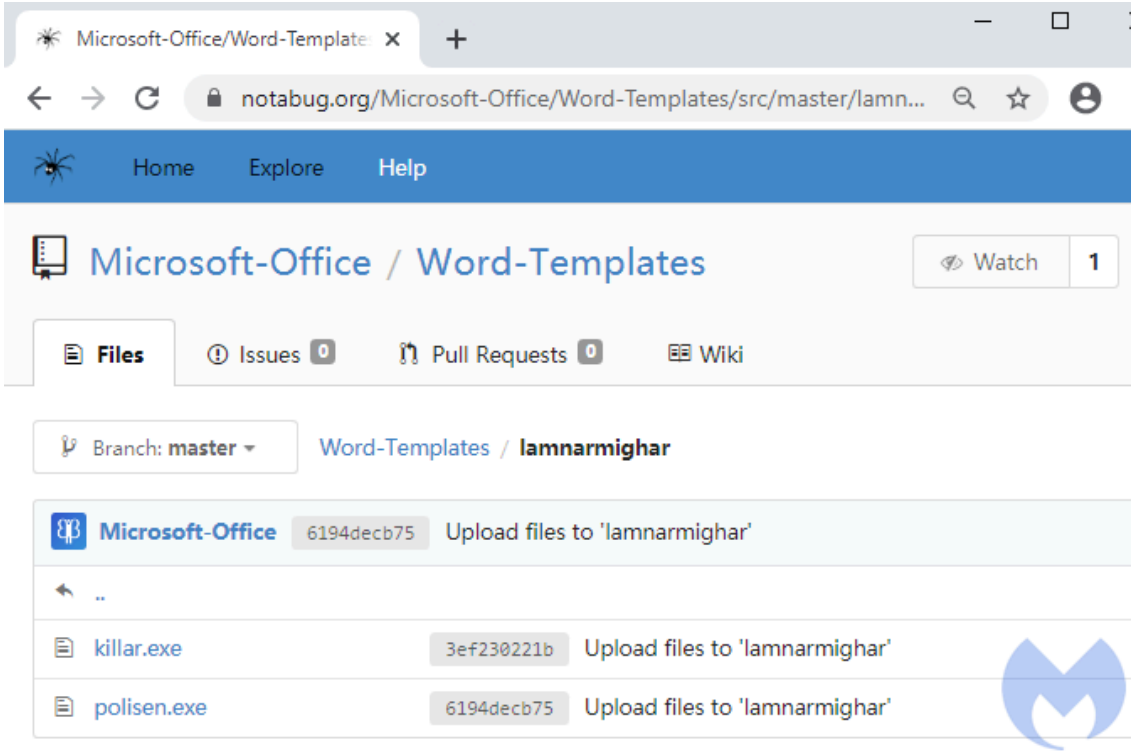
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



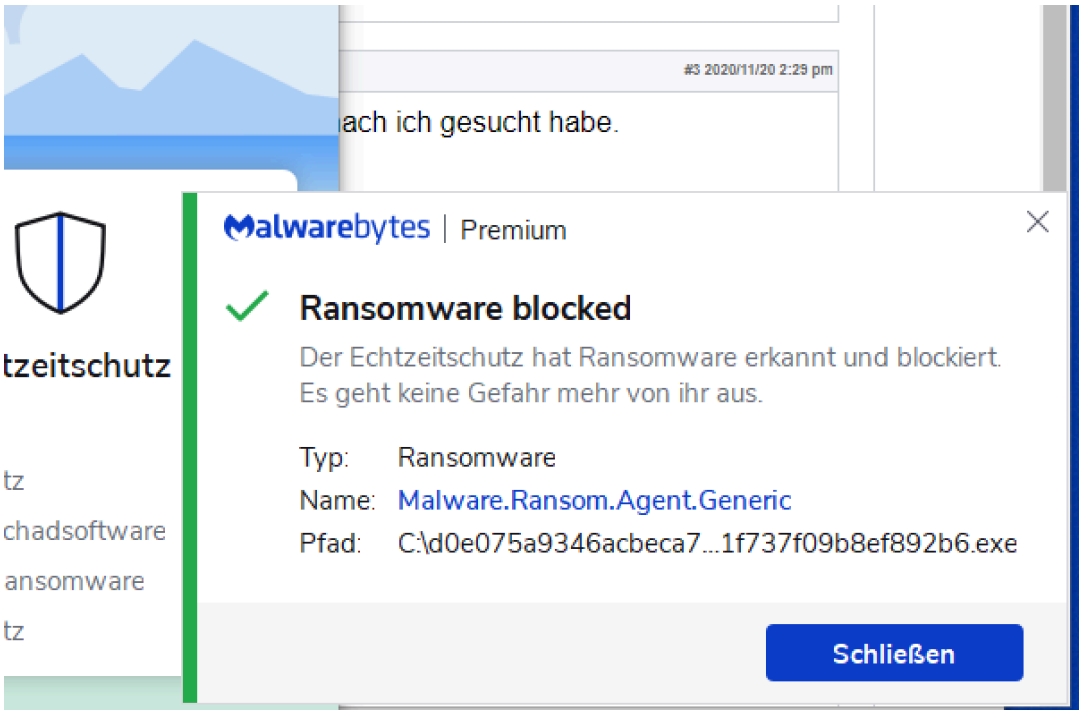
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

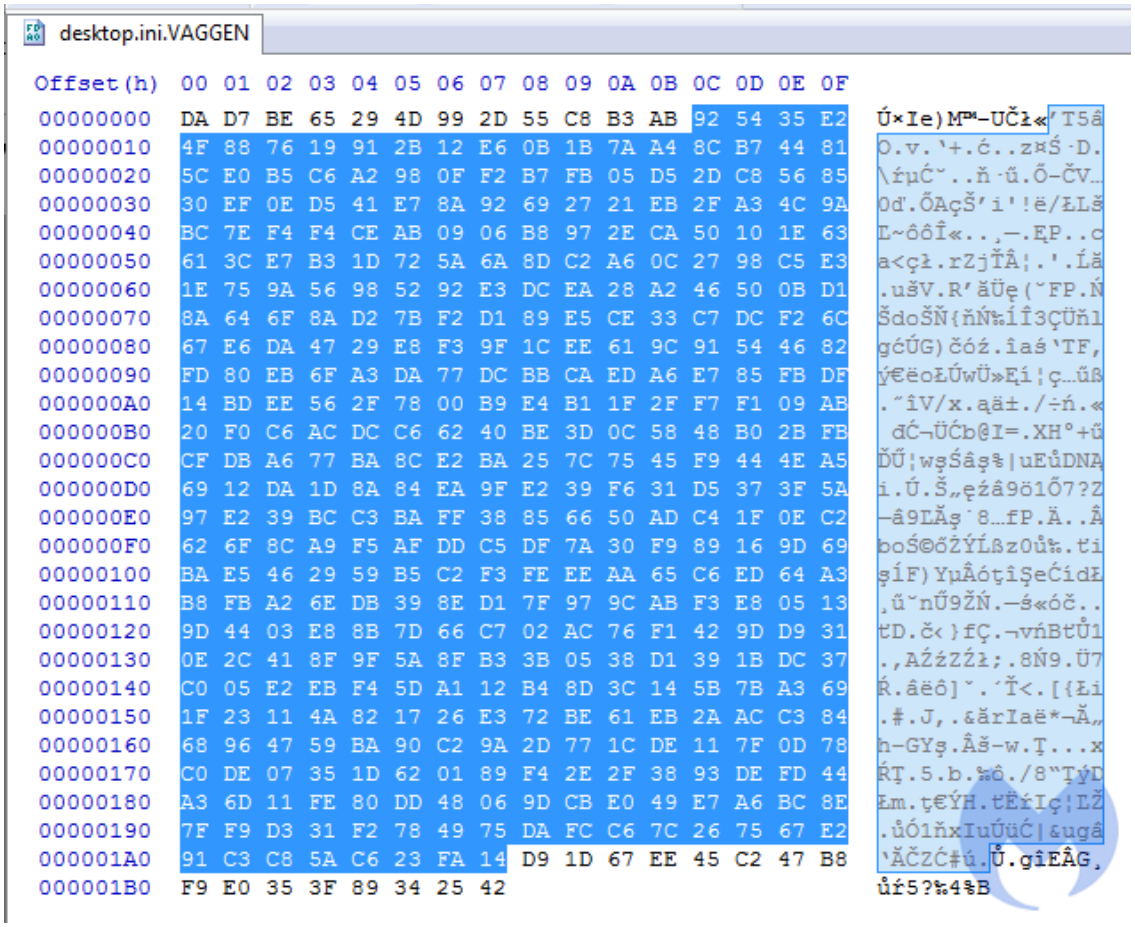
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

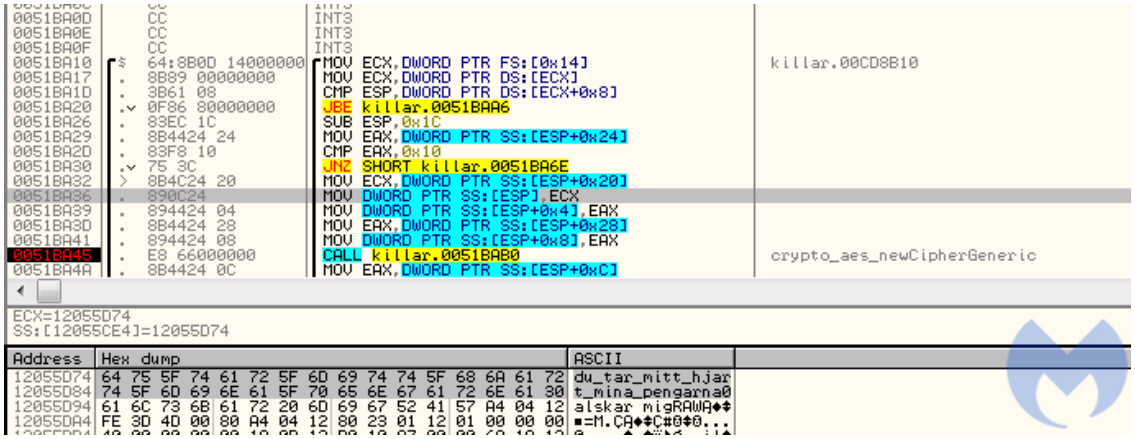


The content of the output file (with .VAGGEN extension) contains:

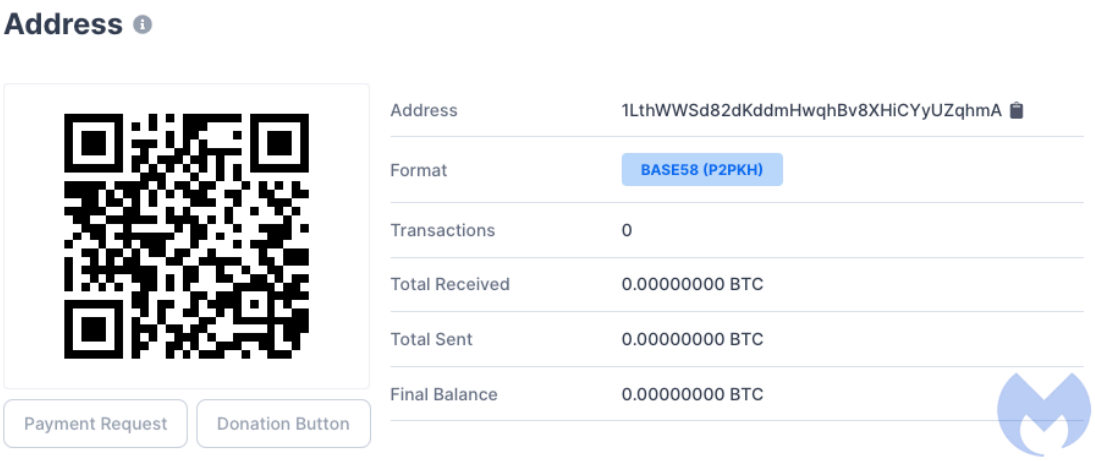
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

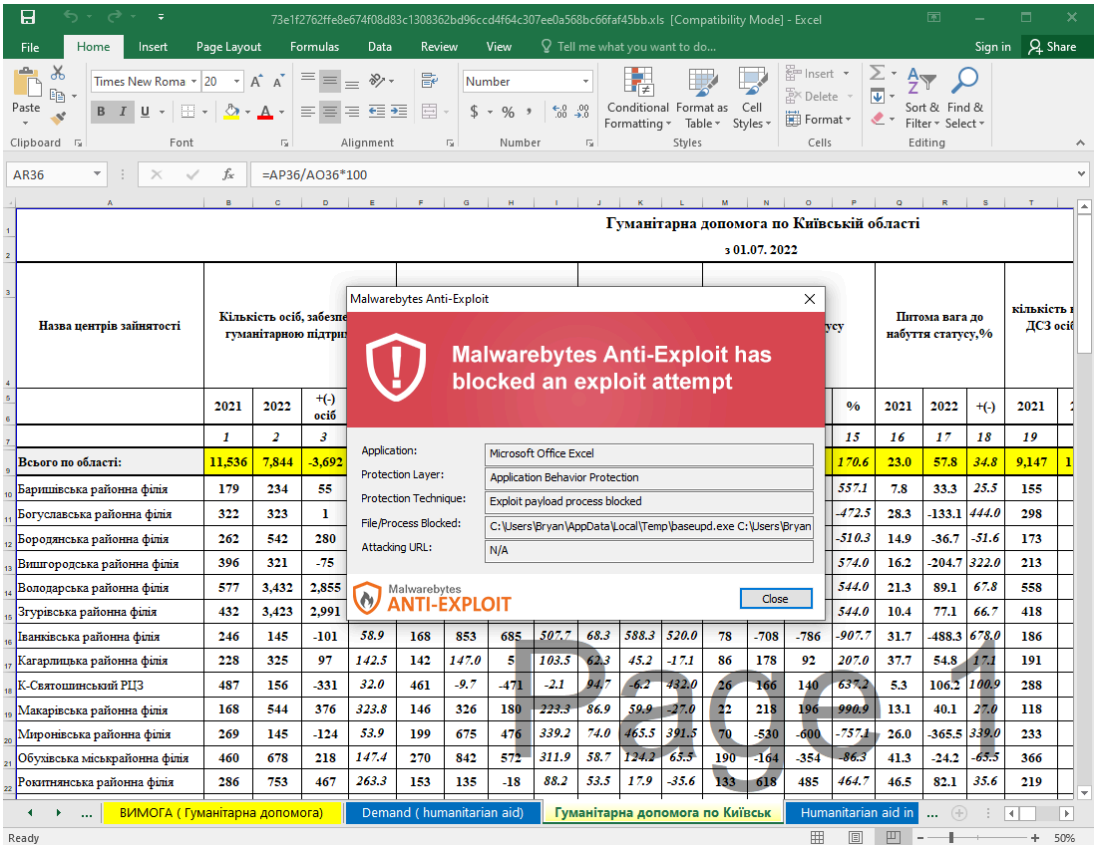
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

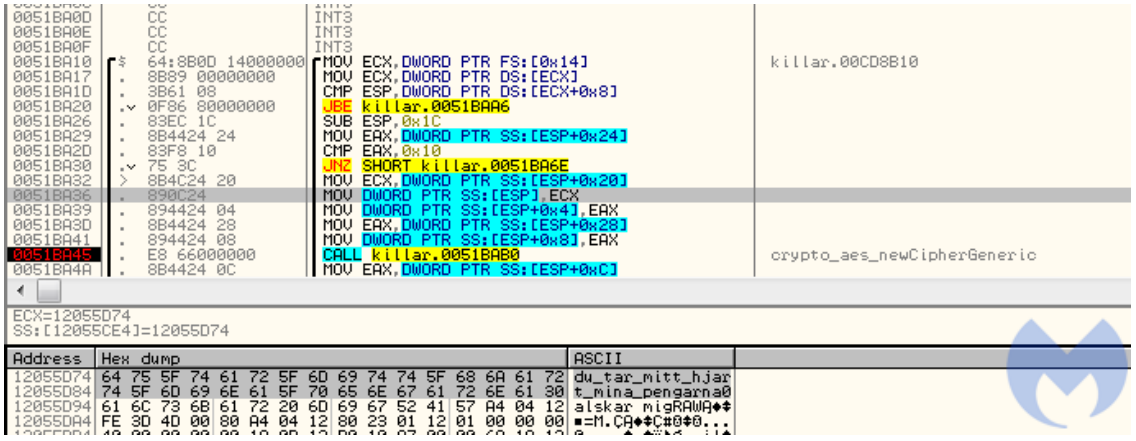
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

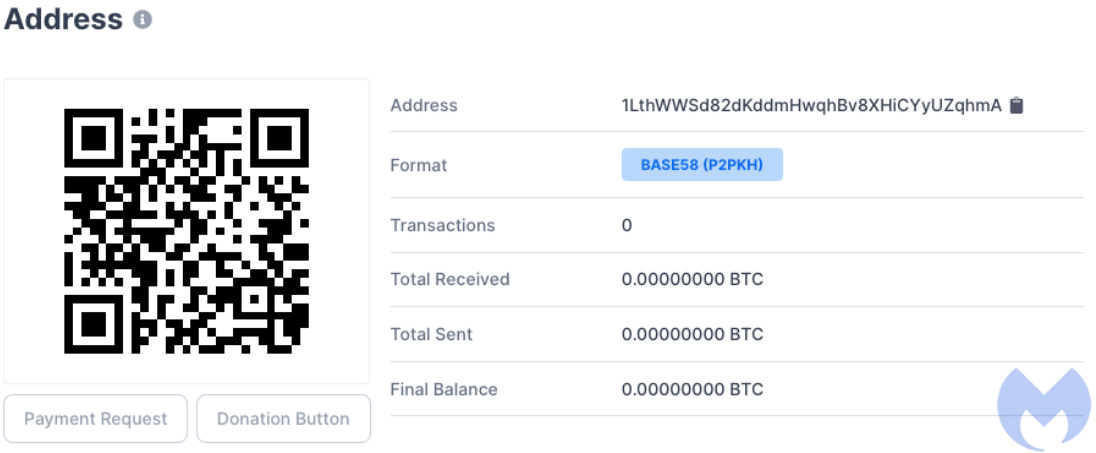
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

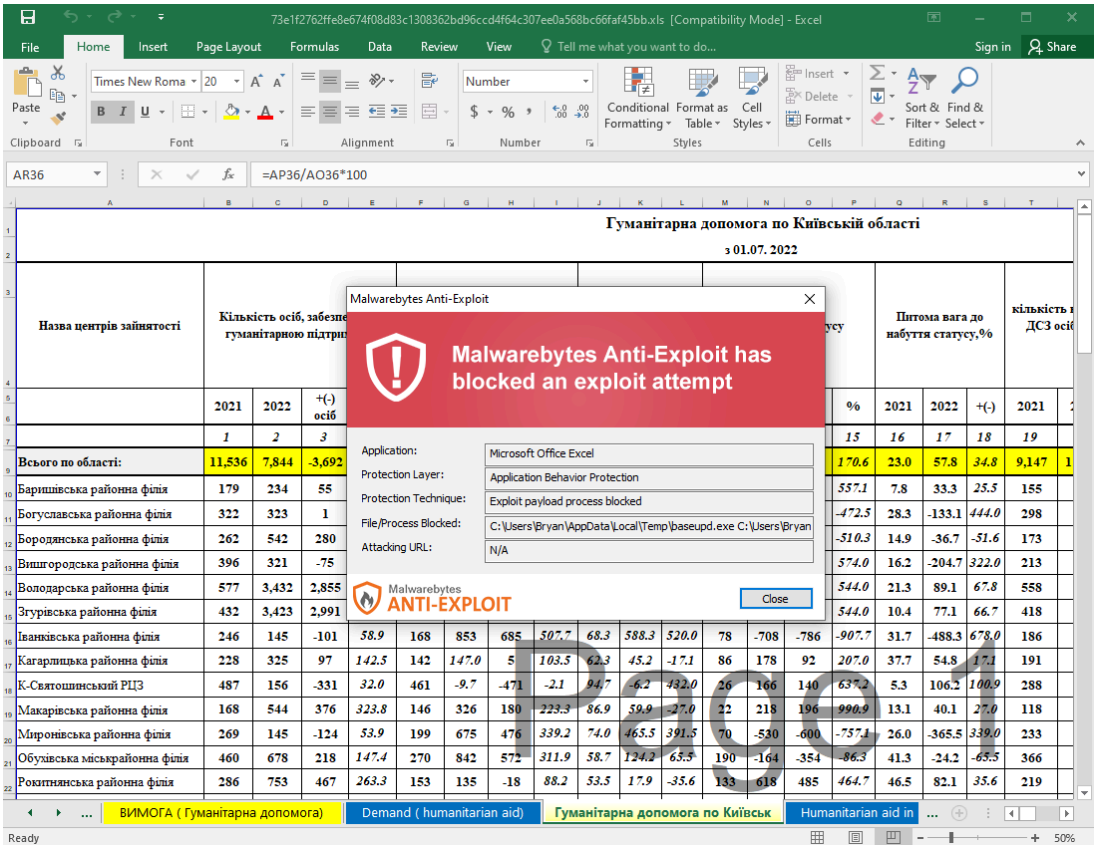
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

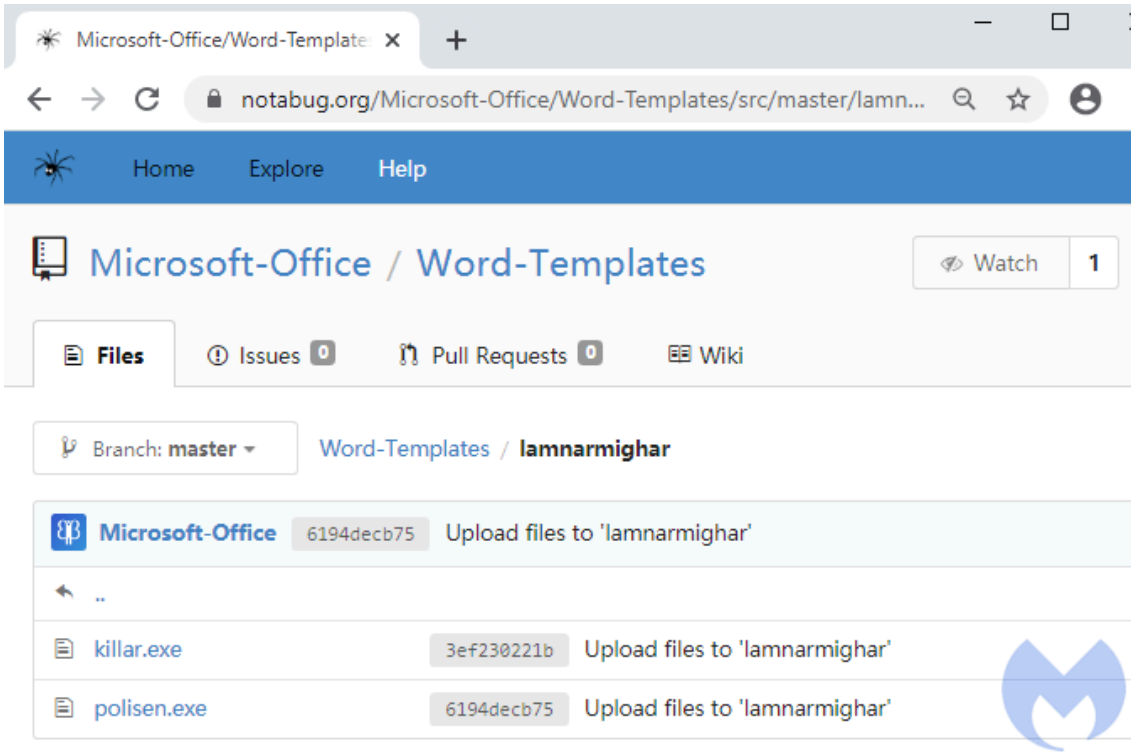
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



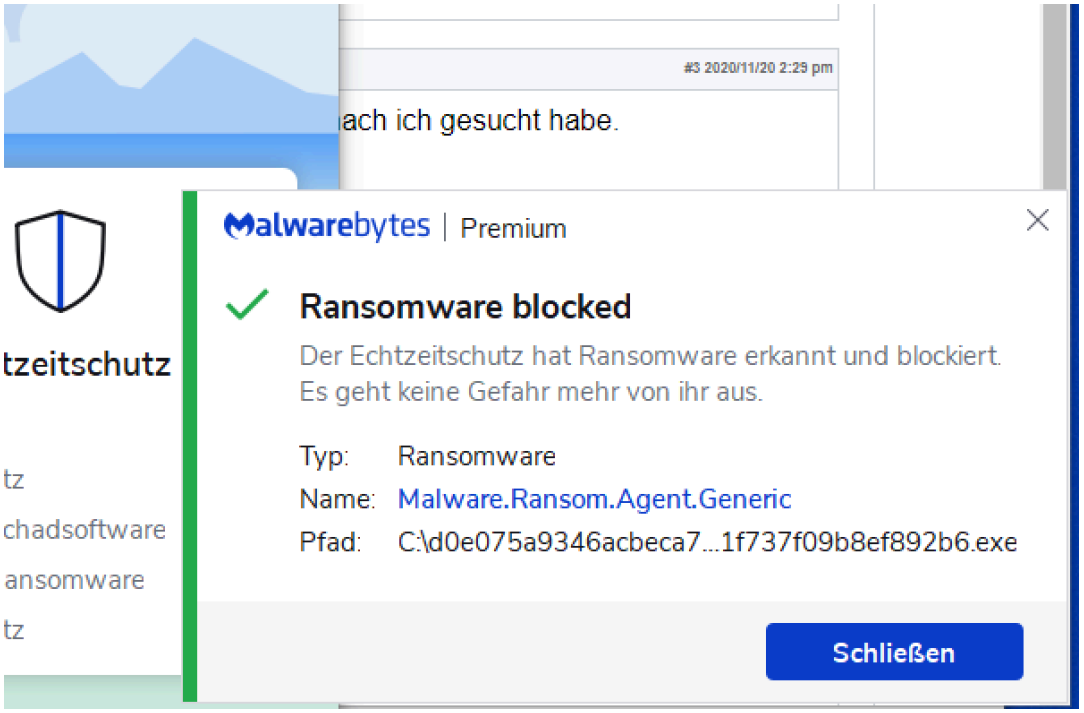
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

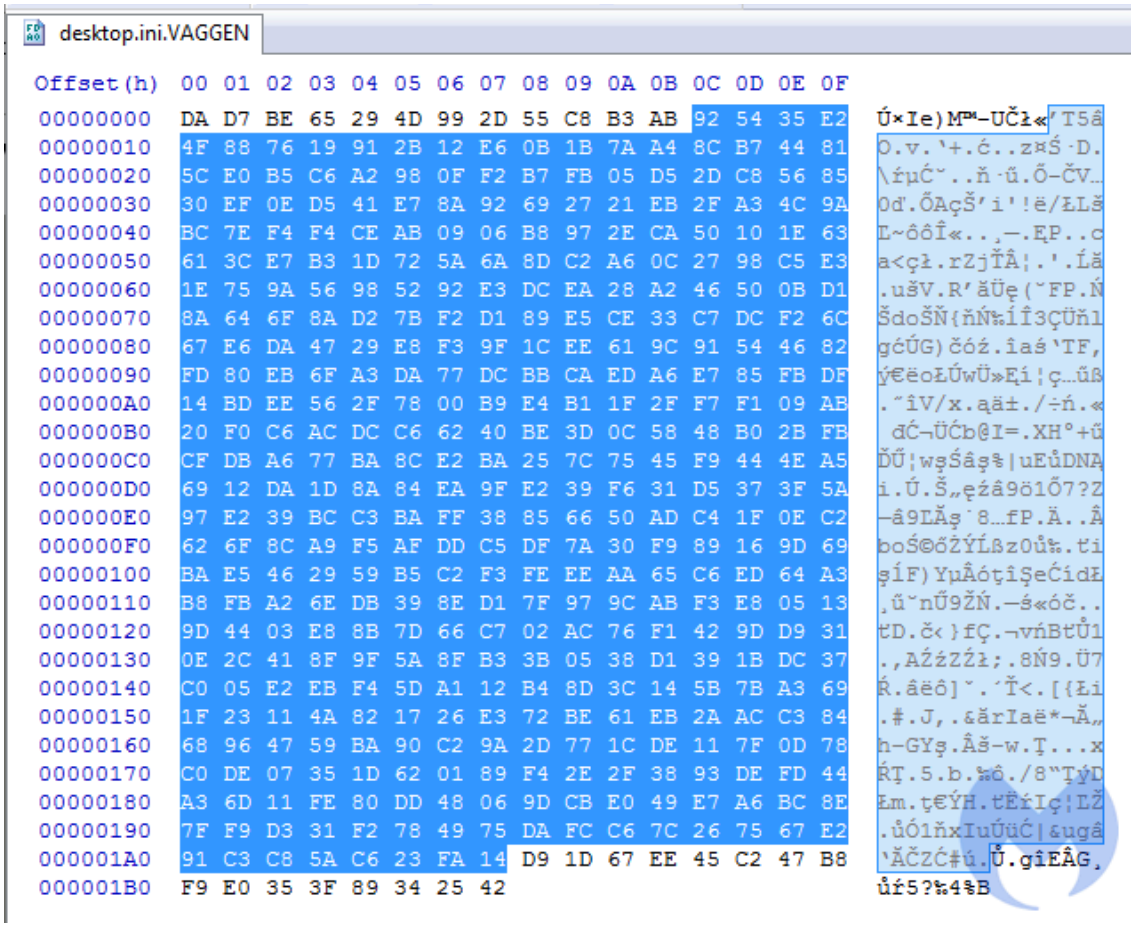
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

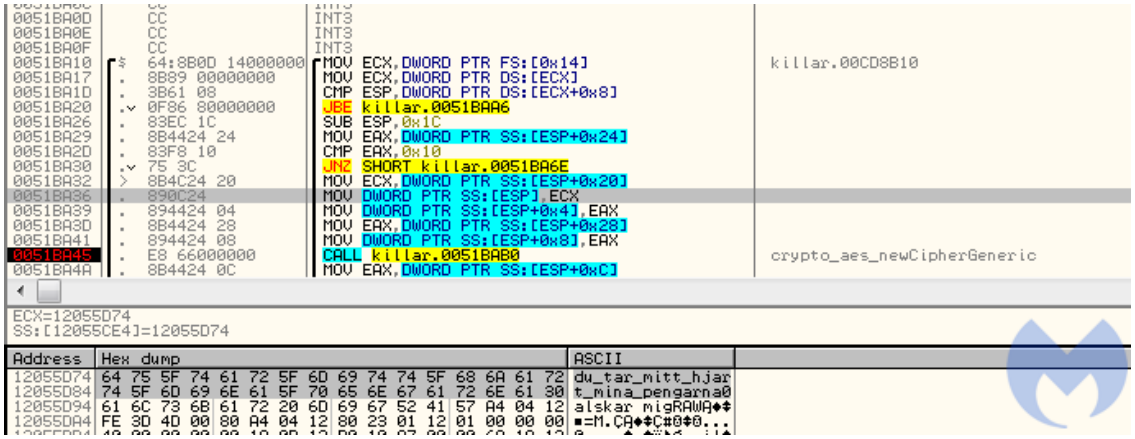


The content of the output file (with .VAGGEN extension) contains:

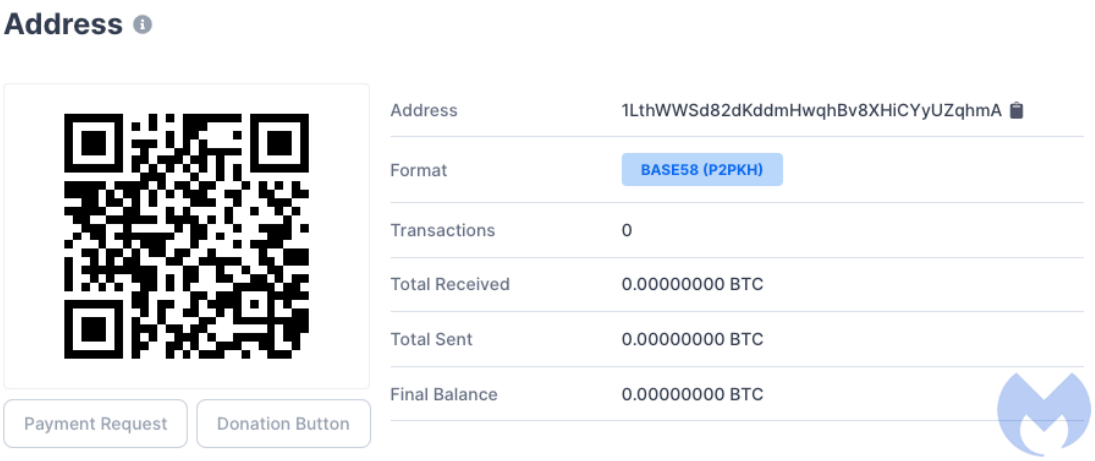
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

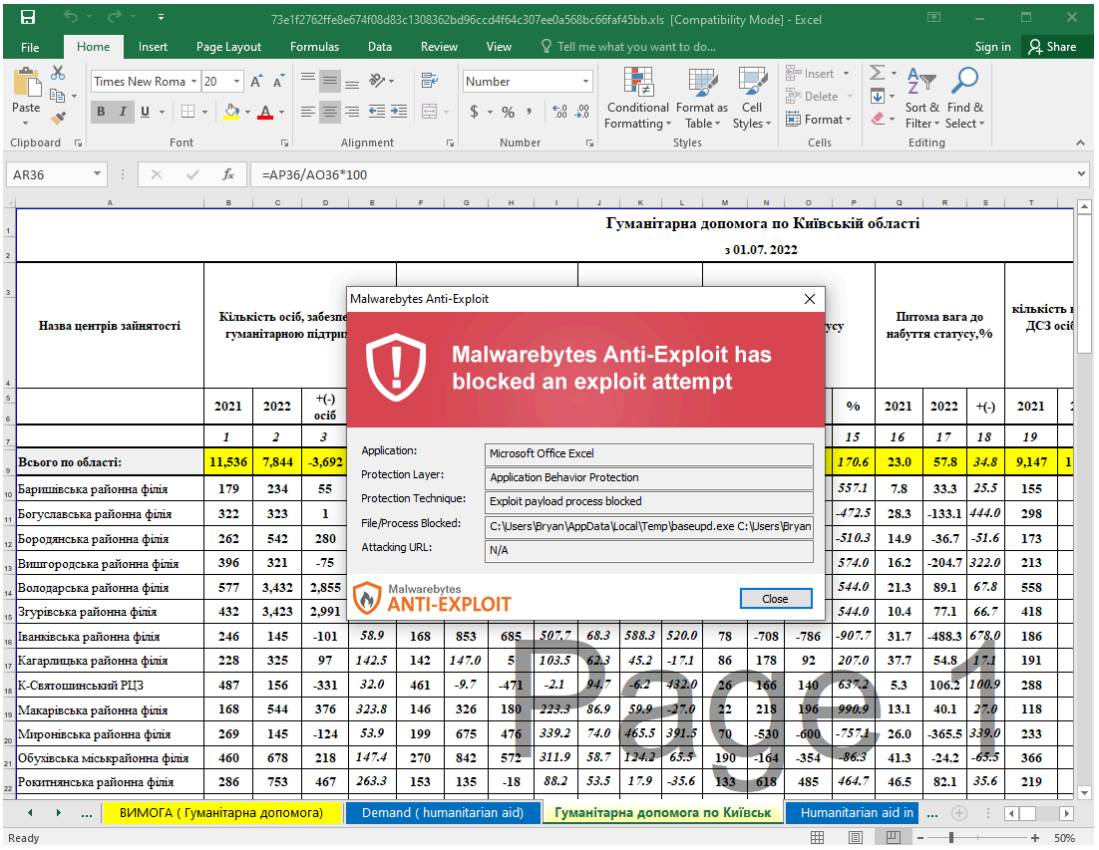
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

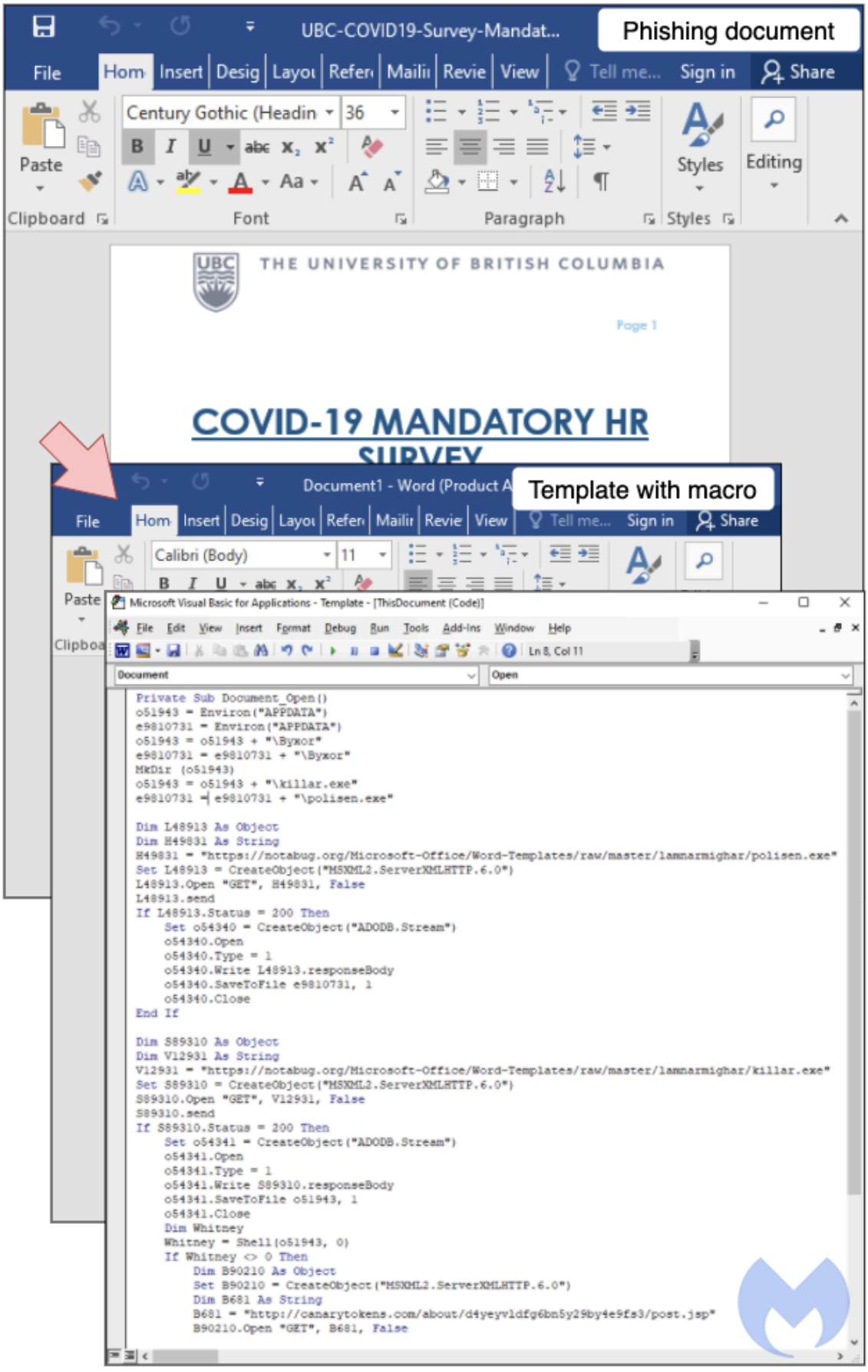
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

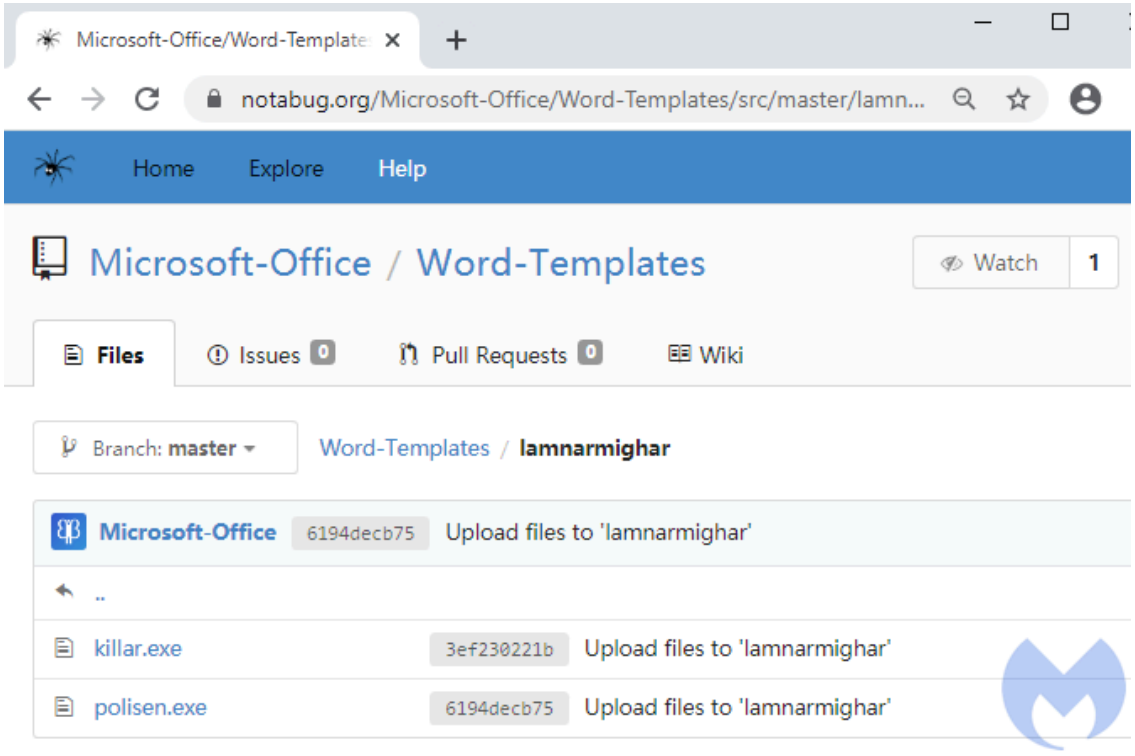
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



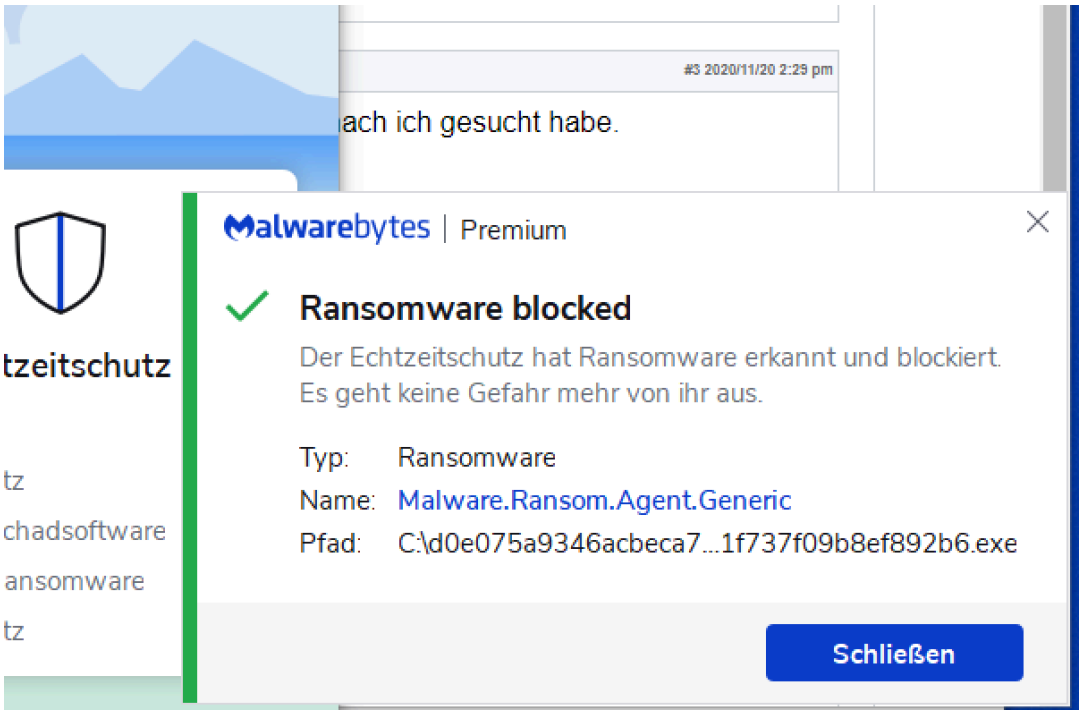
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

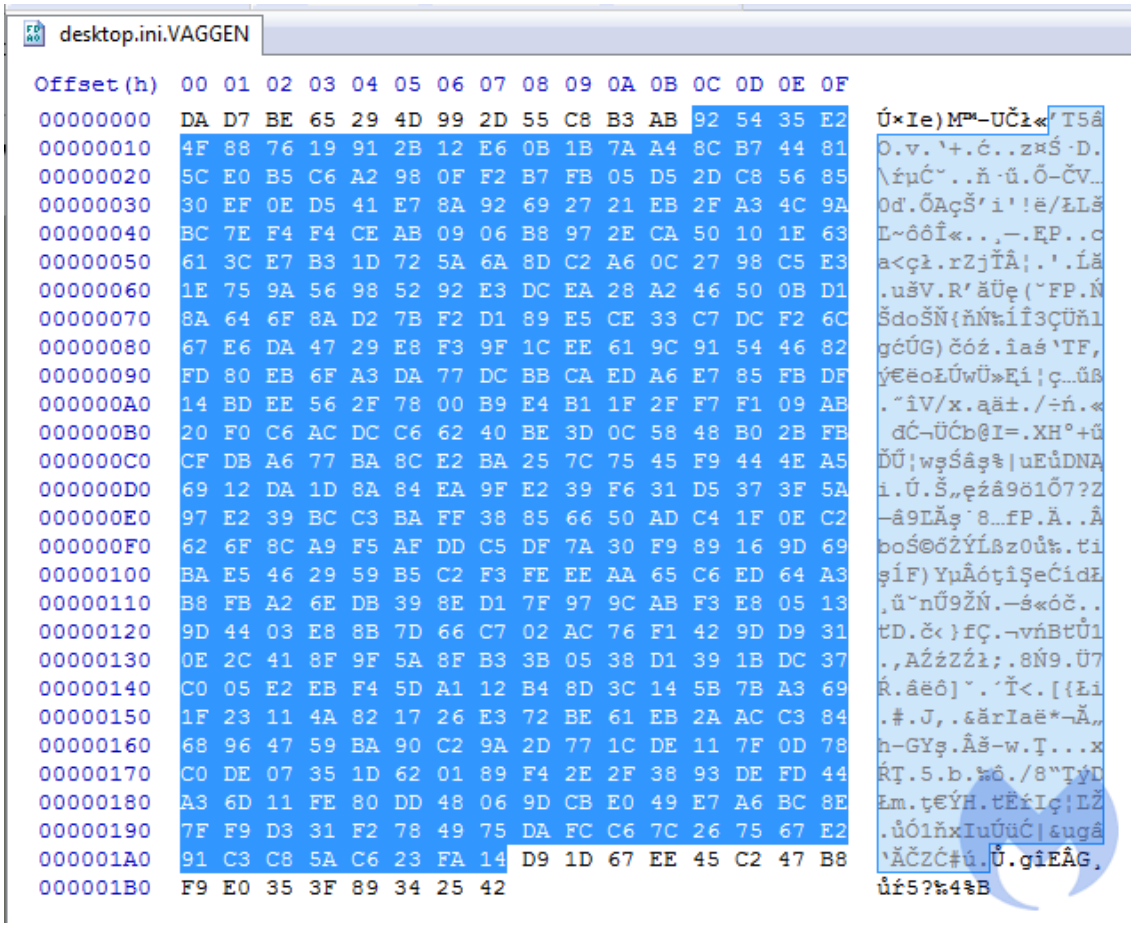
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

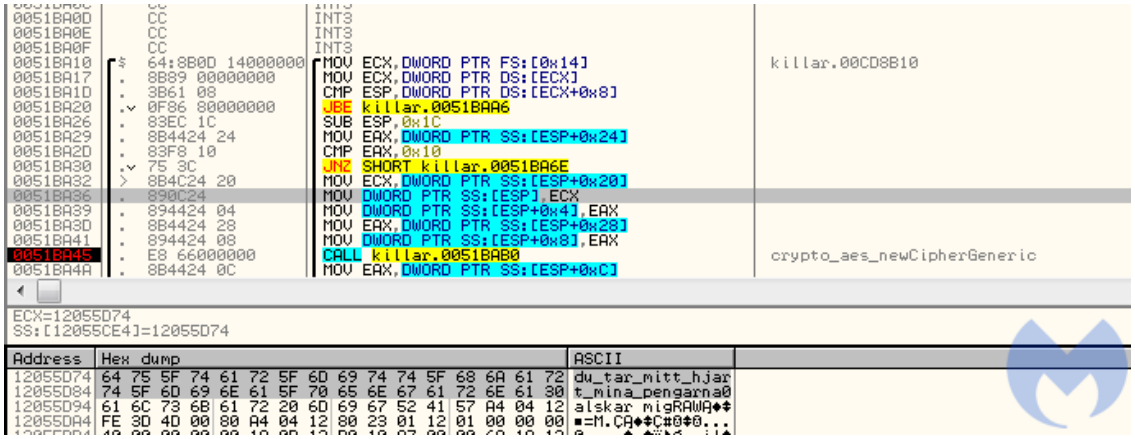


The content of the output file (with .VAGGEN extension) contains:

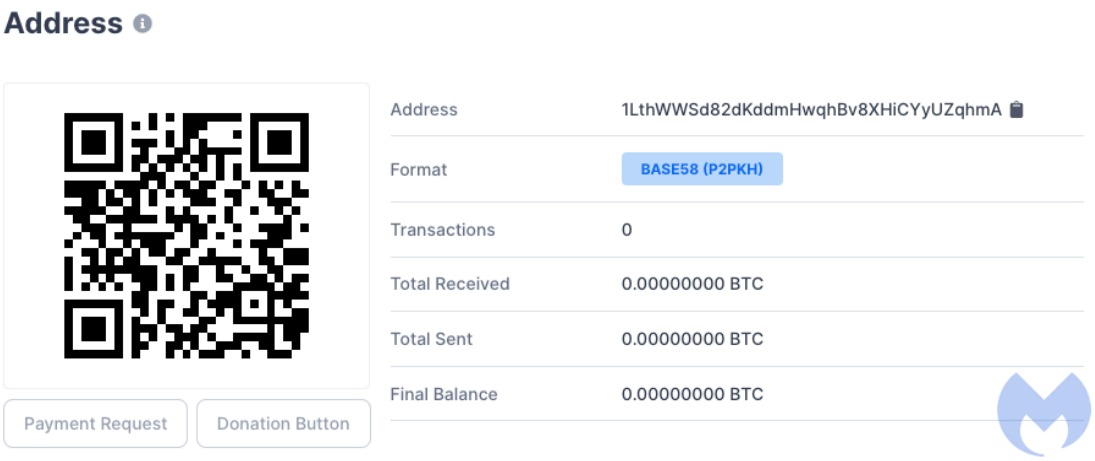
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

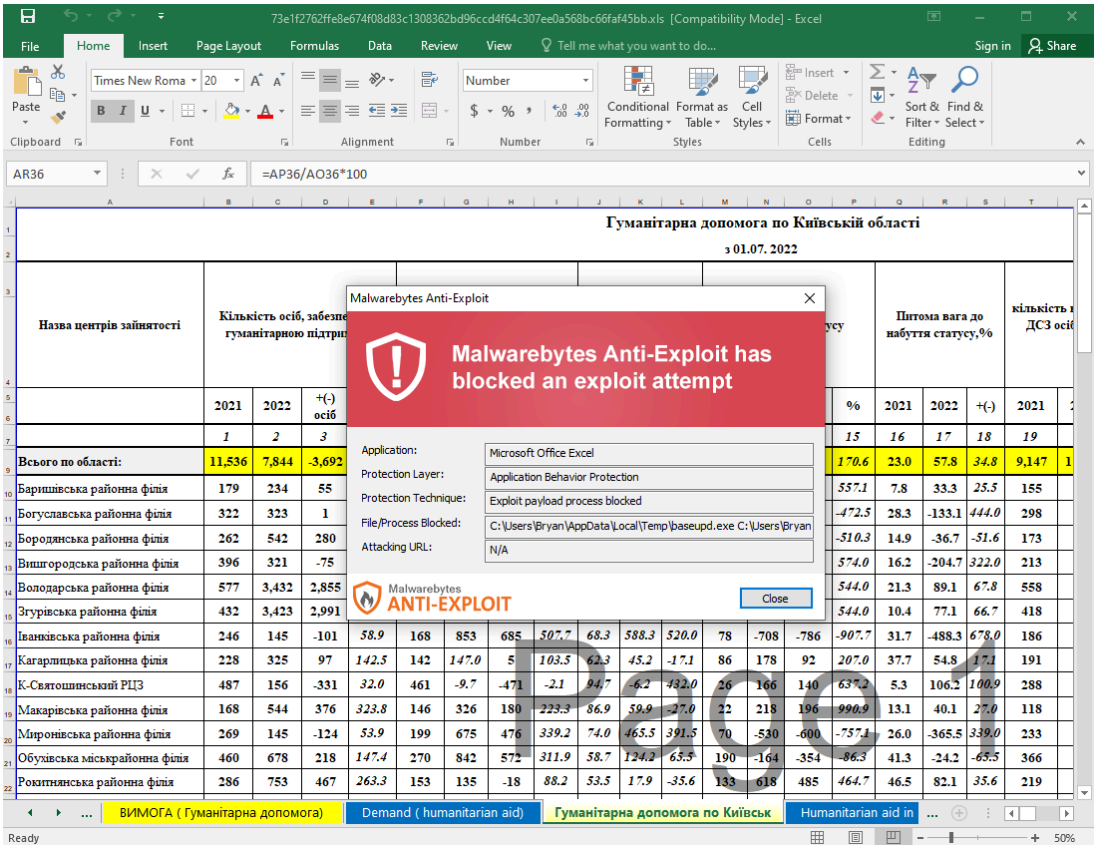
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

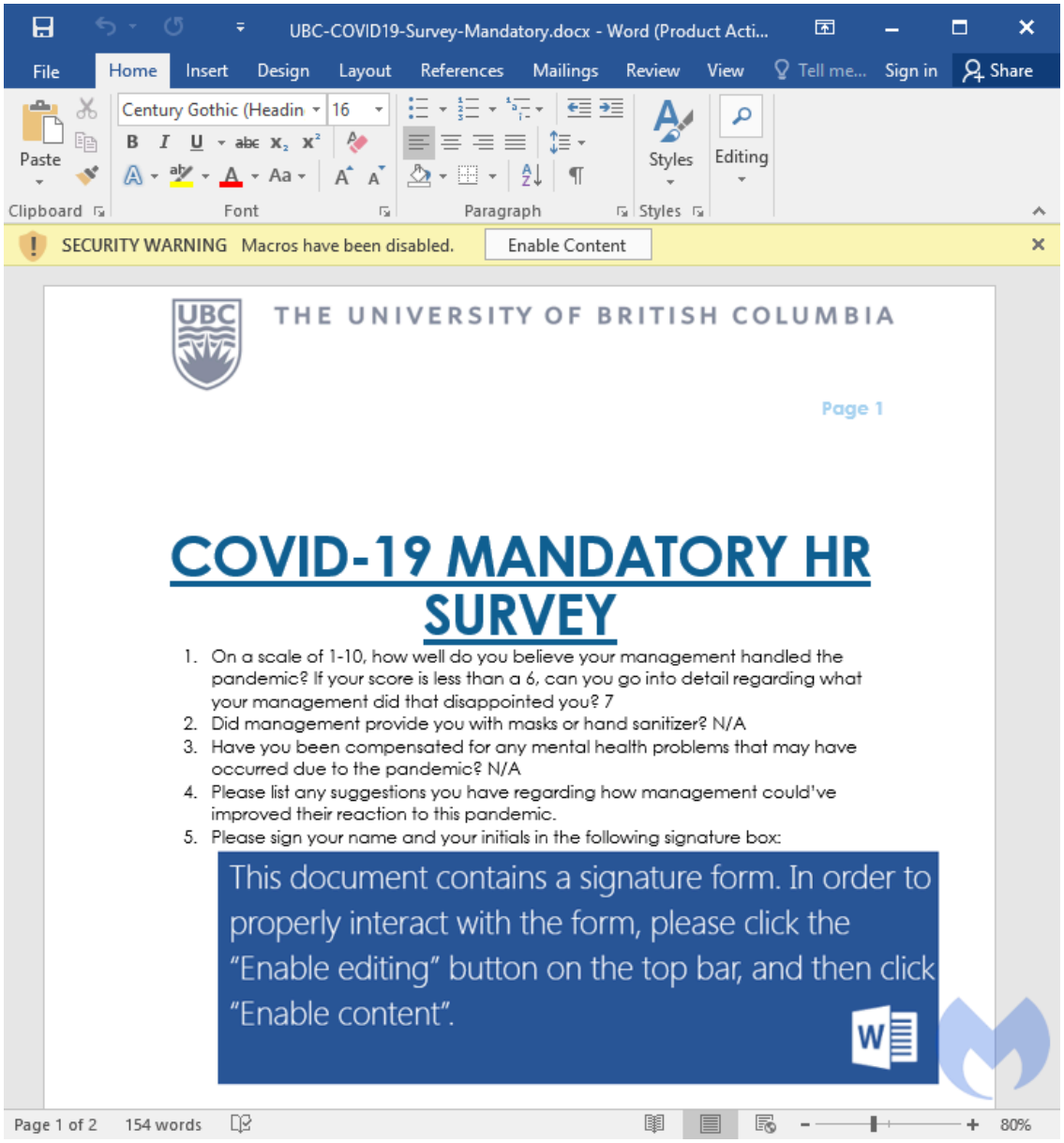
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

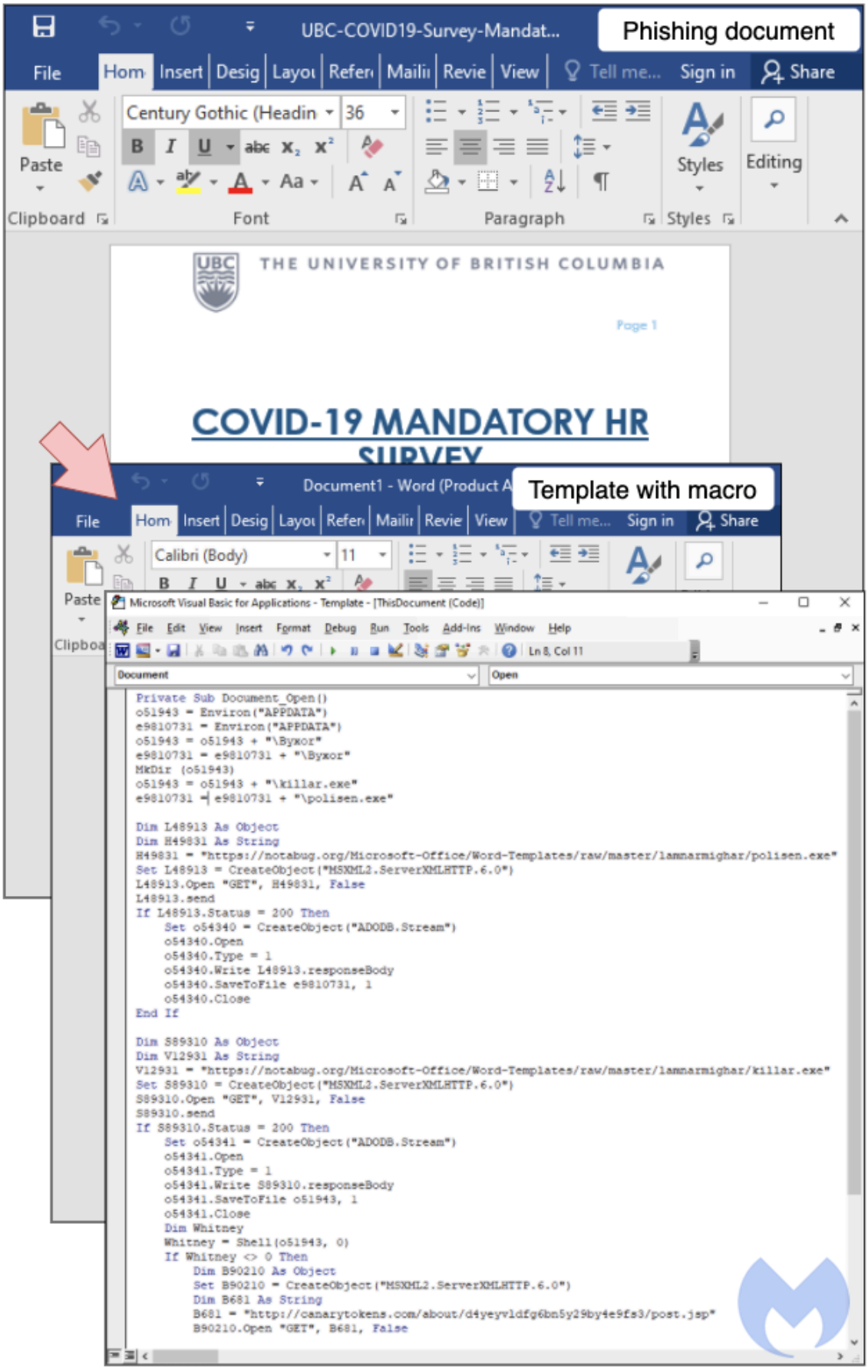
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

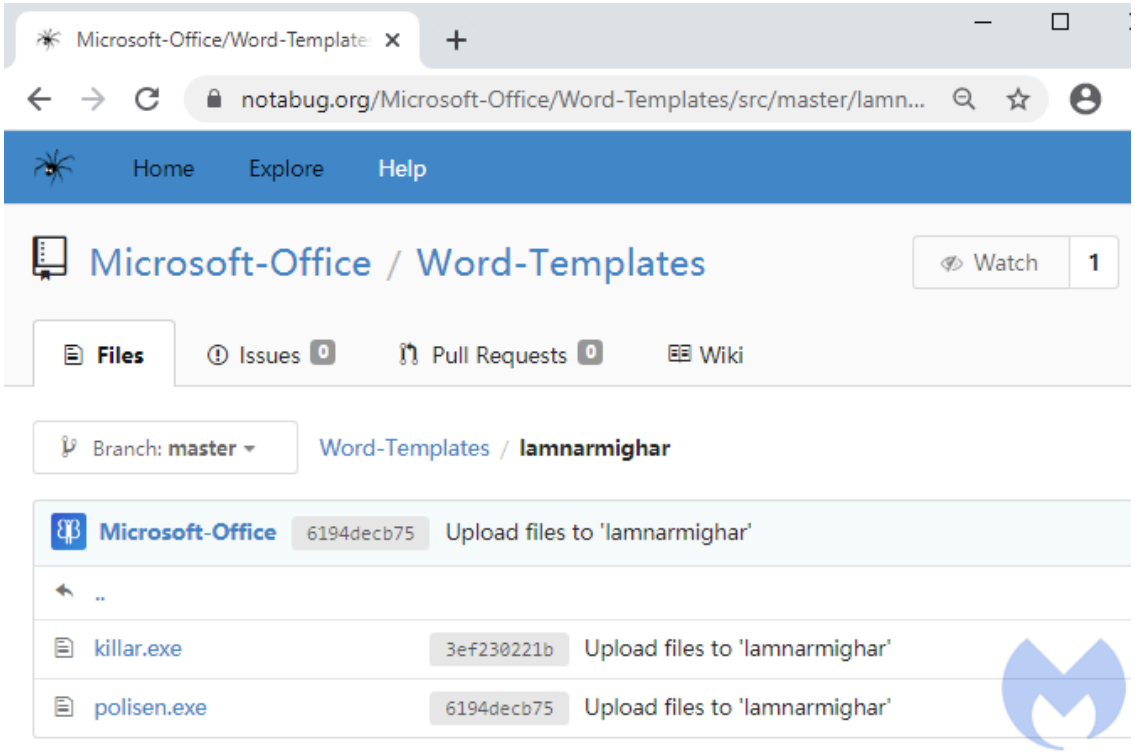
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



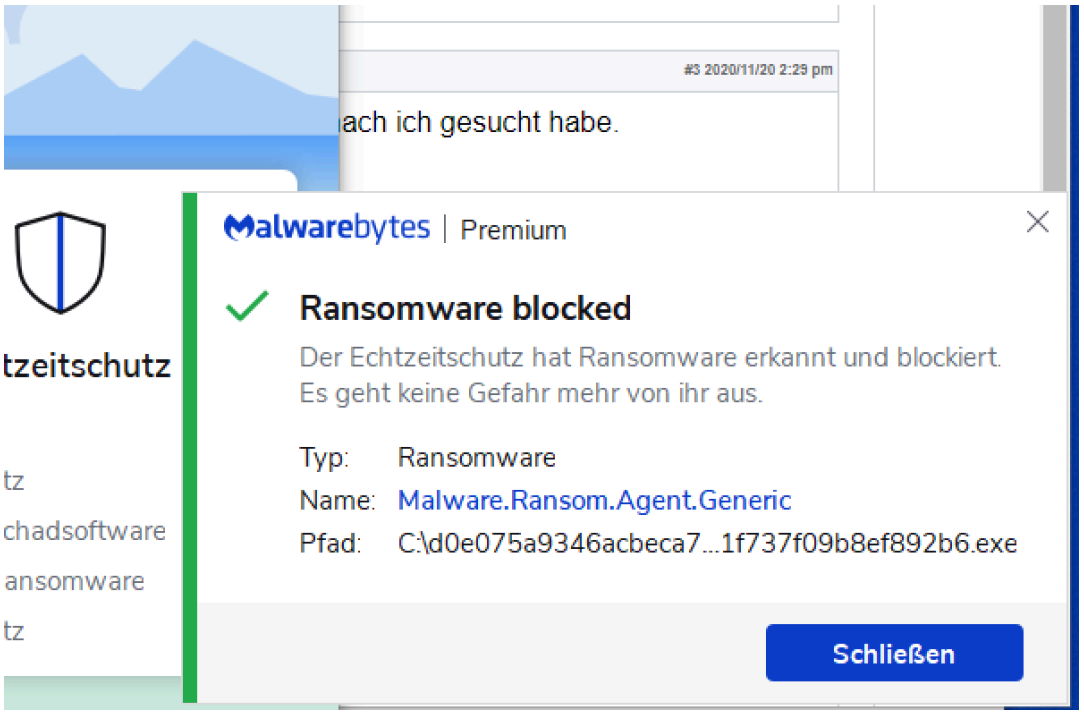
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

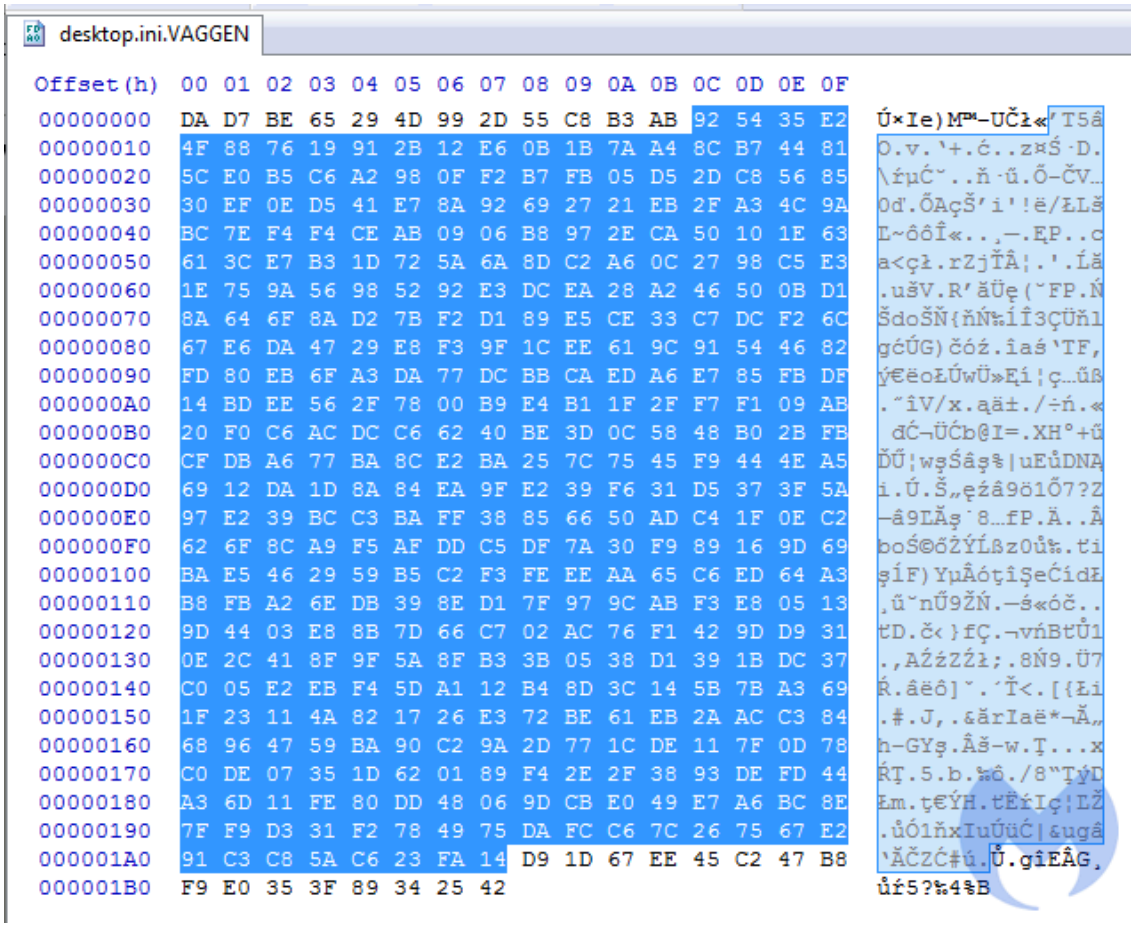
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

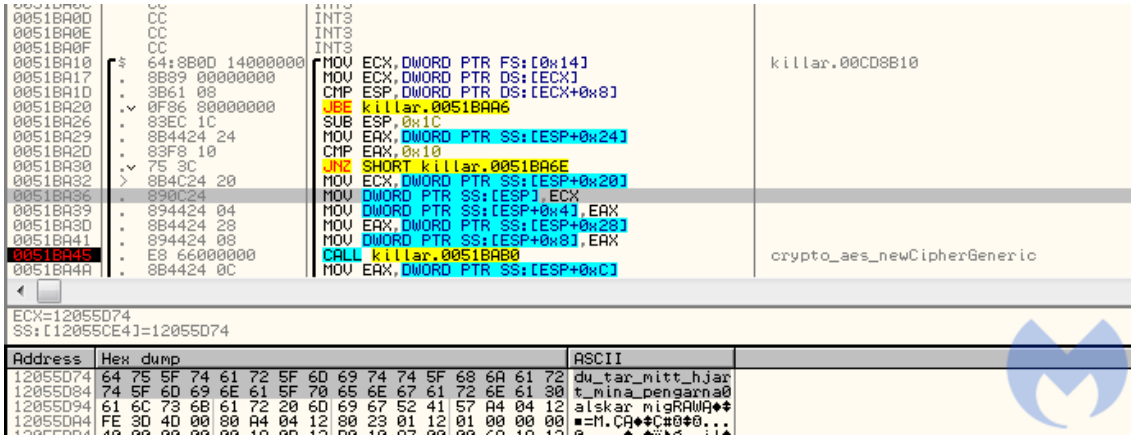


The content of the output file (with .VAGGEN extension) contains:

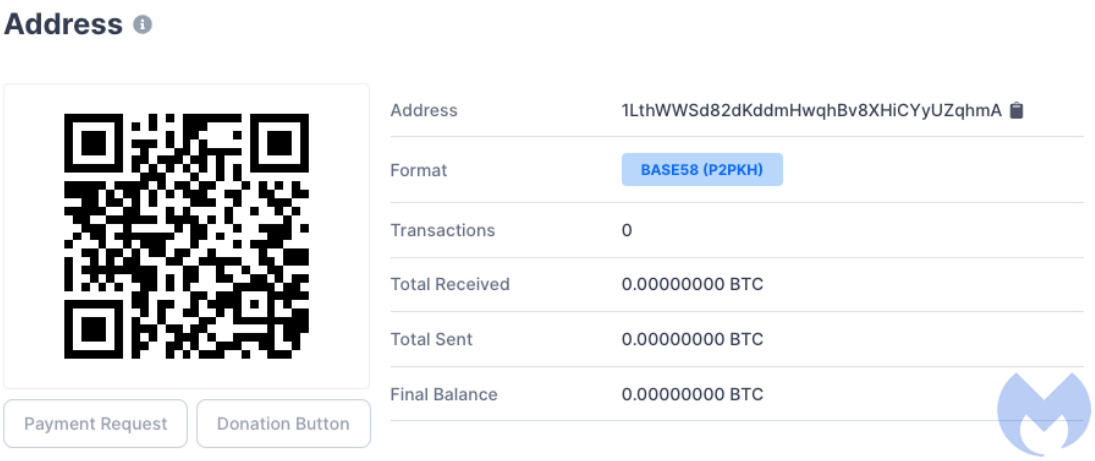
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

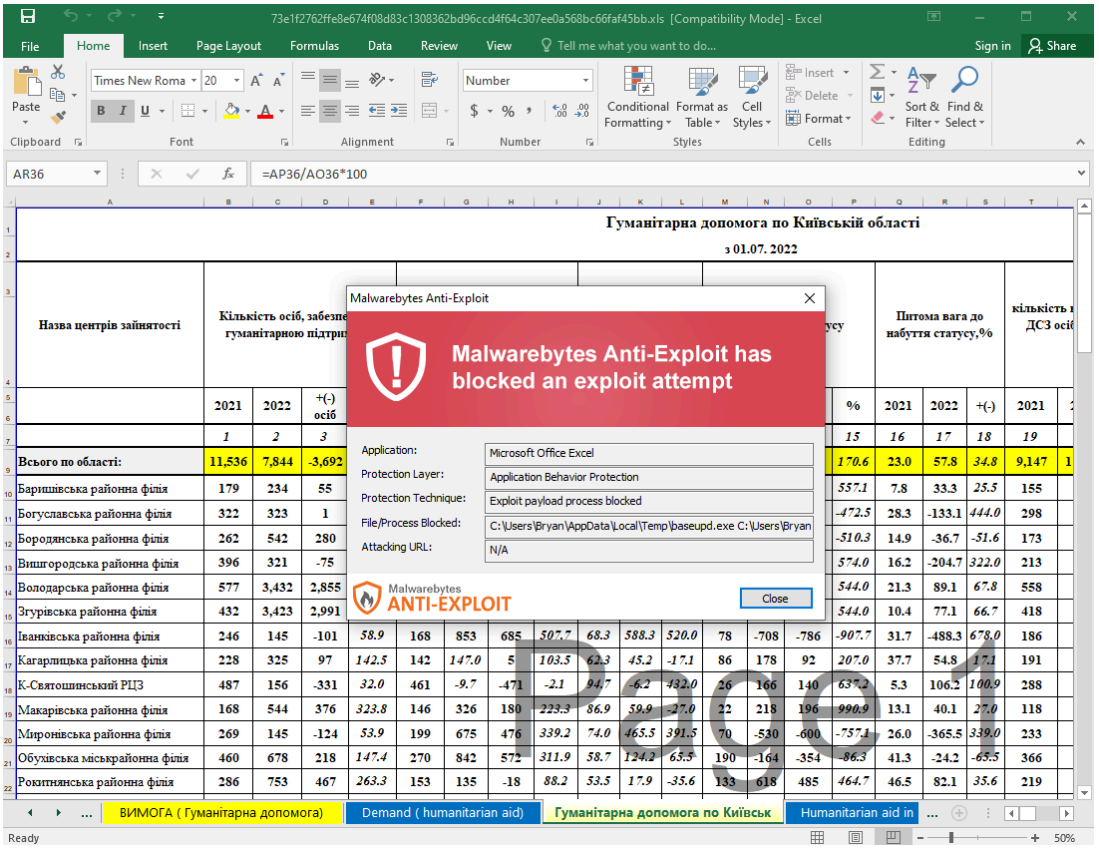
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

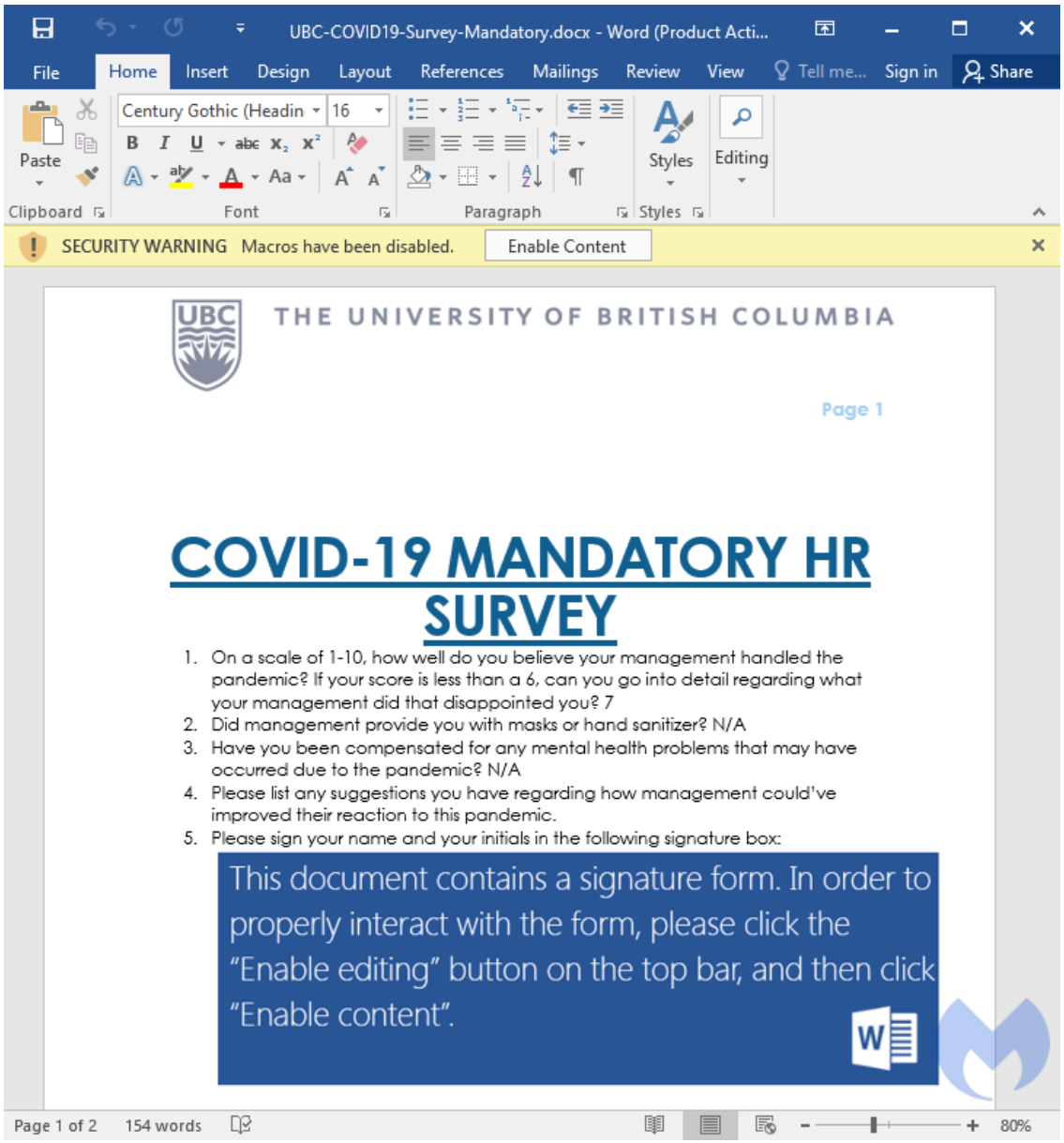
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

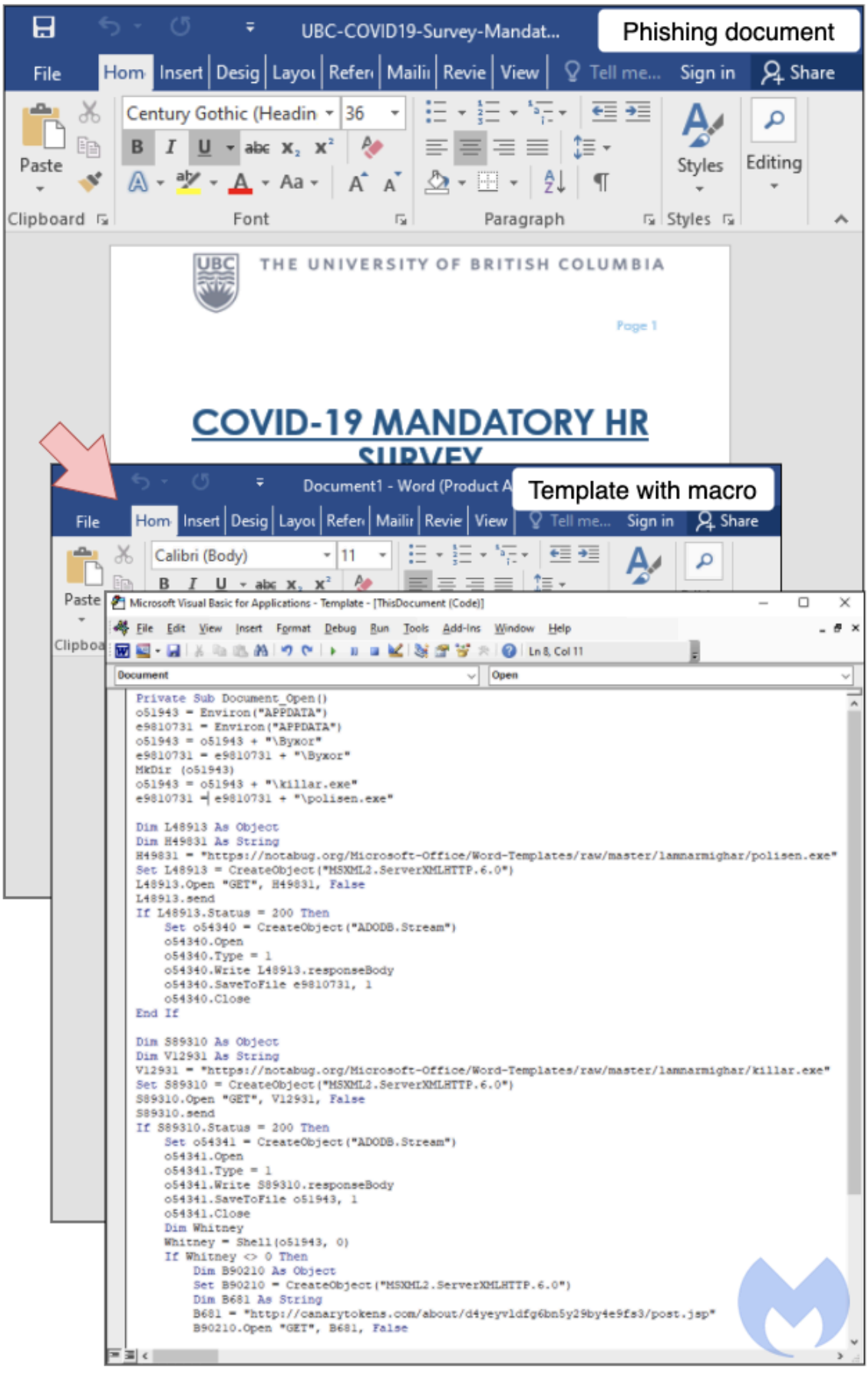
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

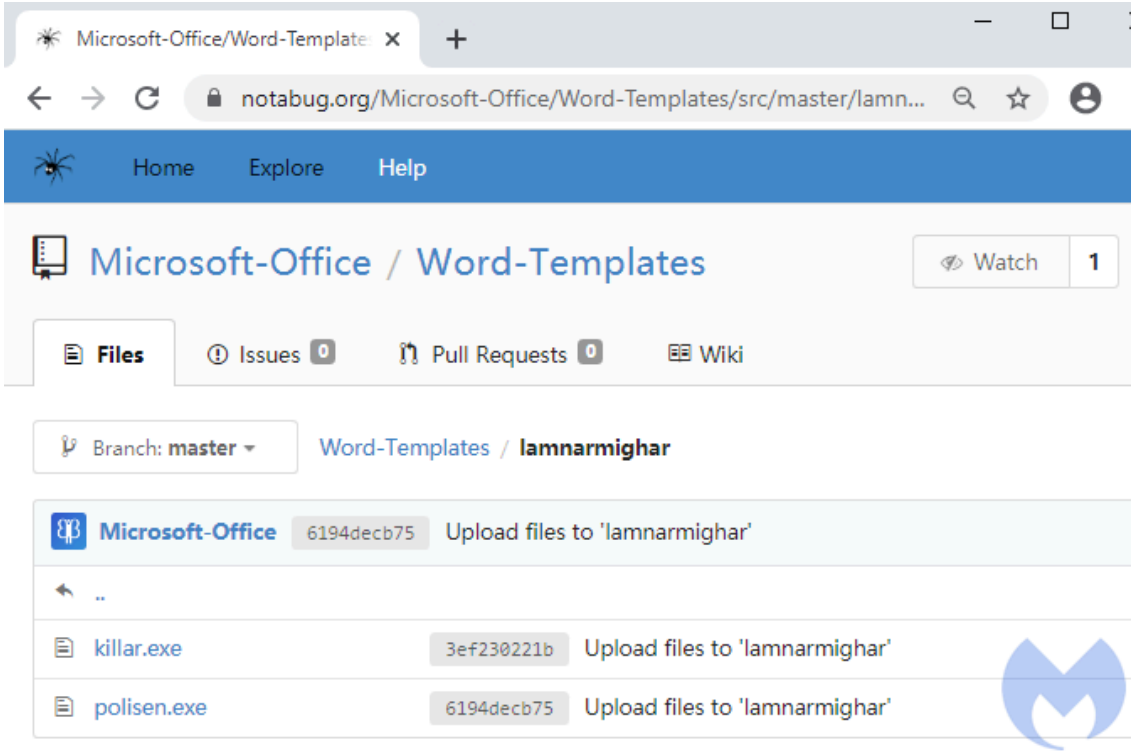
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



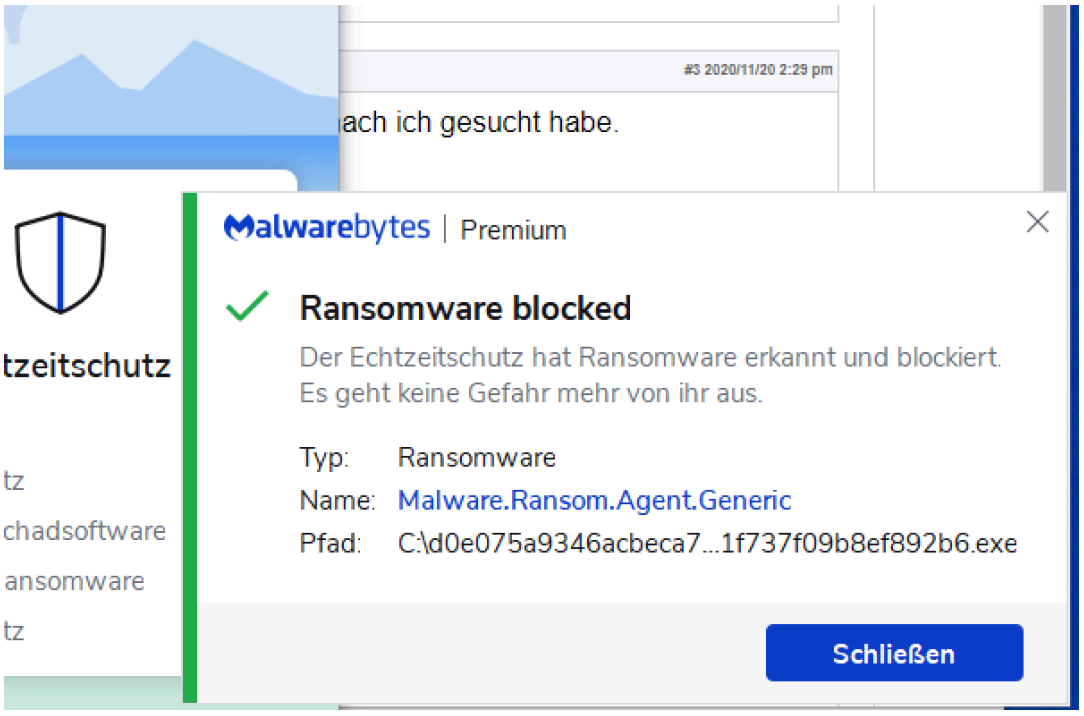
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

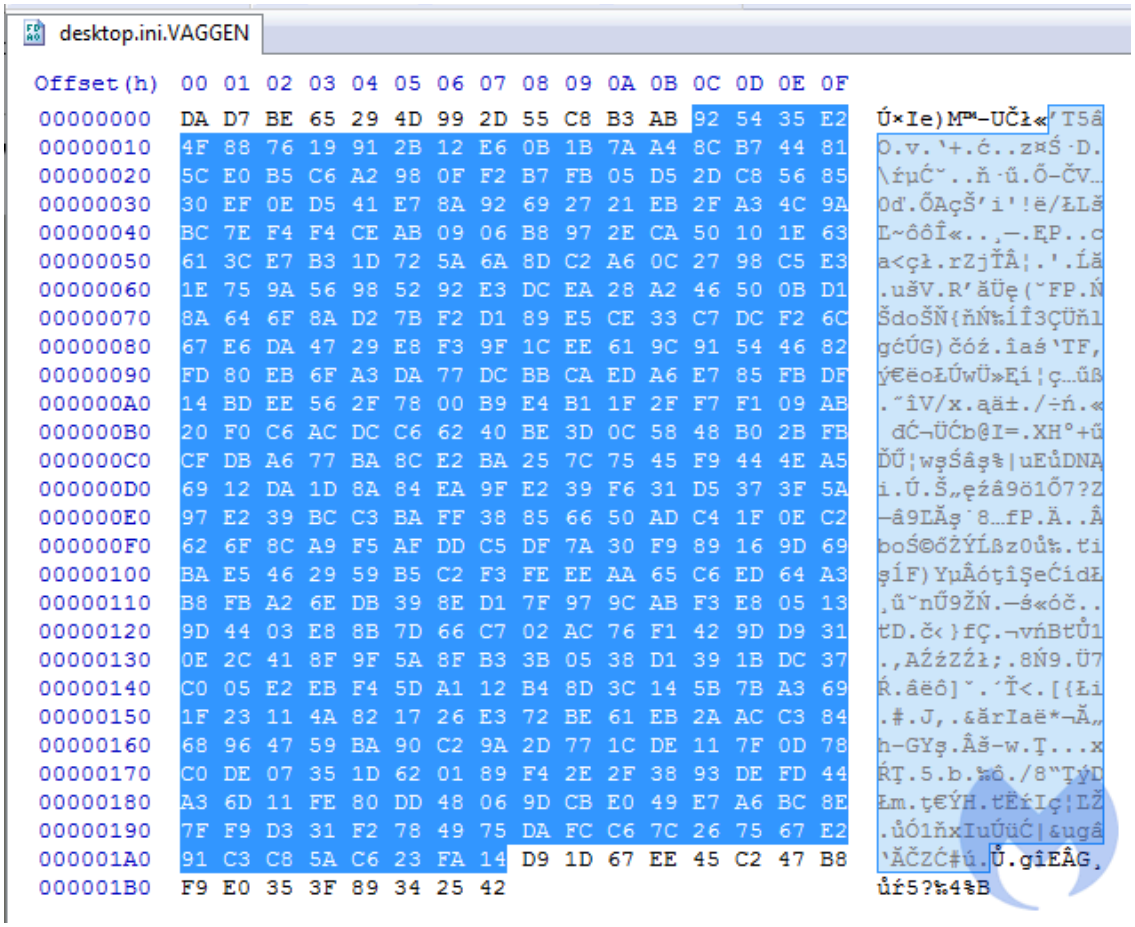
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

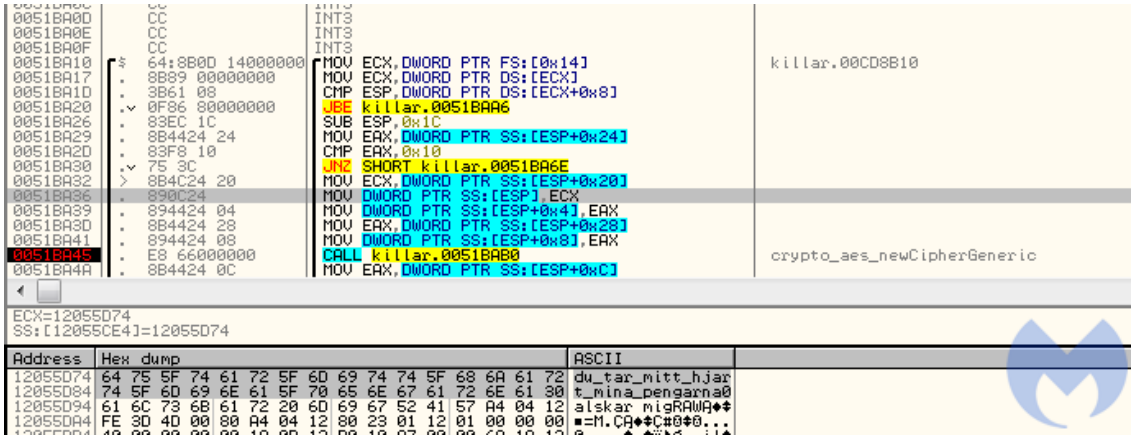


The content of the output file (with .VAGGEN extension) contains:

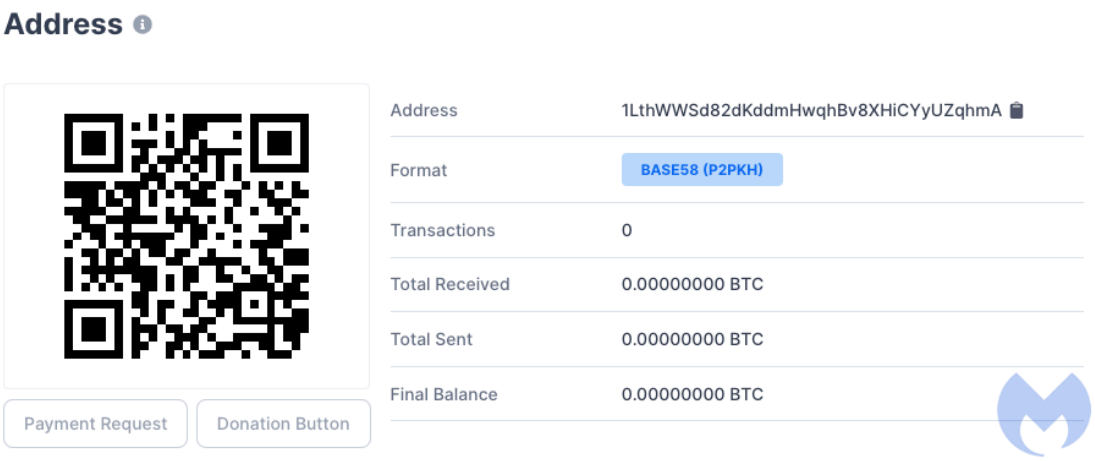
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

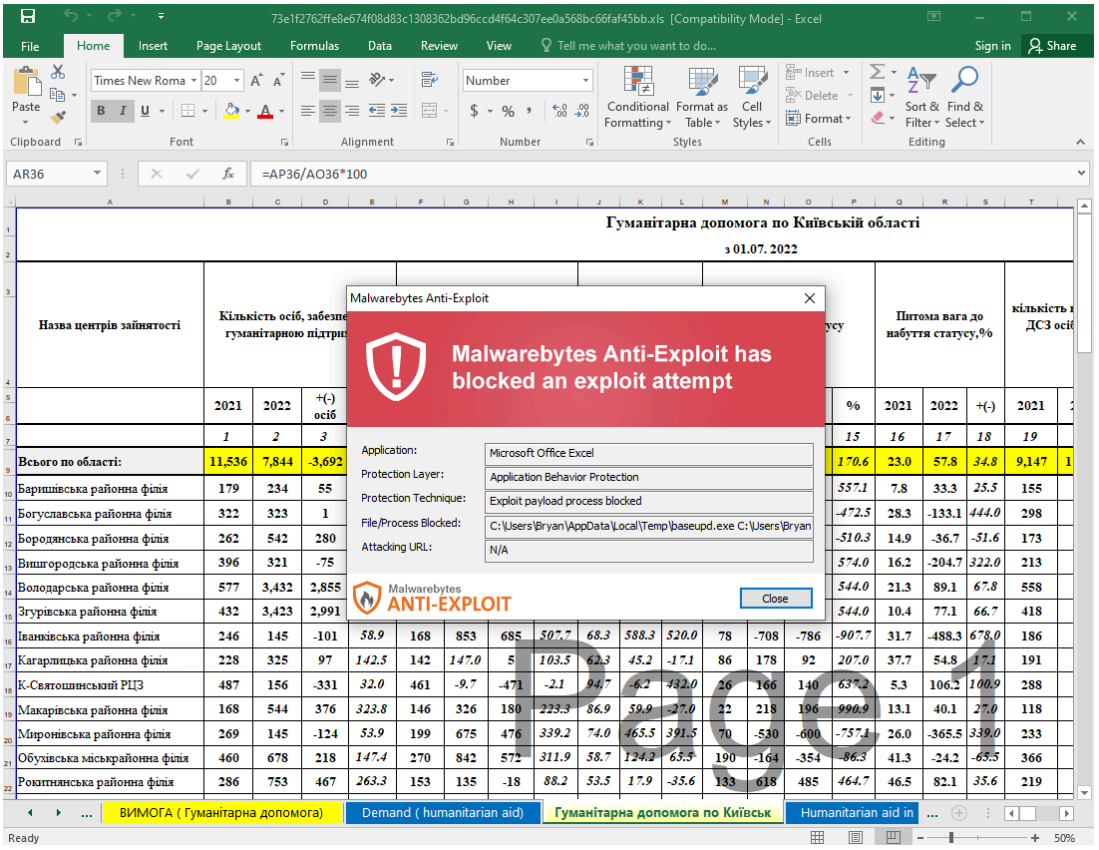
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

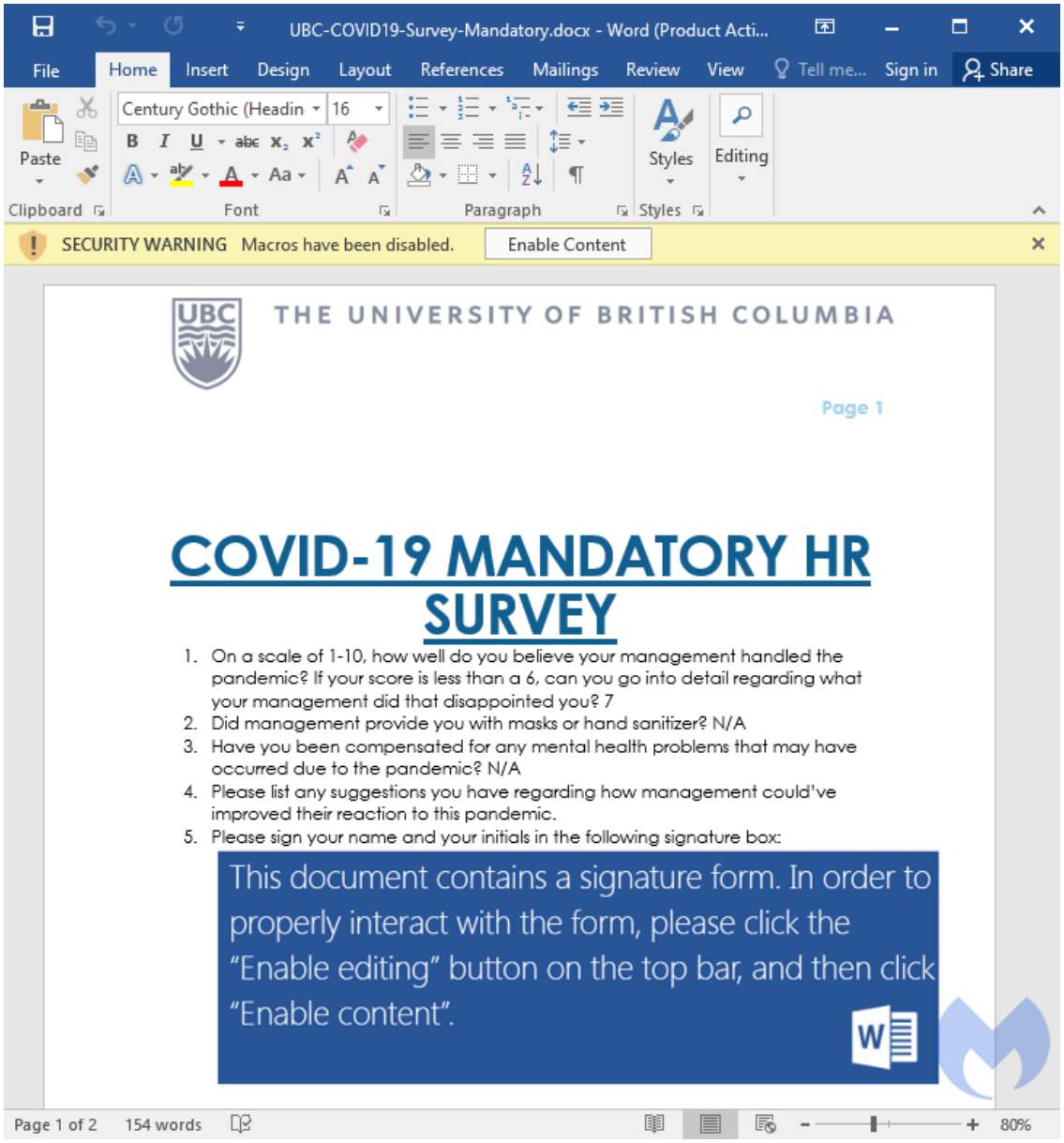
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

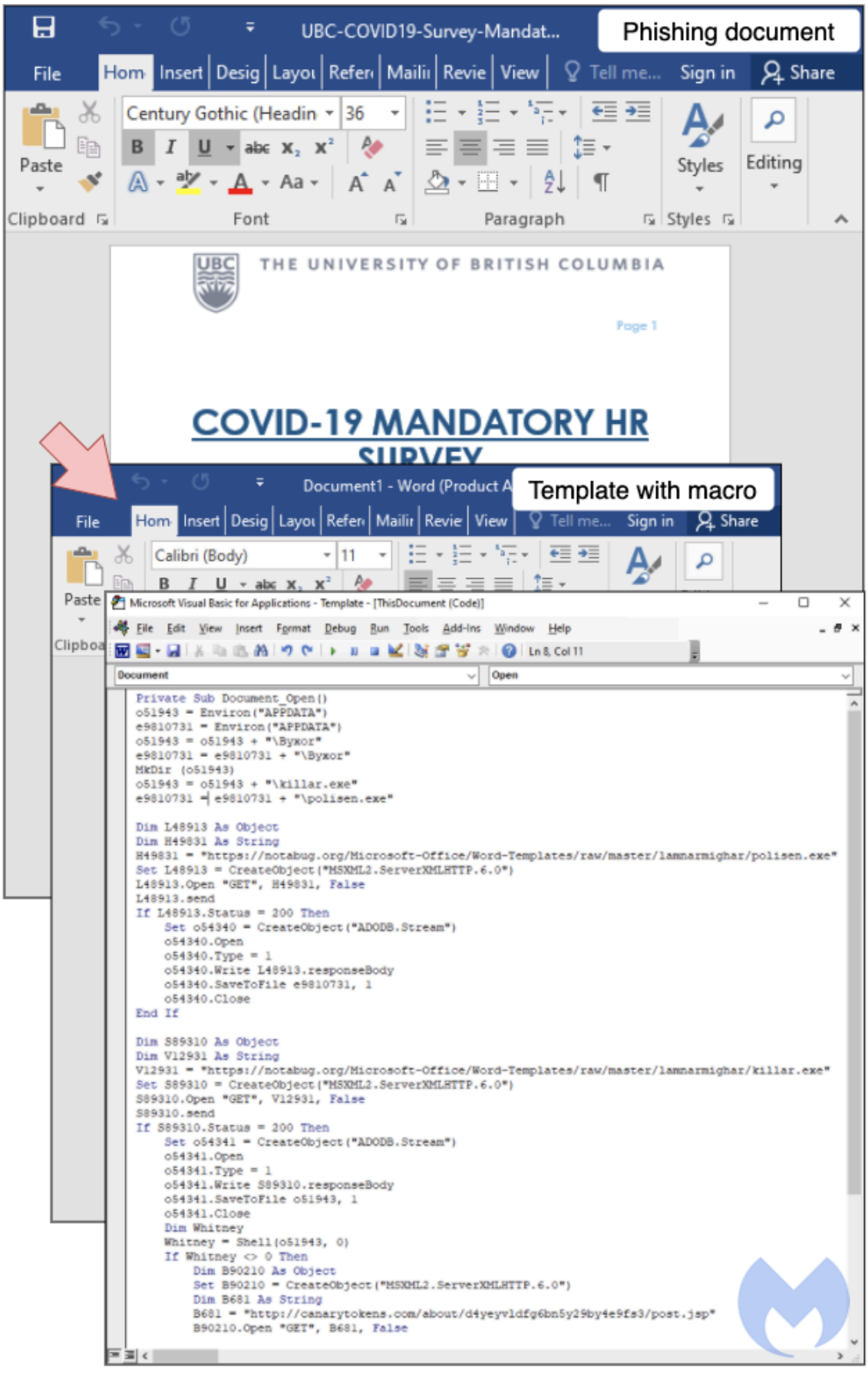
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

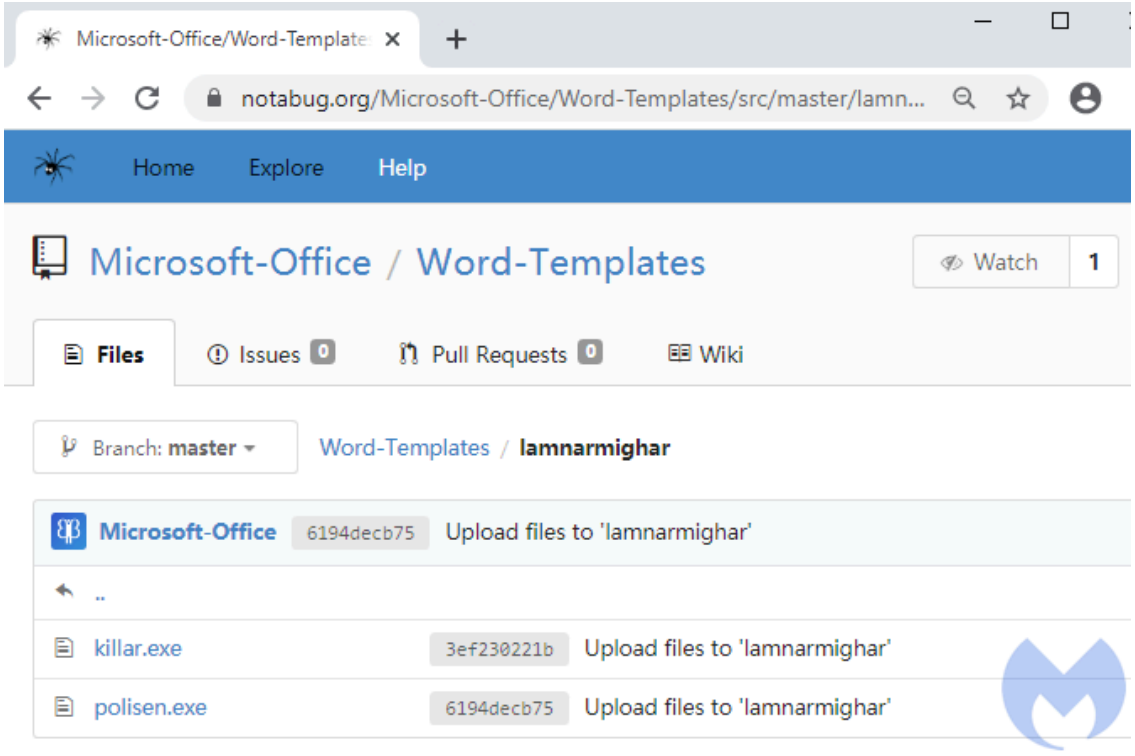
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



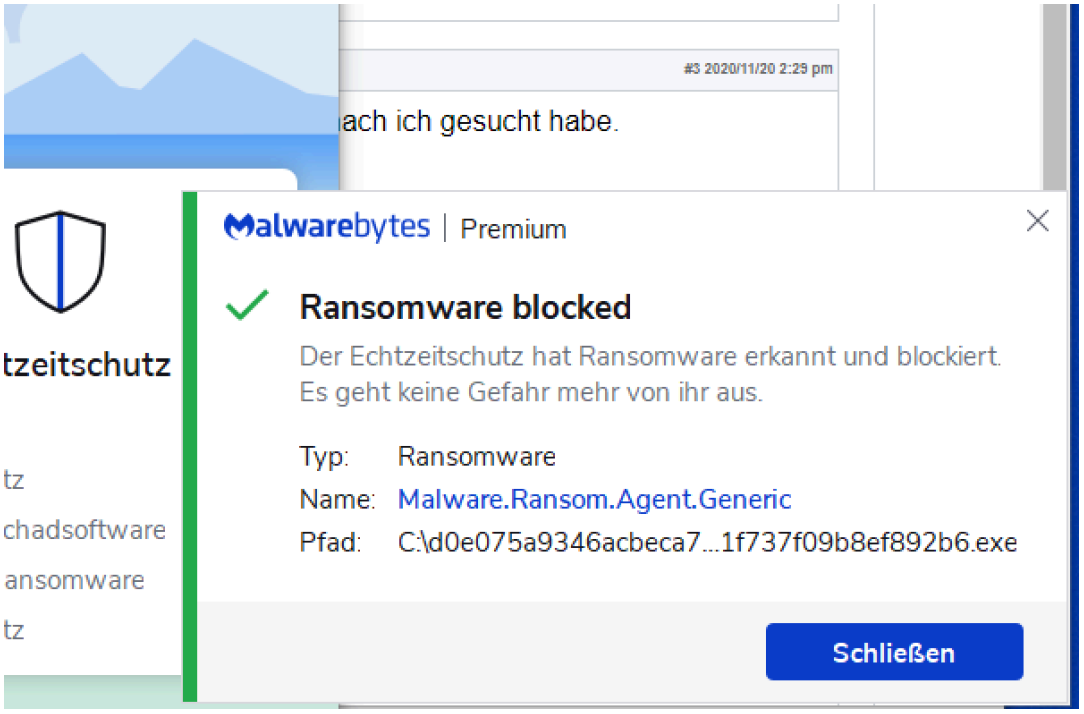
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

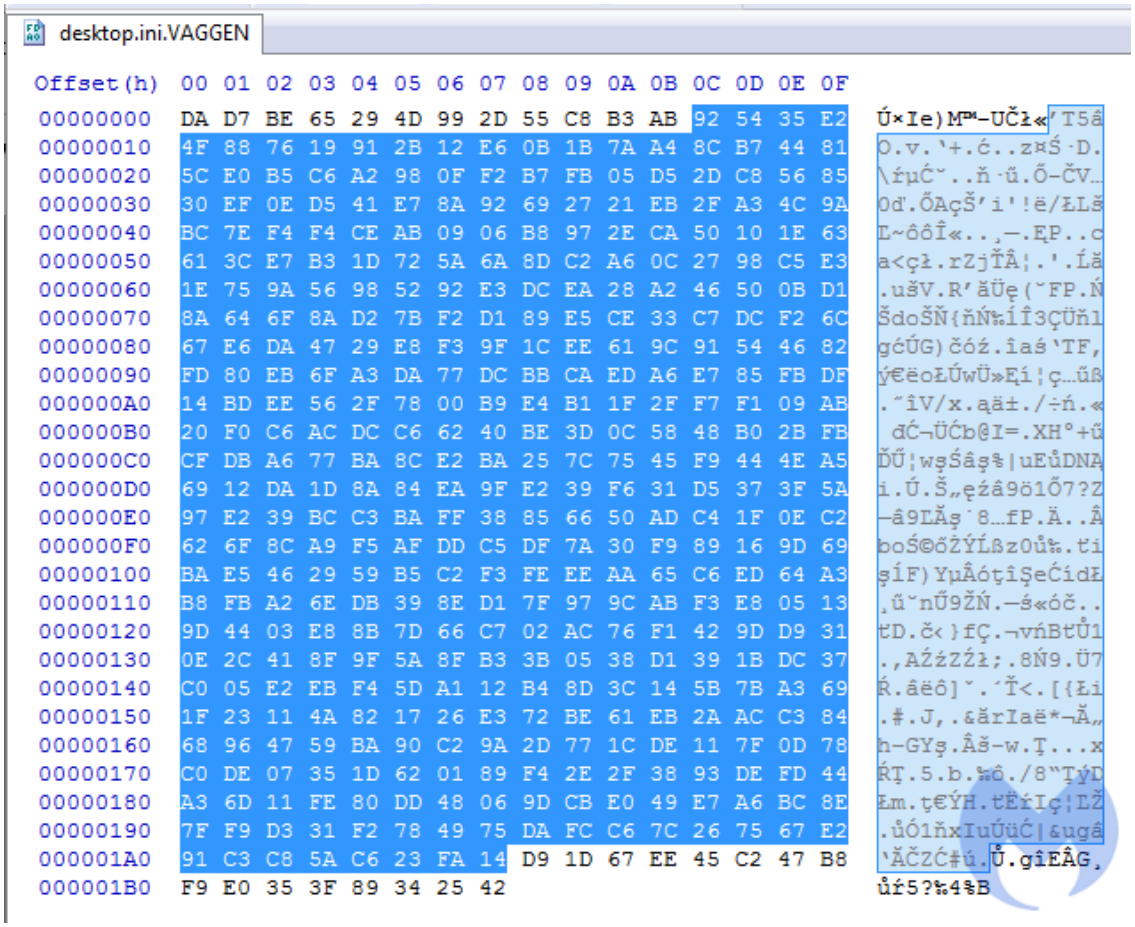
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

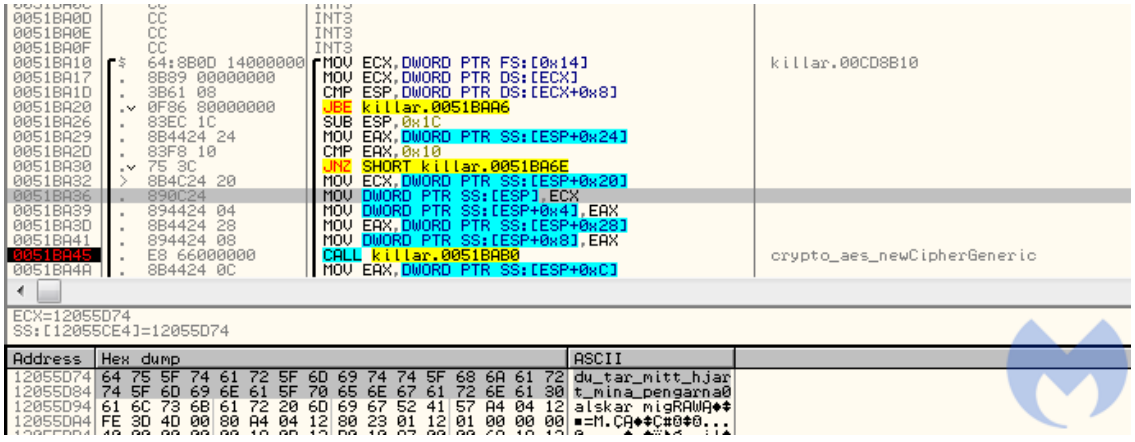


The content of the output file (with .VAGGEN extension) contains:

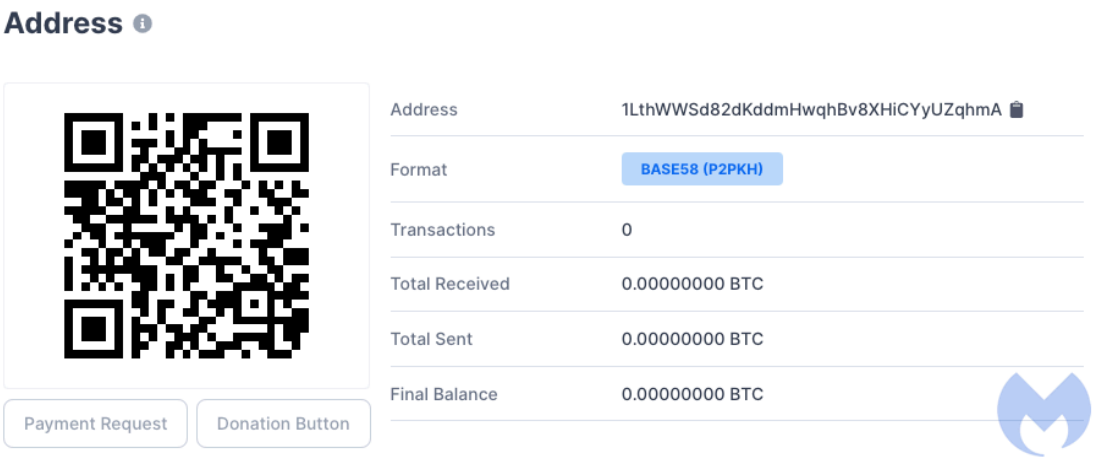
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

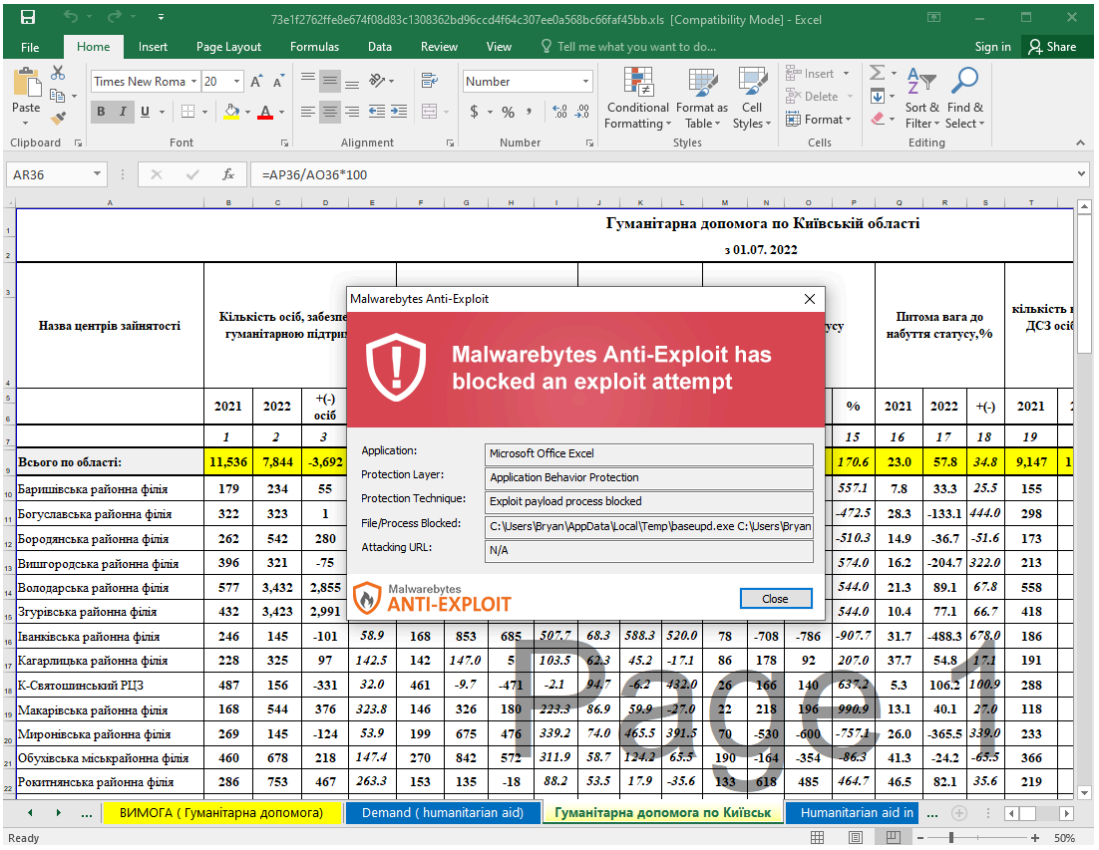
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

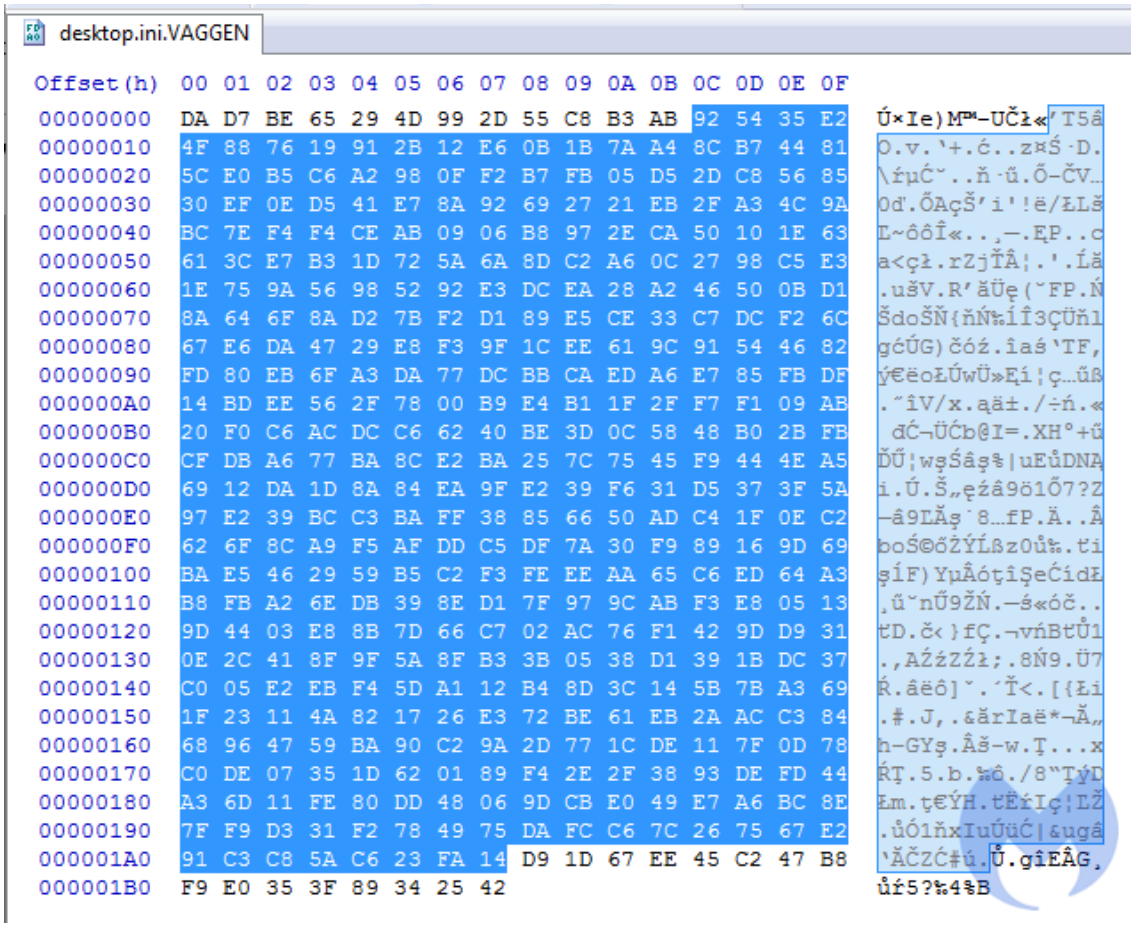
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

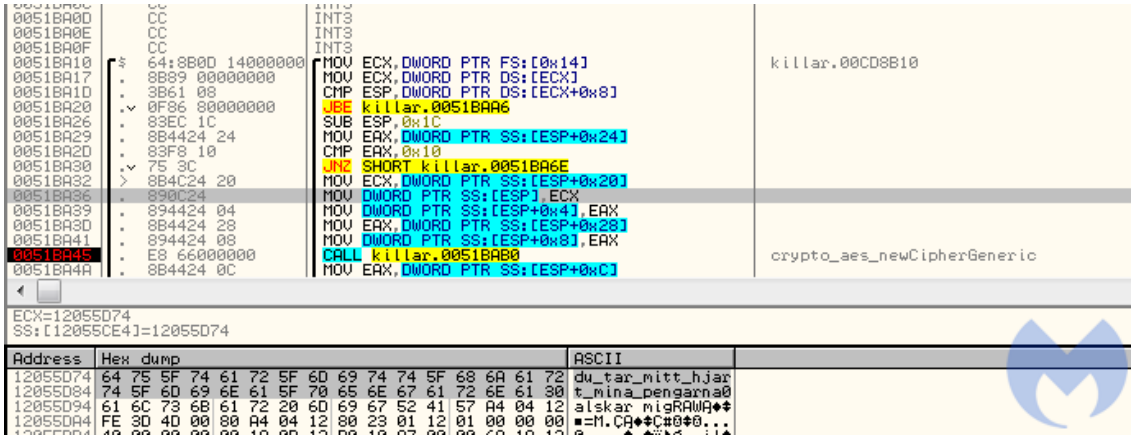
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

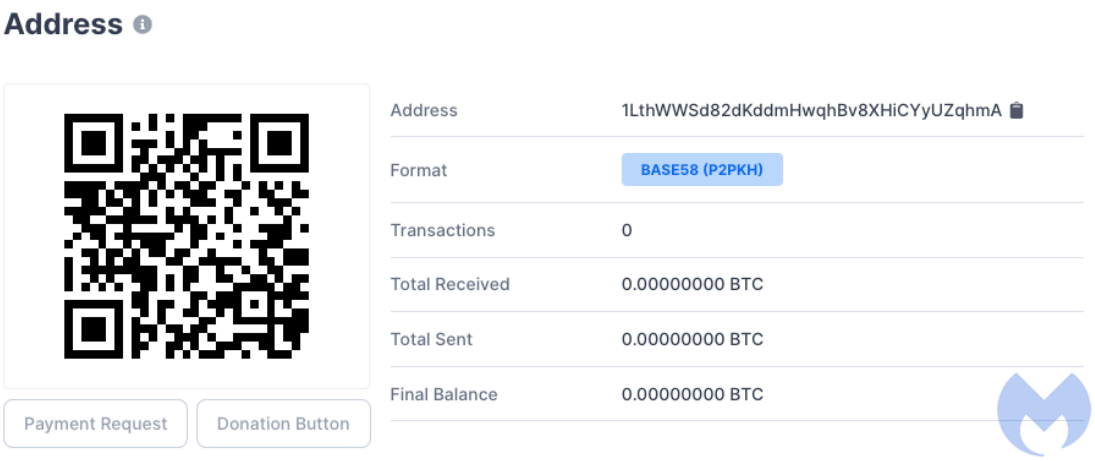
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

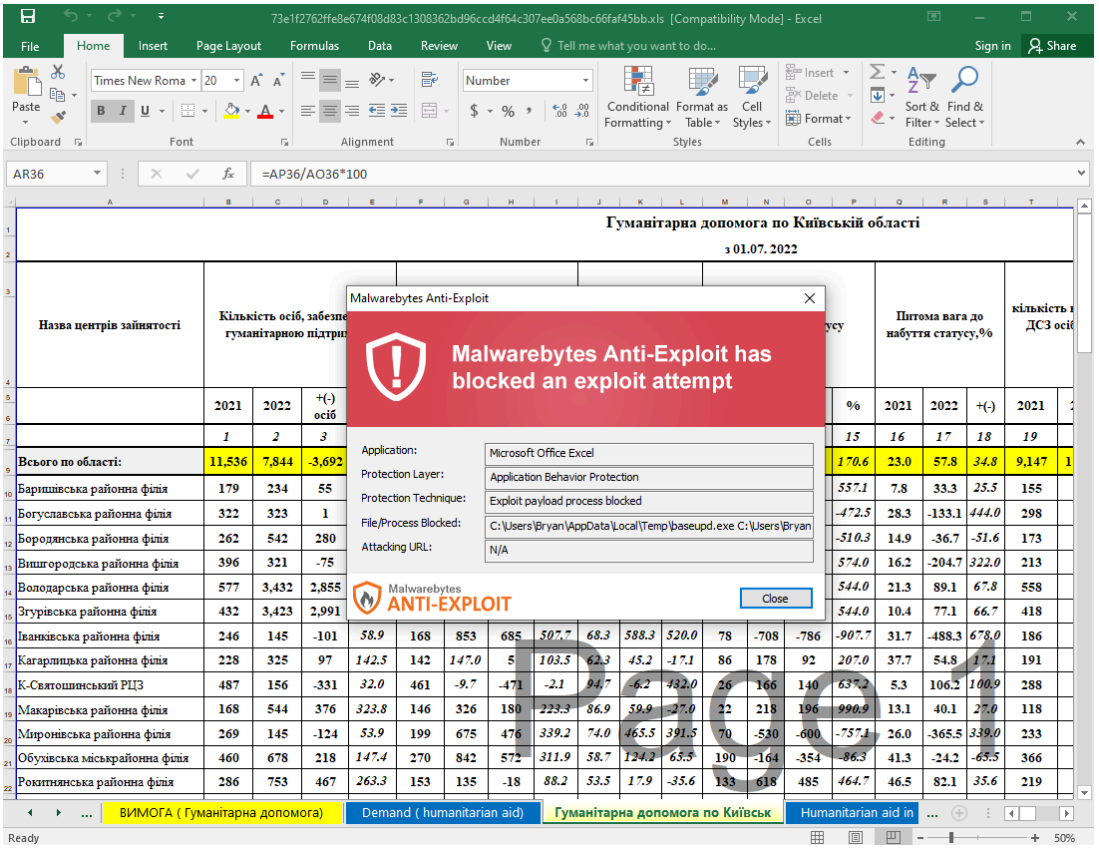
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

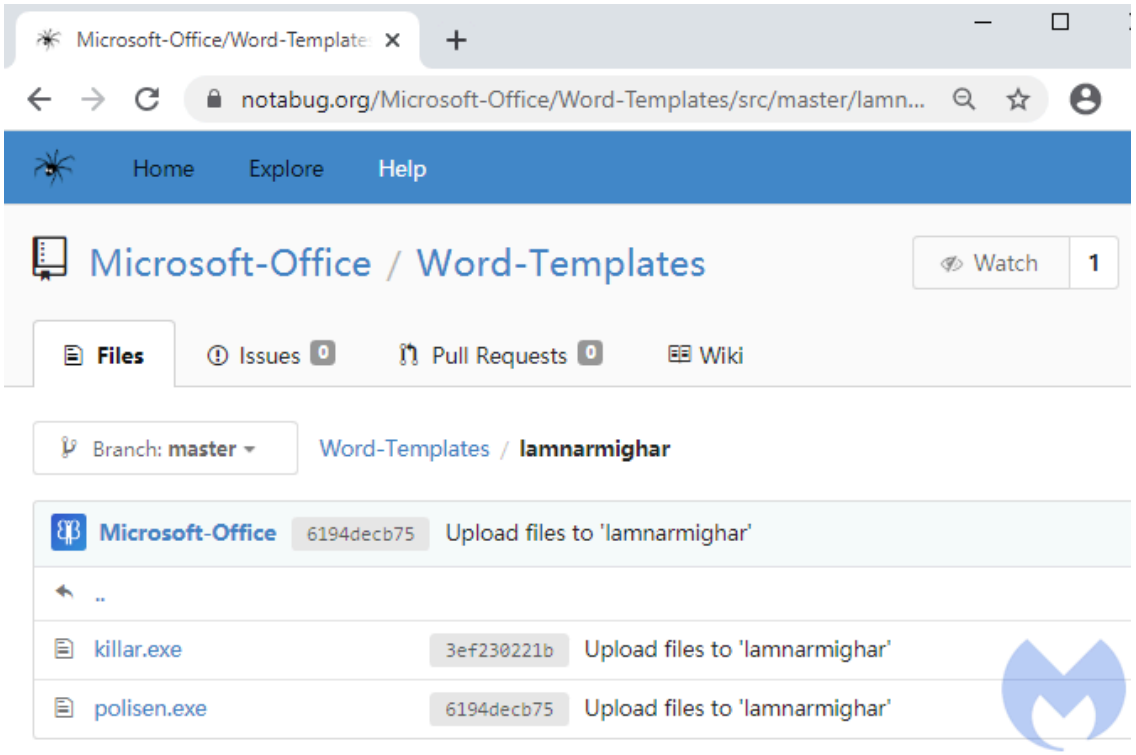
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



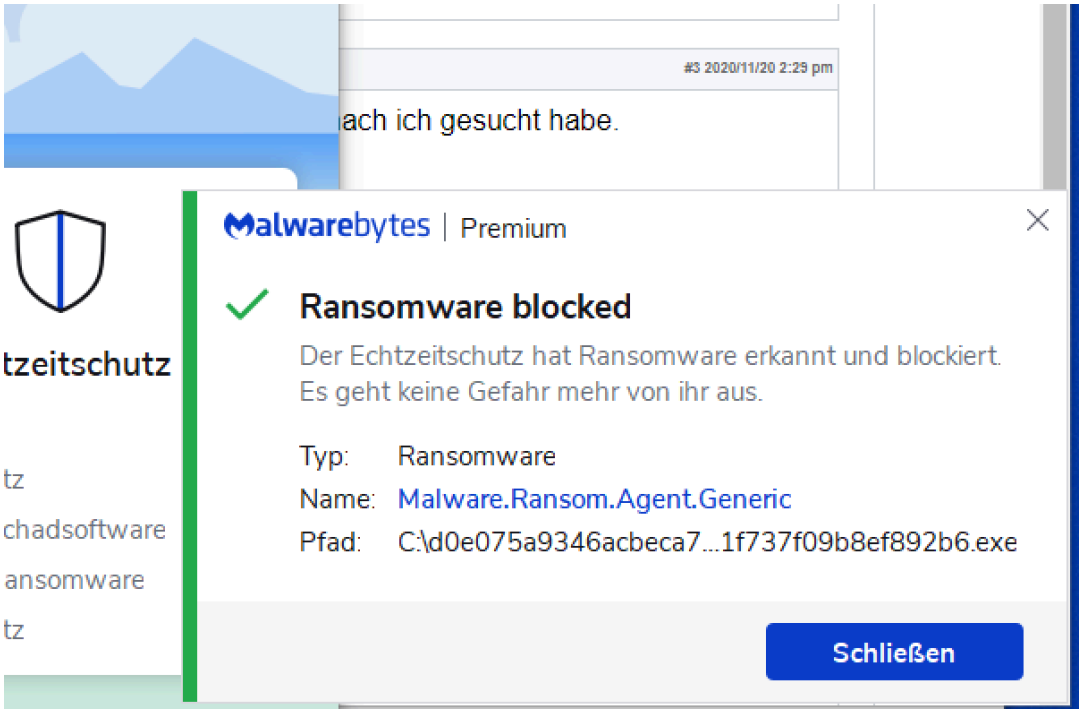
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

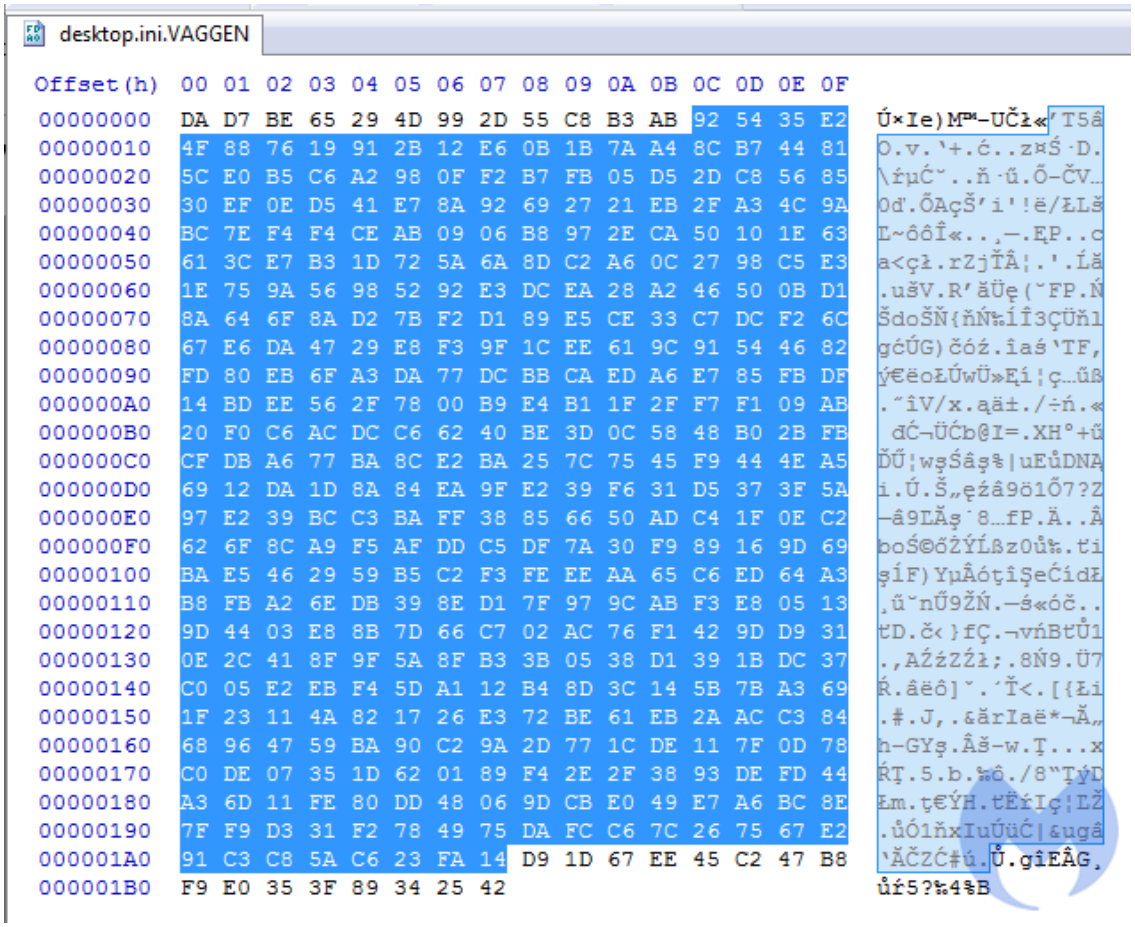
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

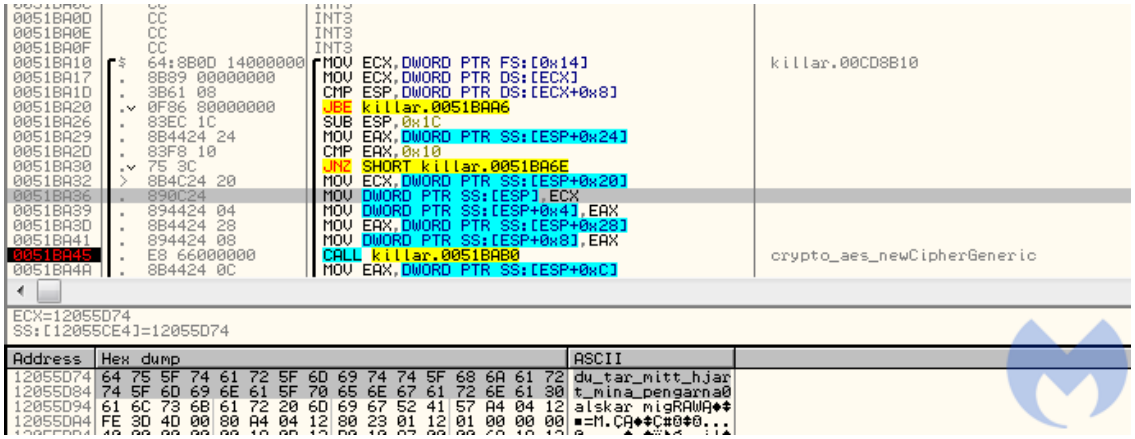


The content of the output file (with .VAGGEN extension) contains:

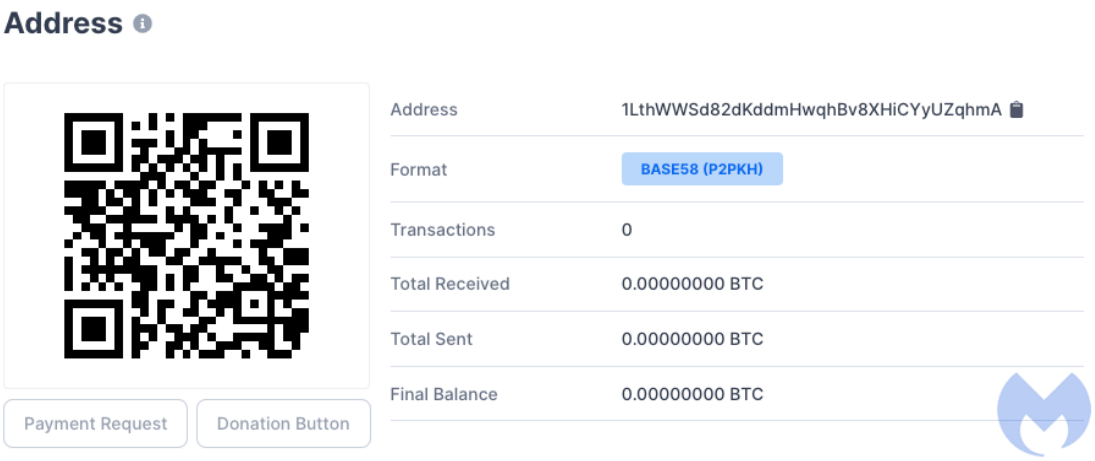
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

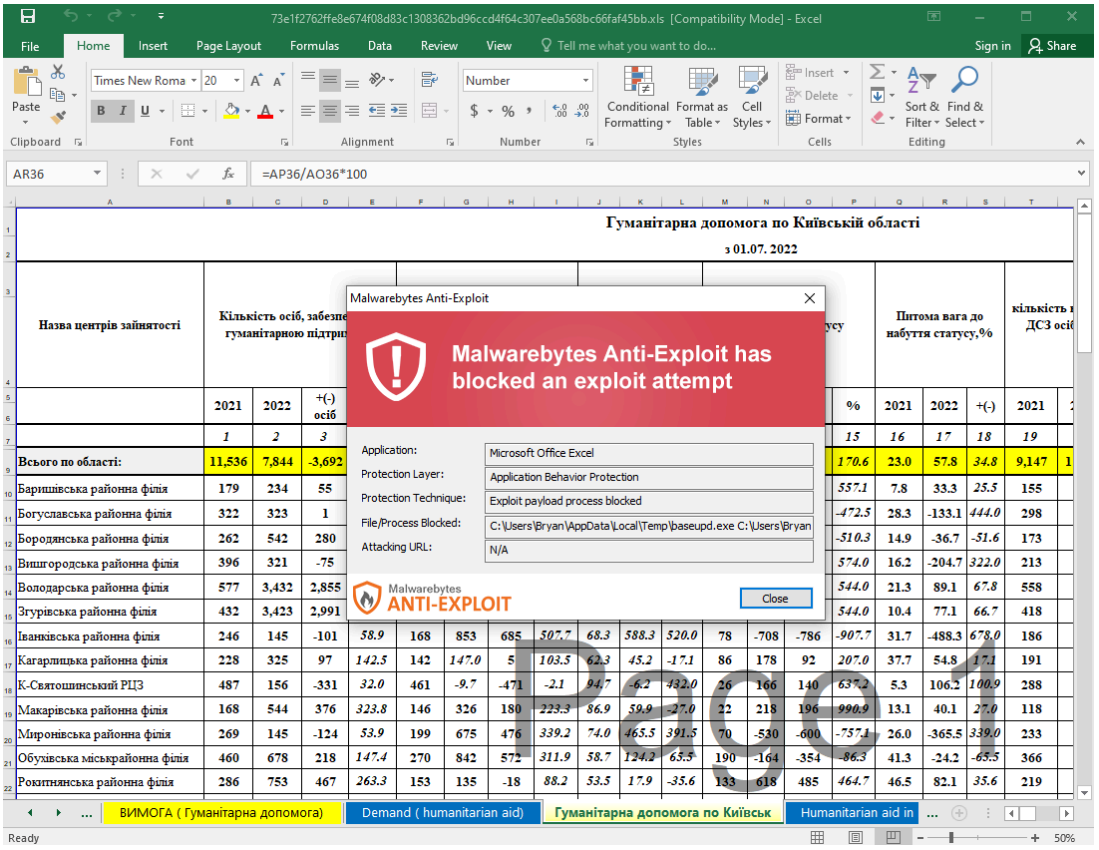
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

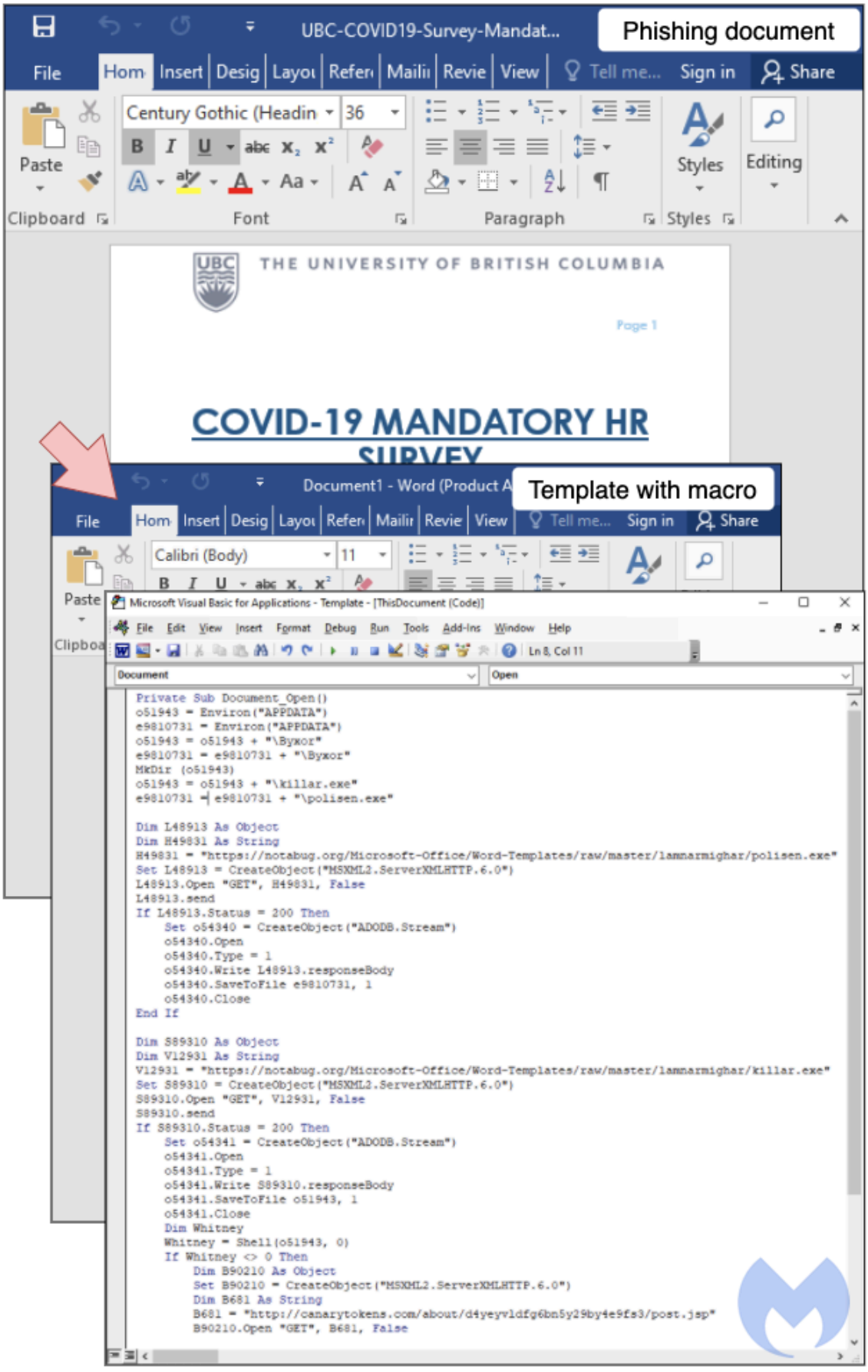
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

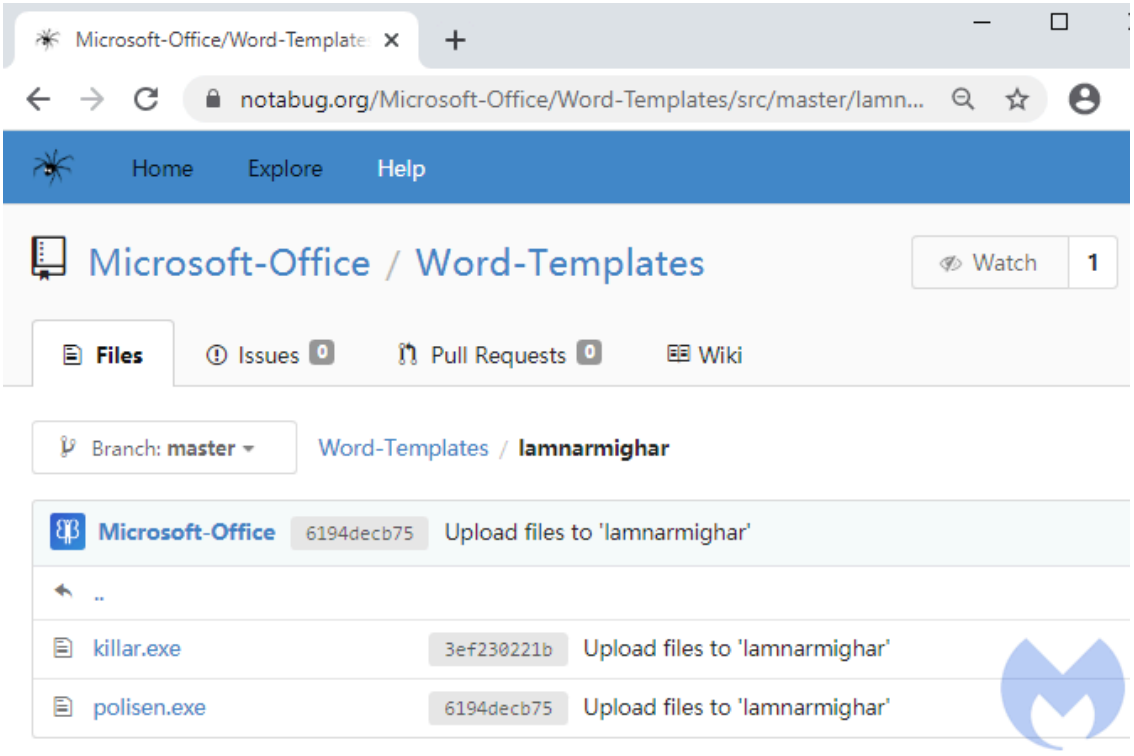
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



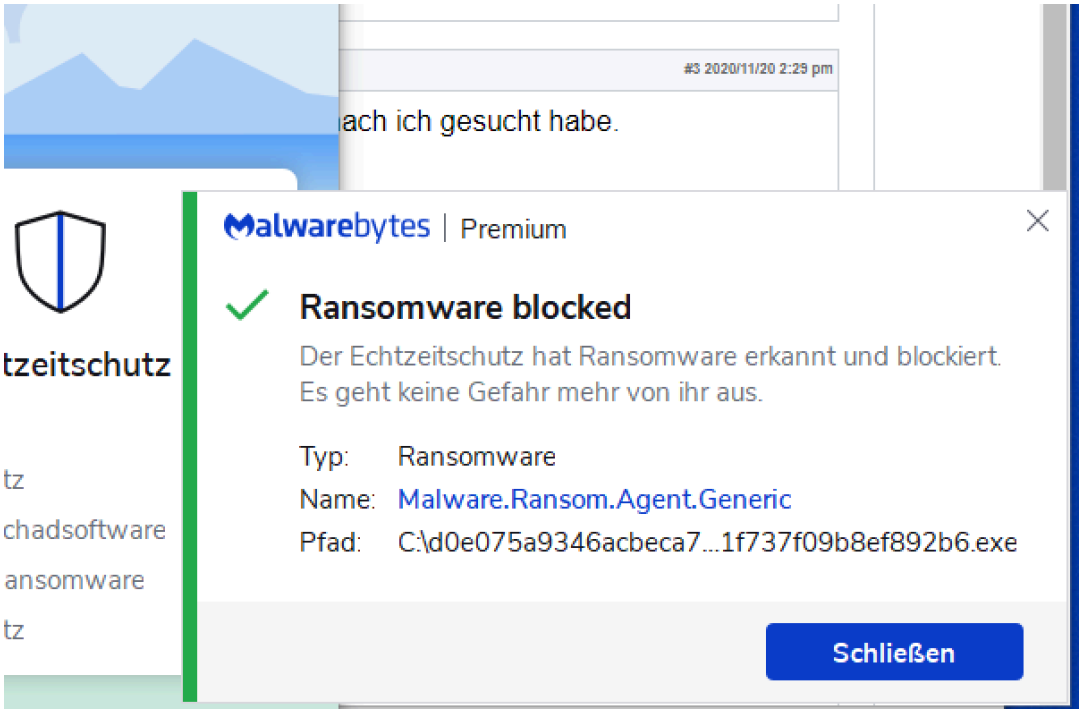
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

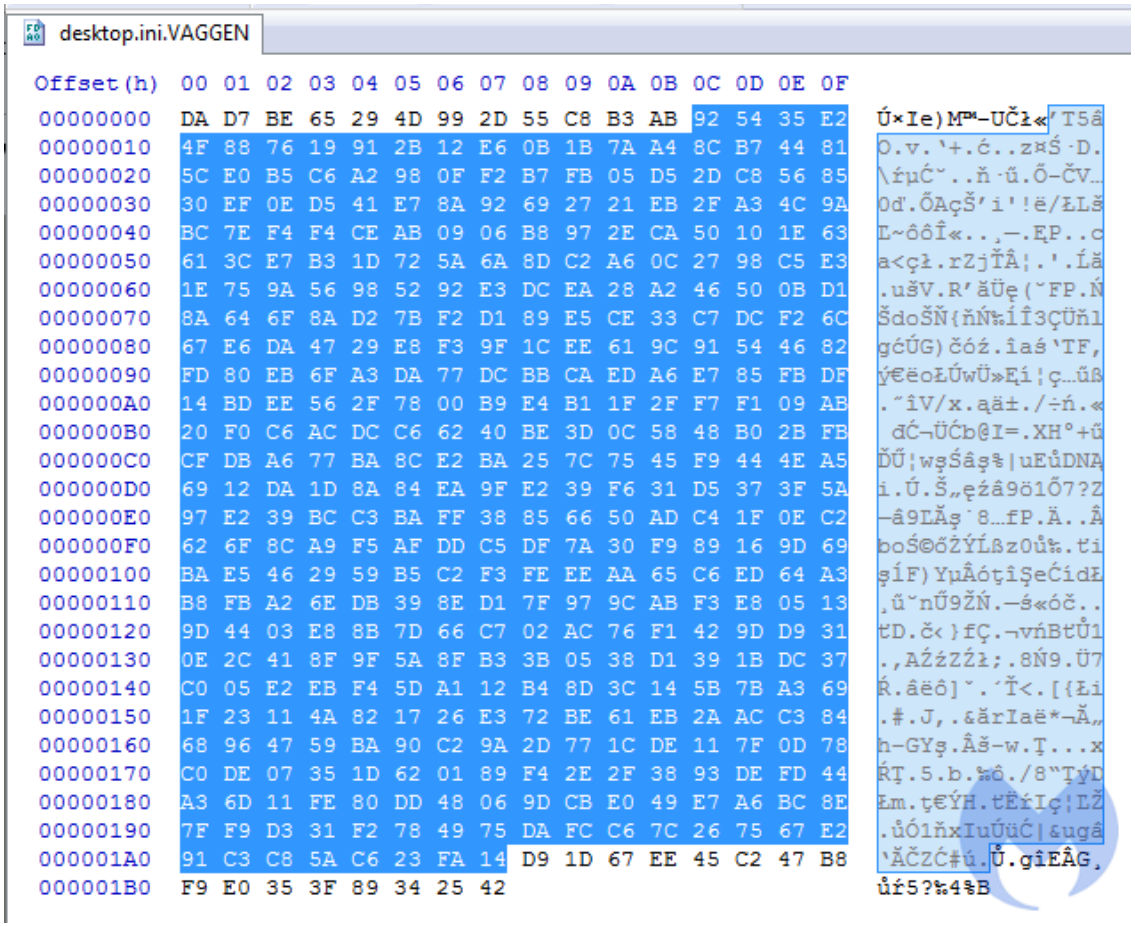
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

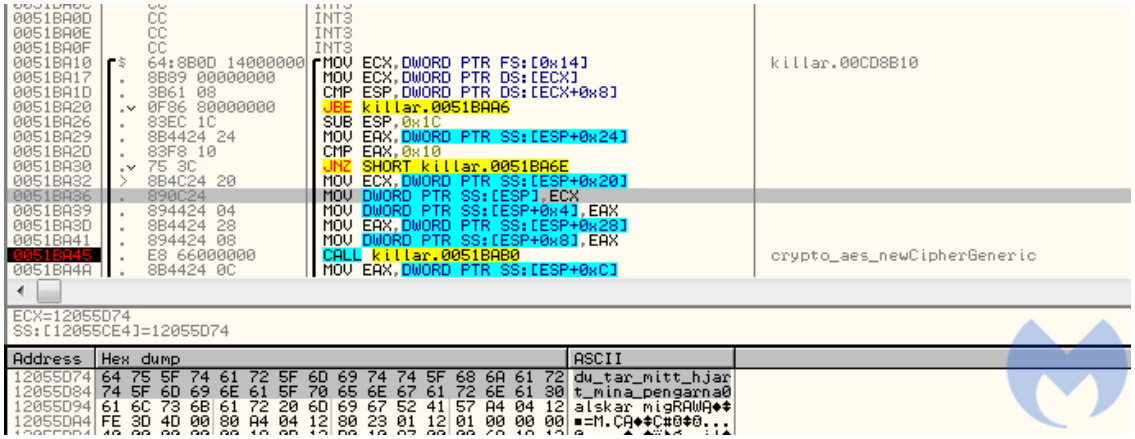


The content of the output file (with .VAGGEN extension) contains:

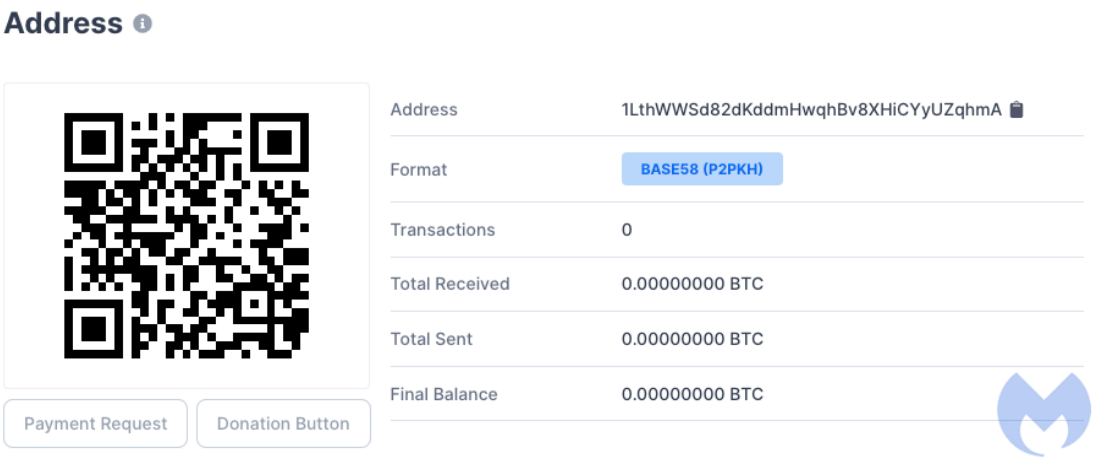
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

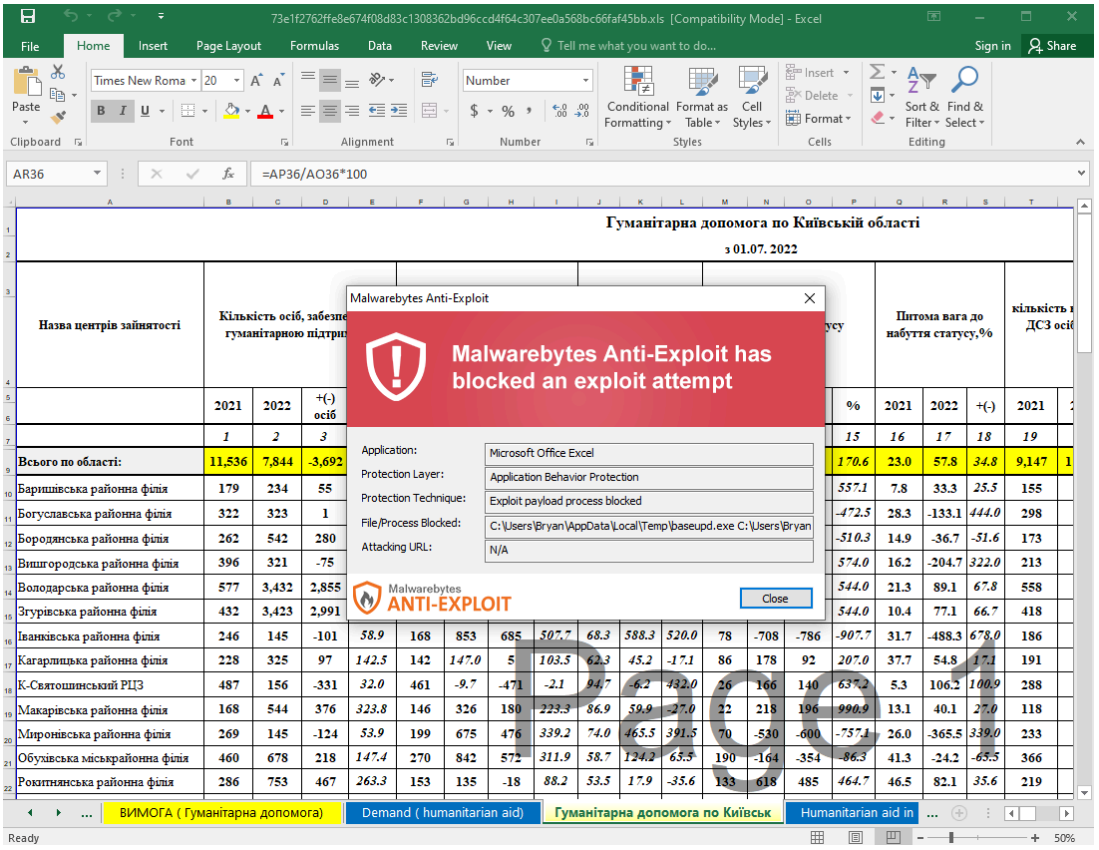
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

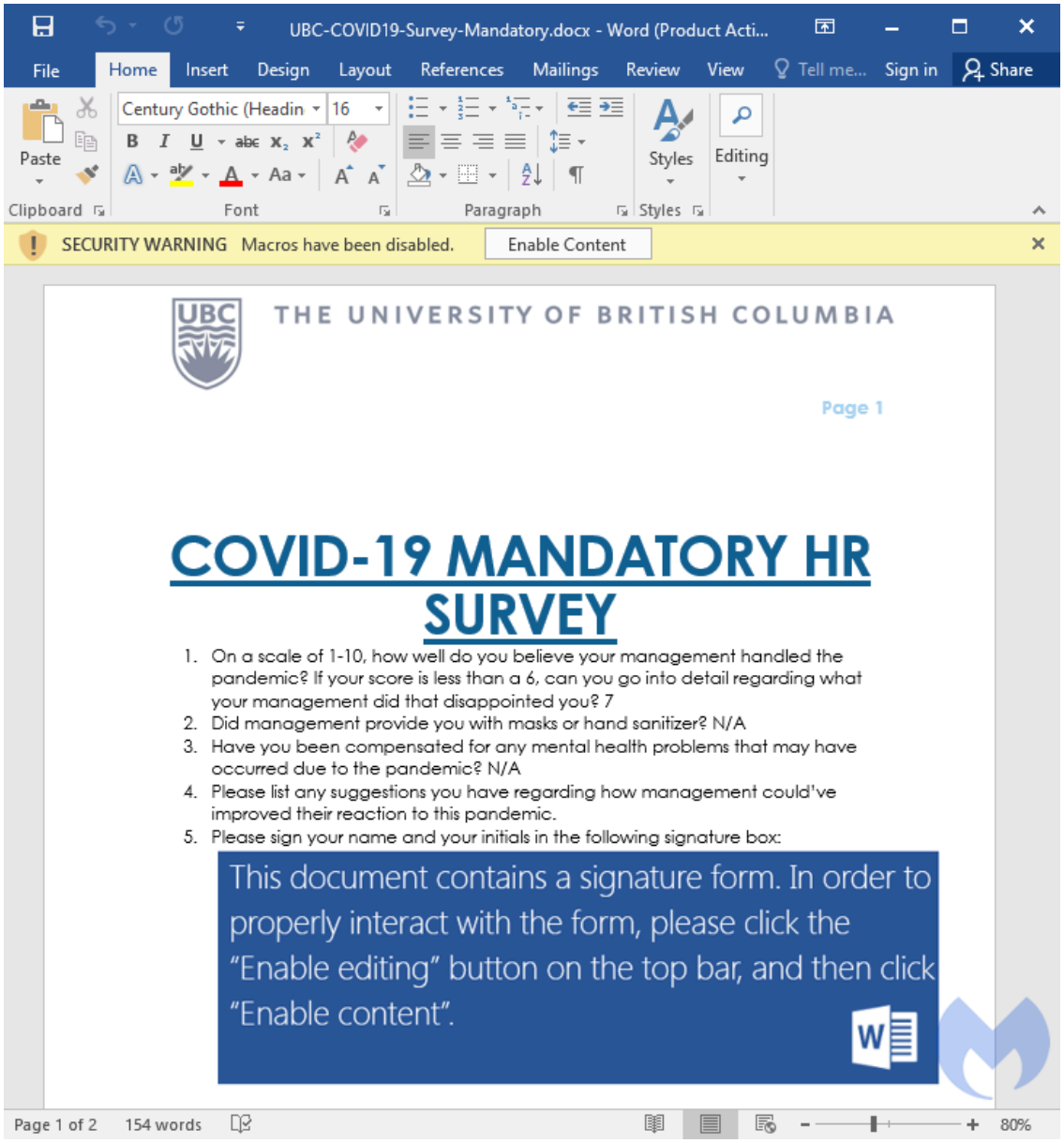
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

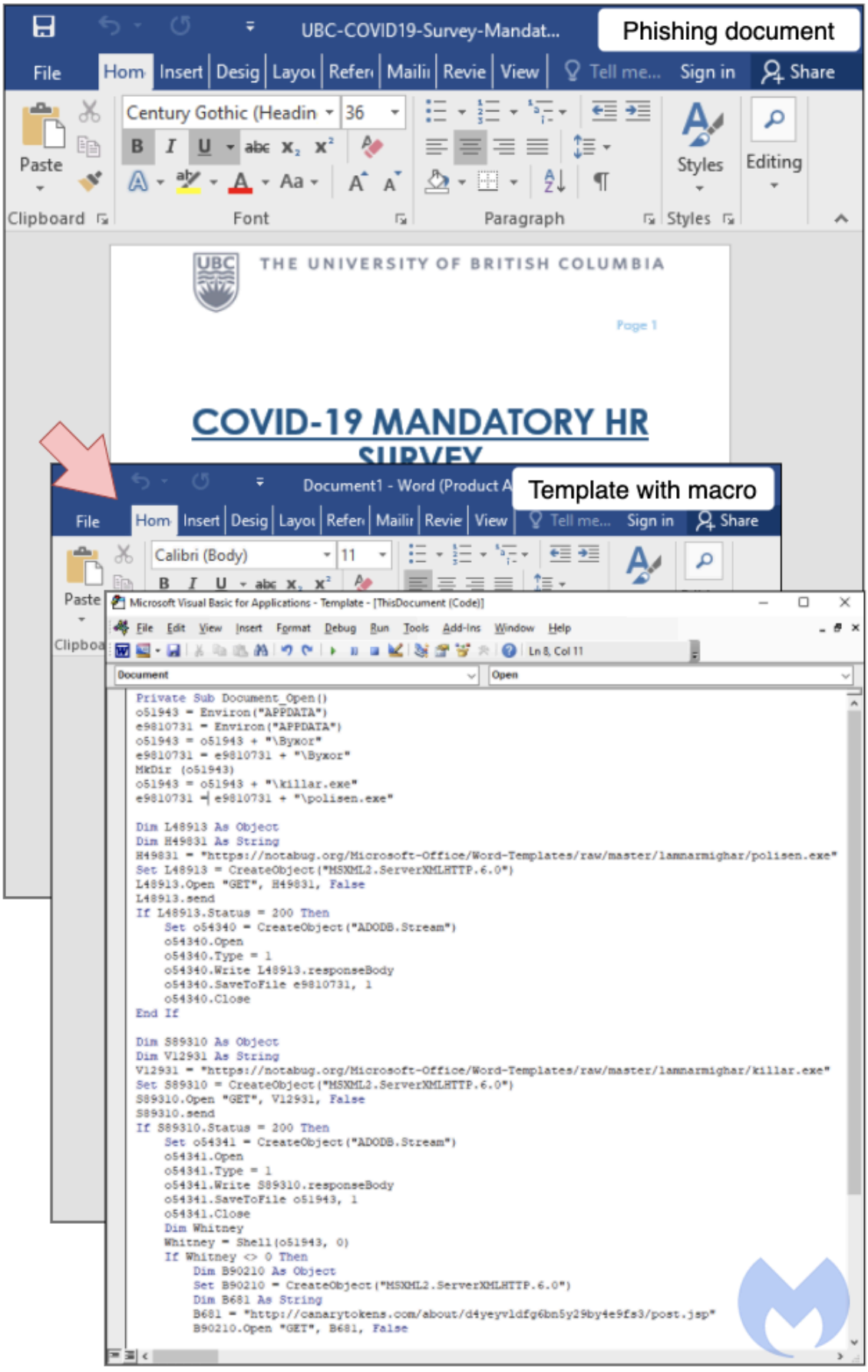
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

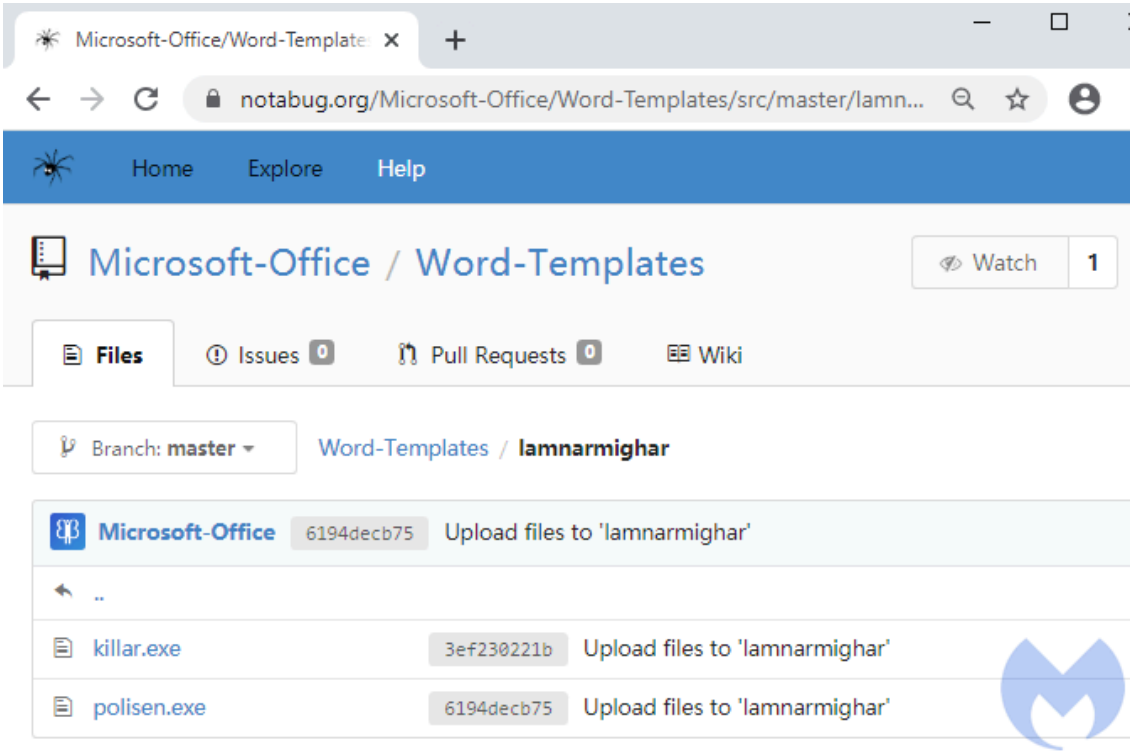
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



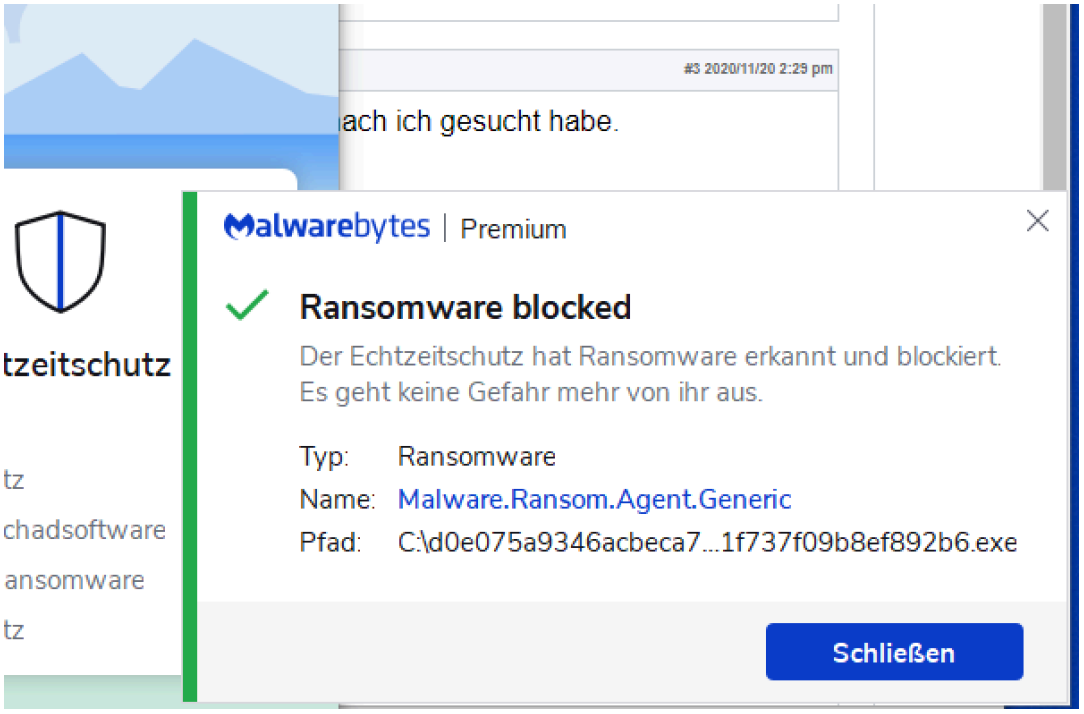
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

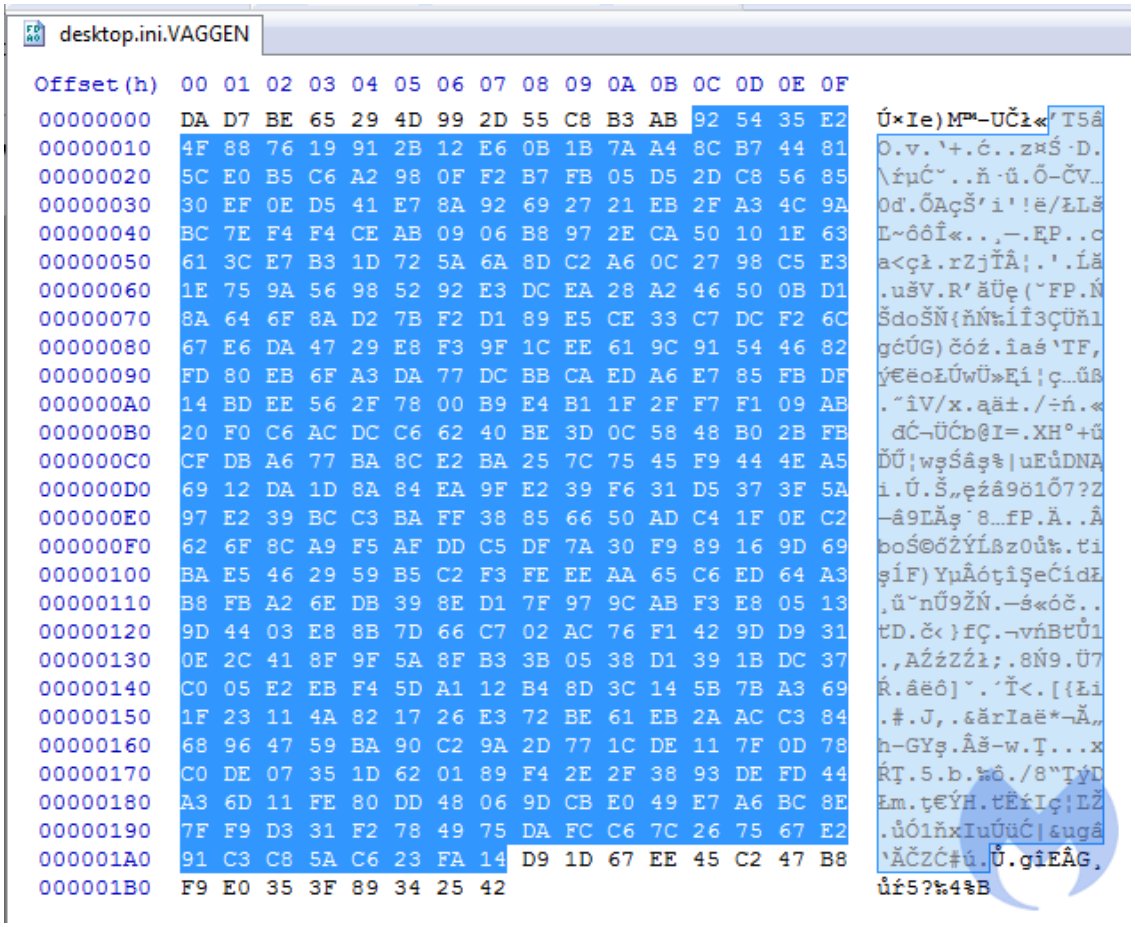
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



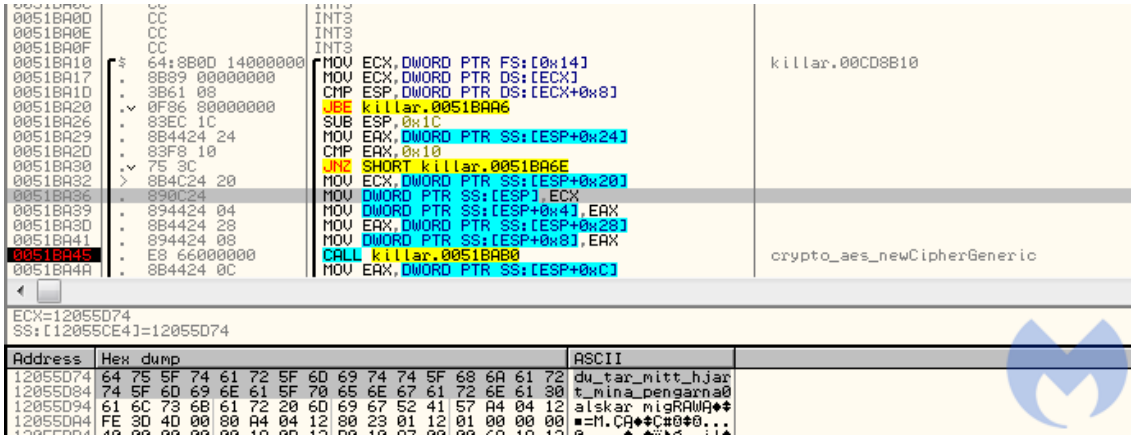
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

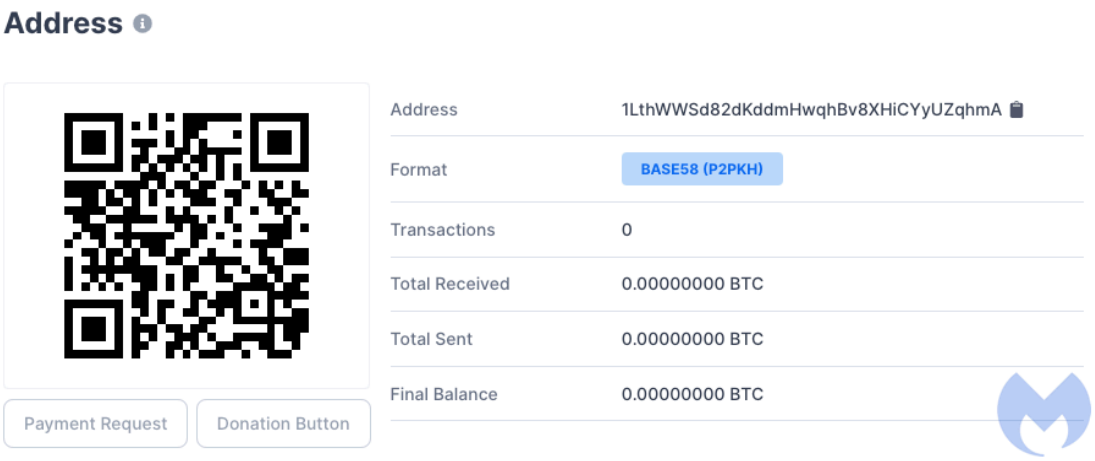
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

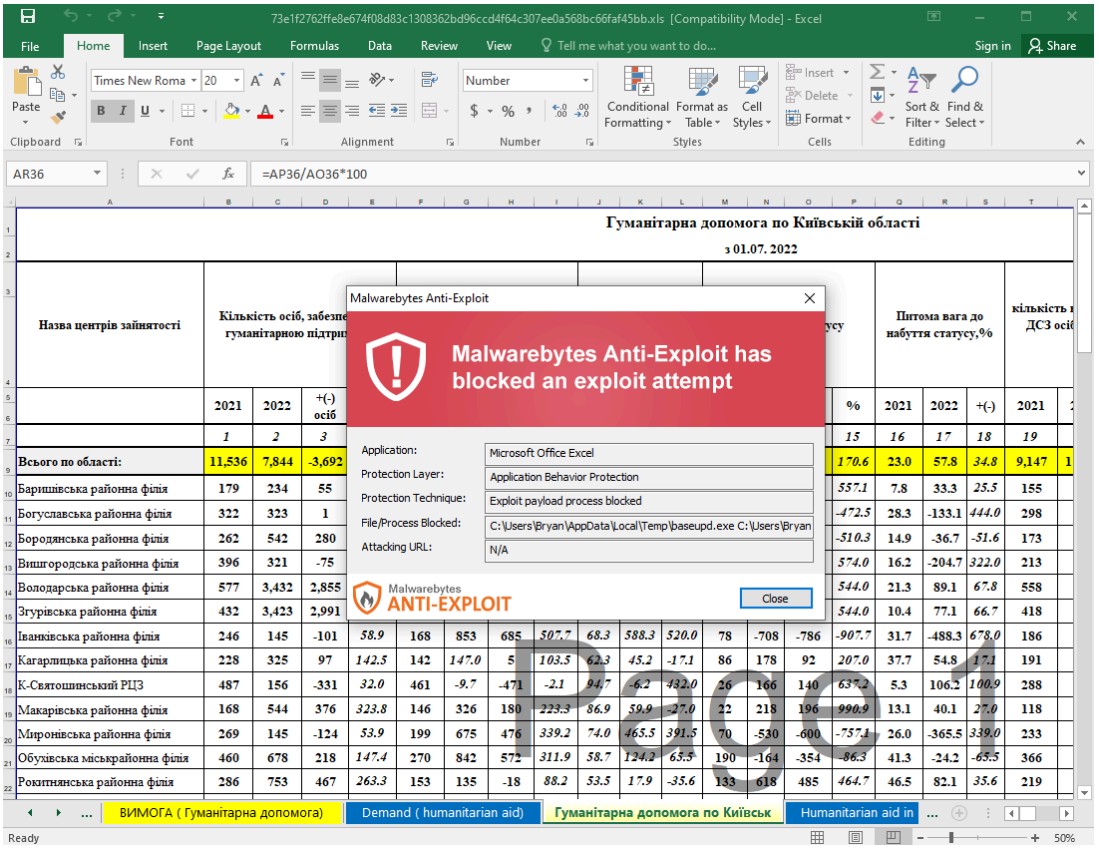
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

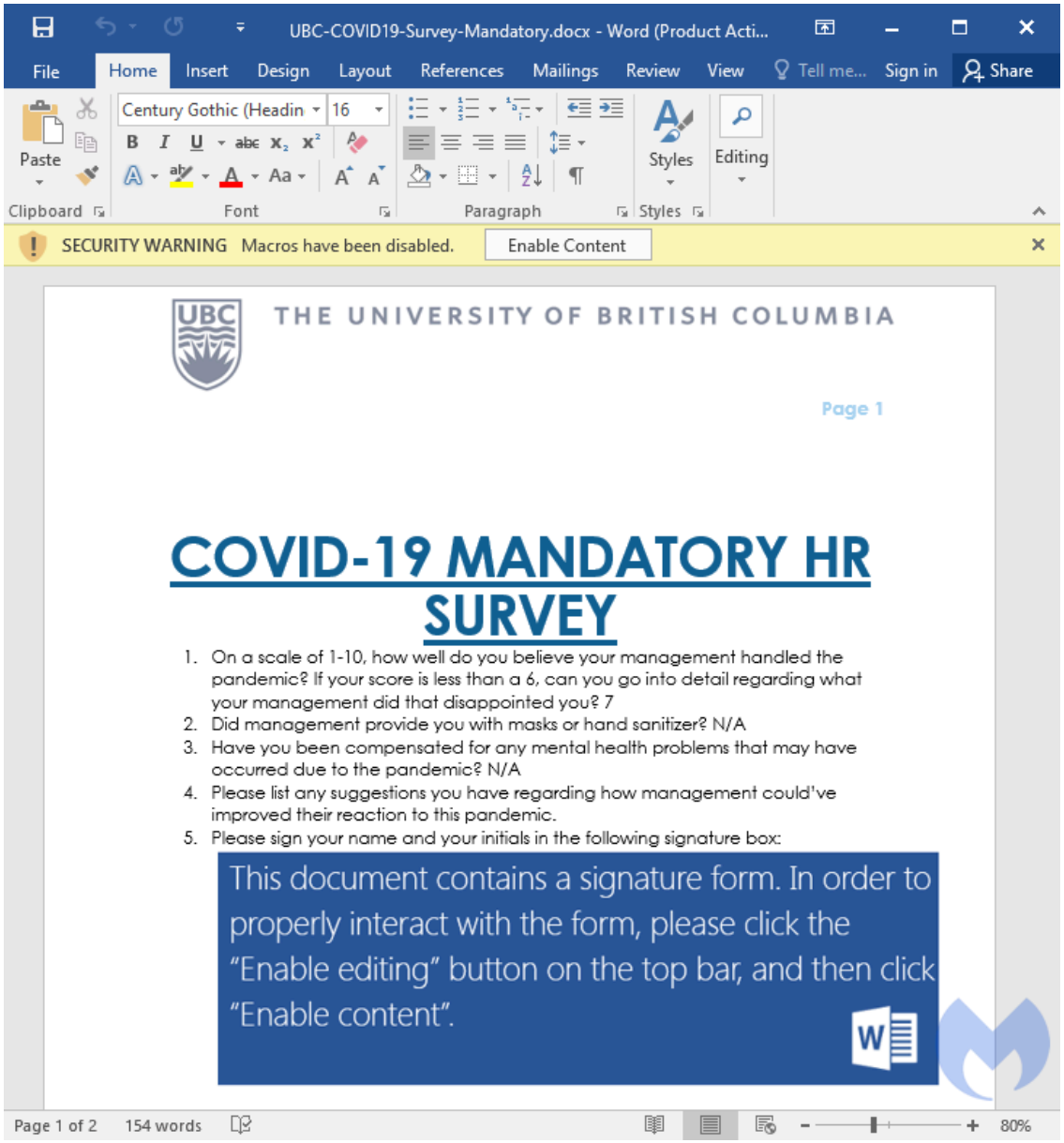
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

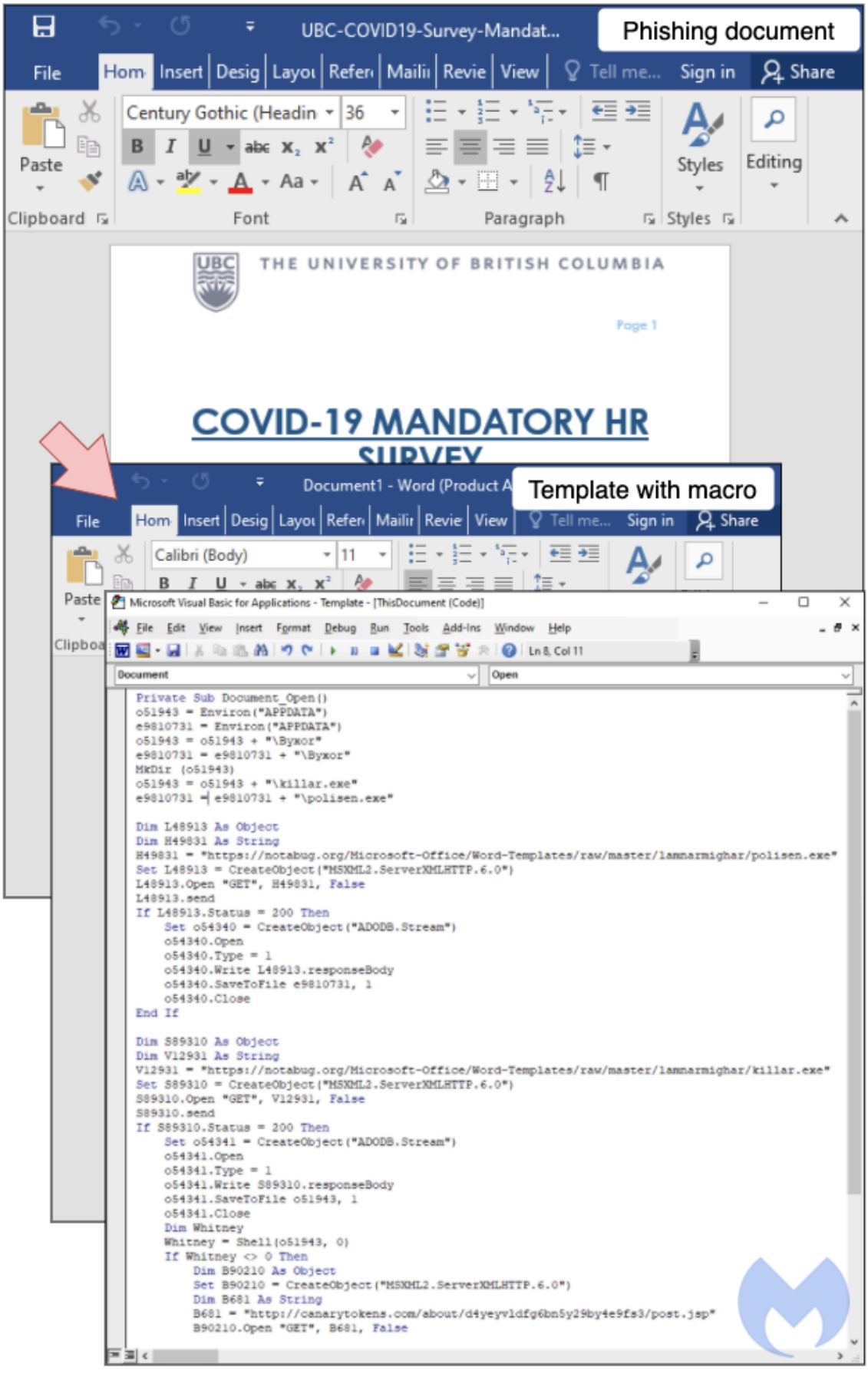
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

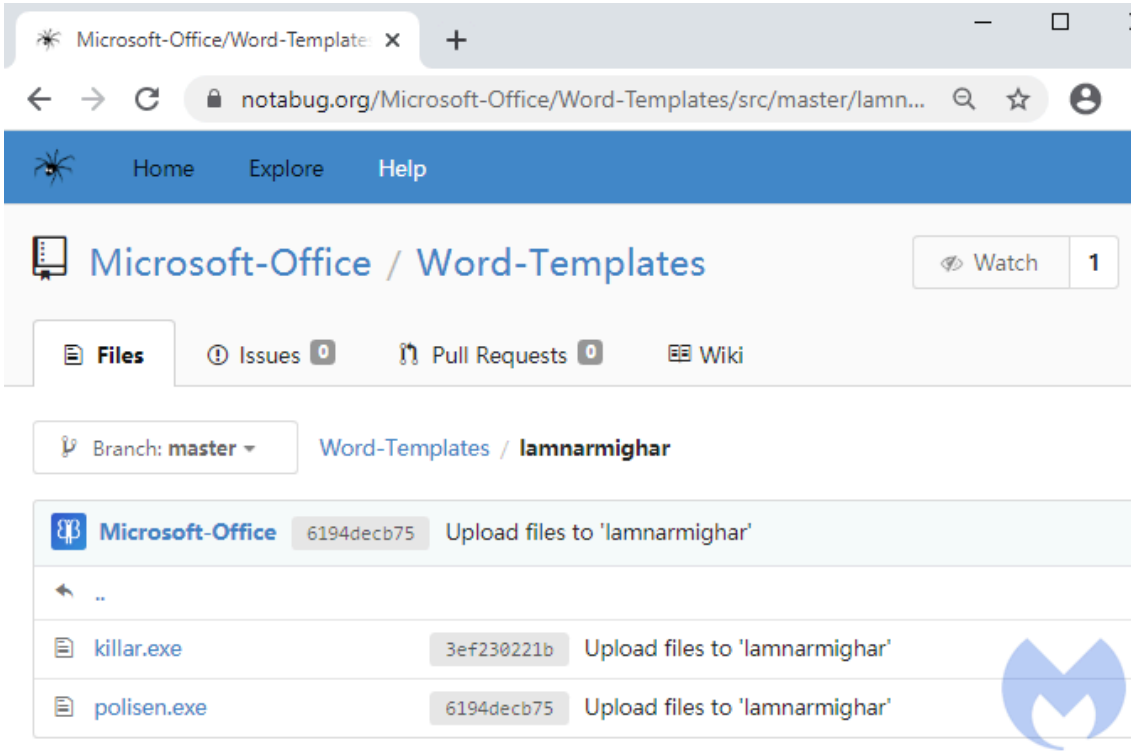
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



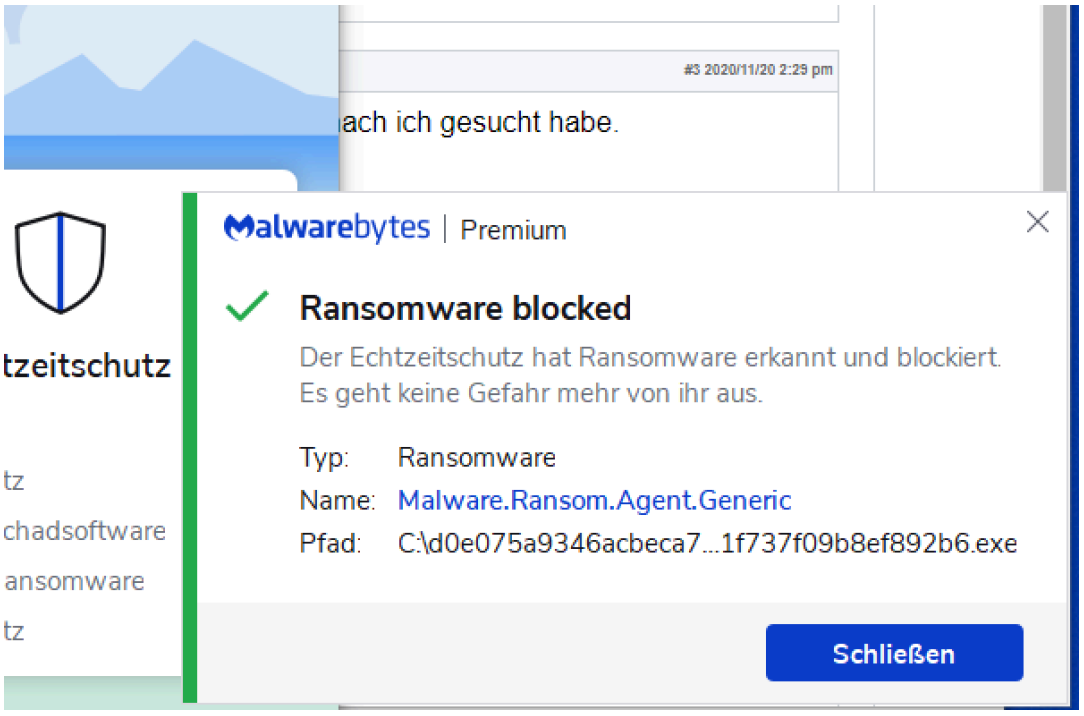
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

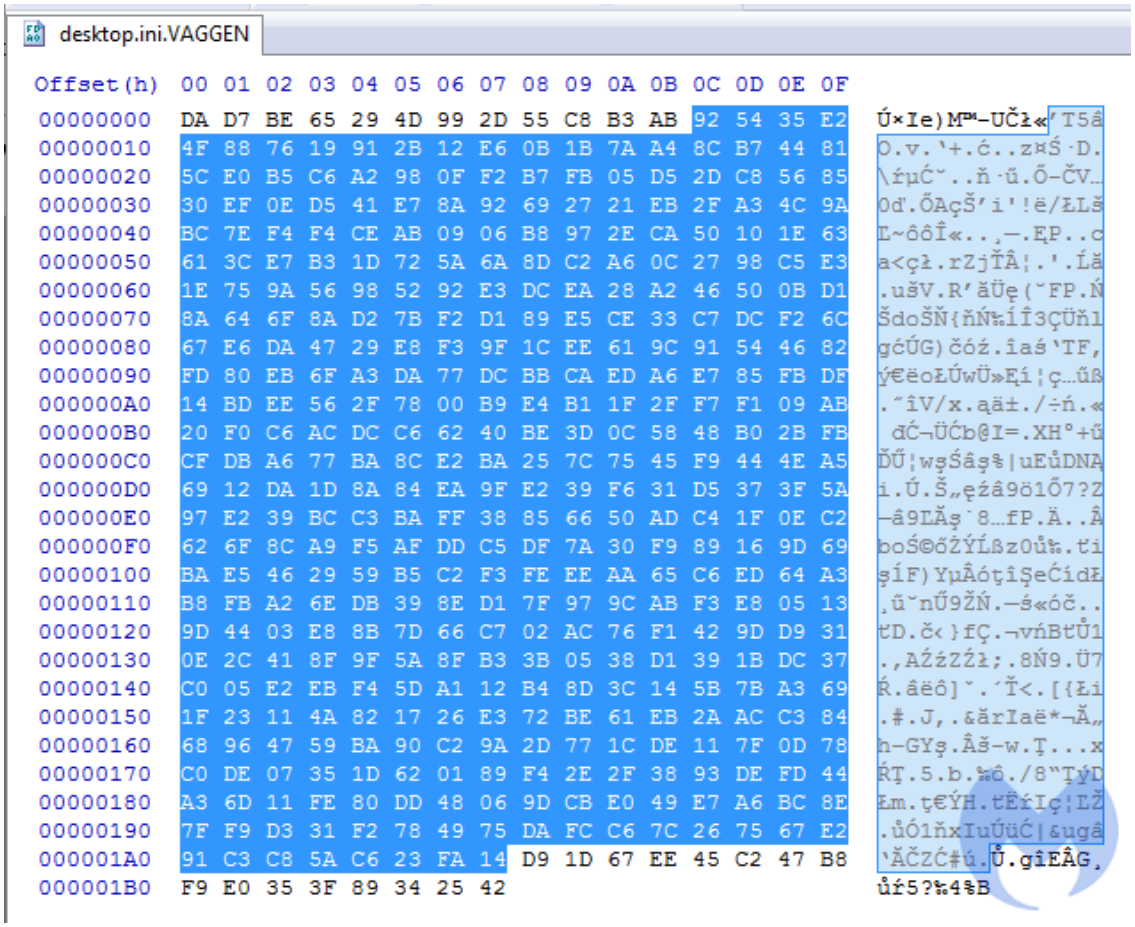
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

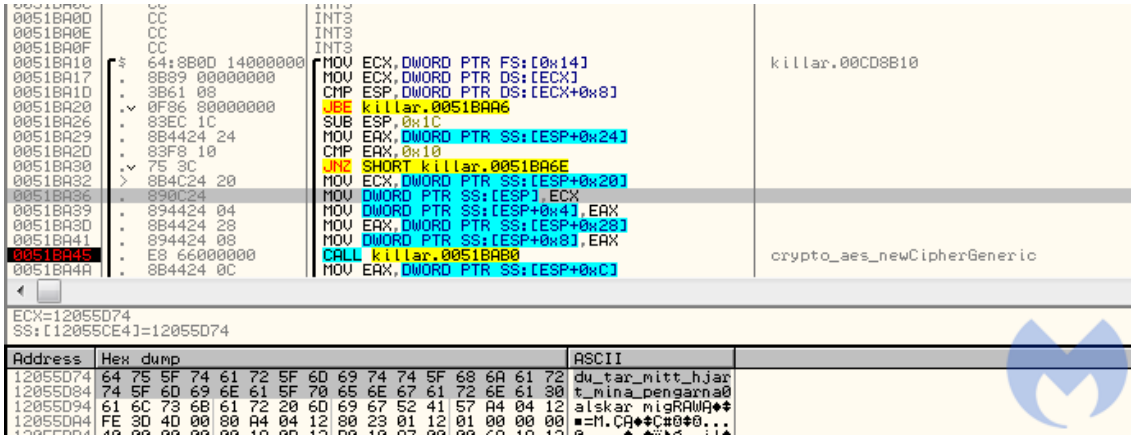


The content of the output file (with .VAGGEN extension) contains:

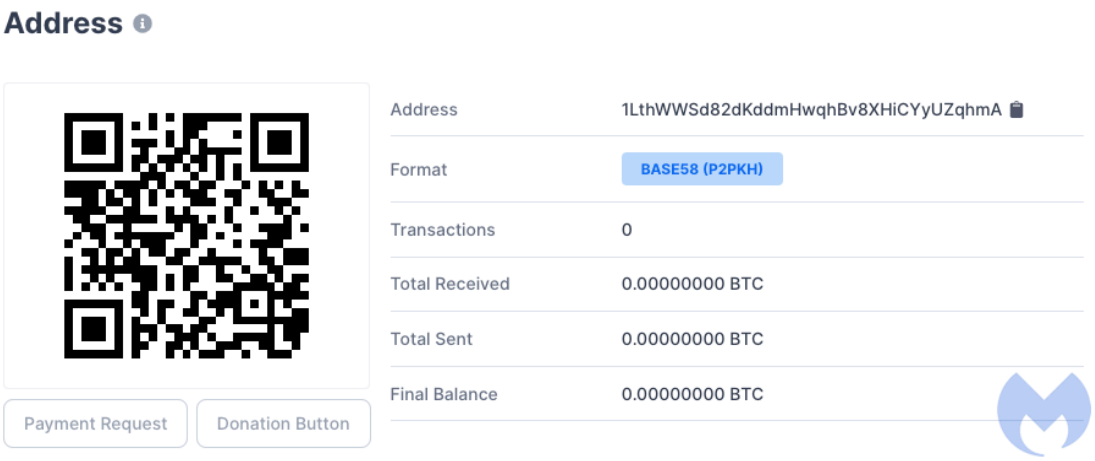
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

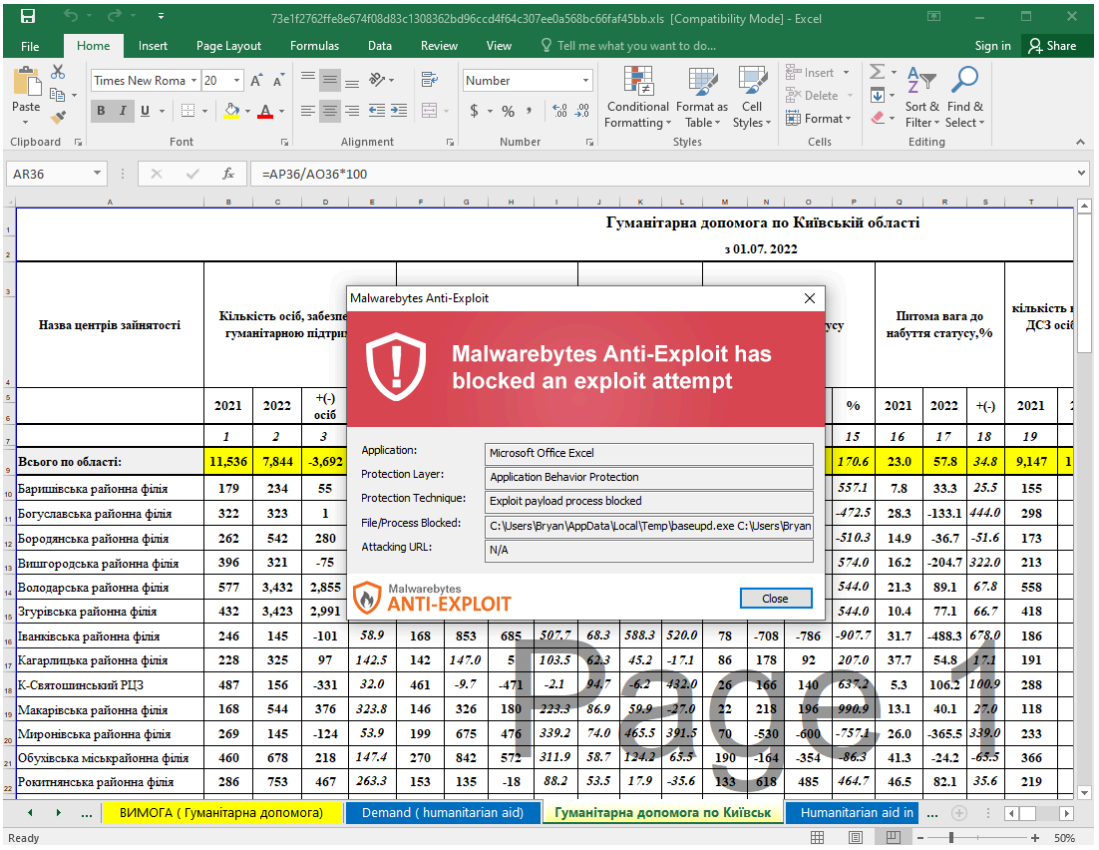
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

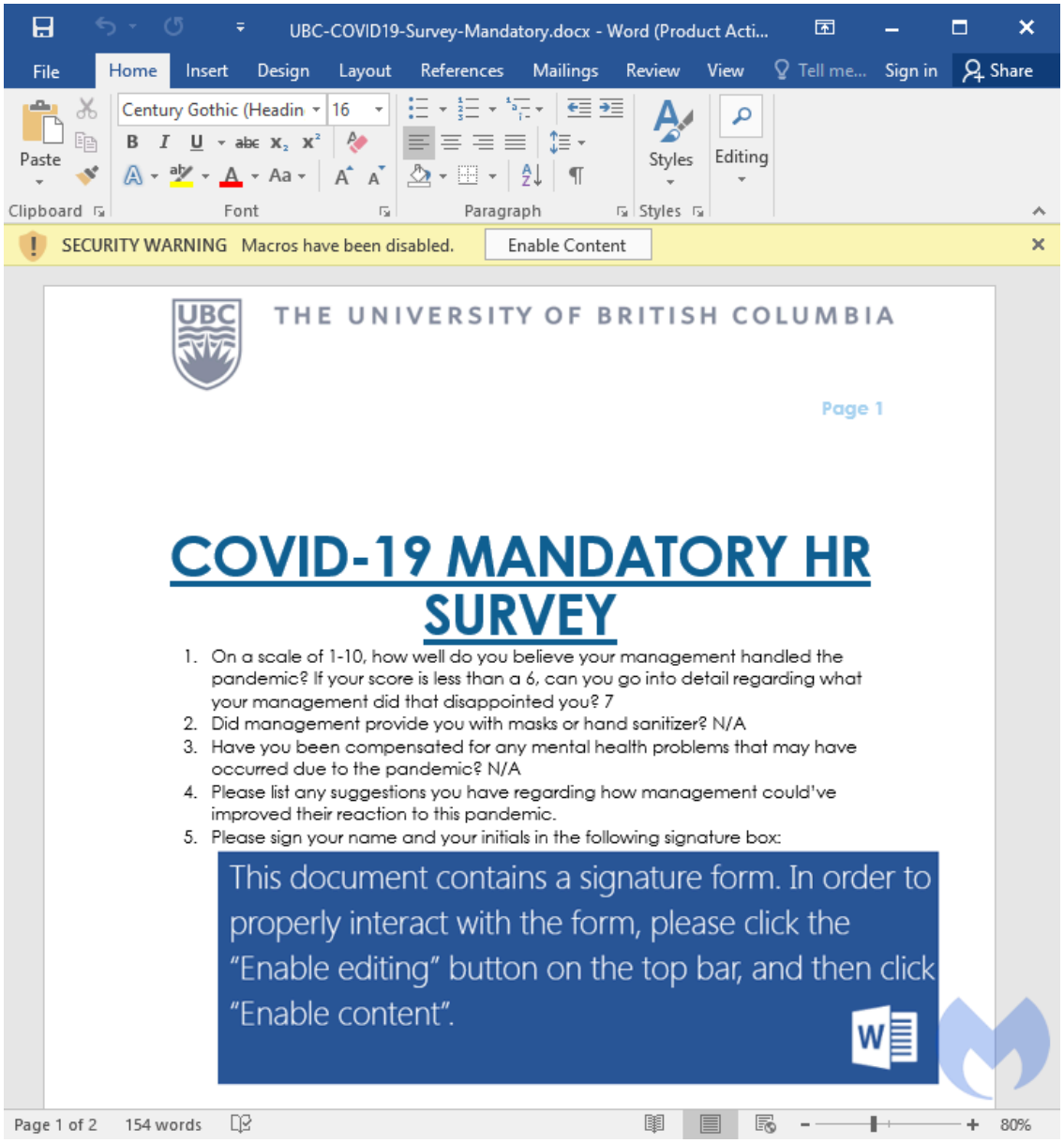
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

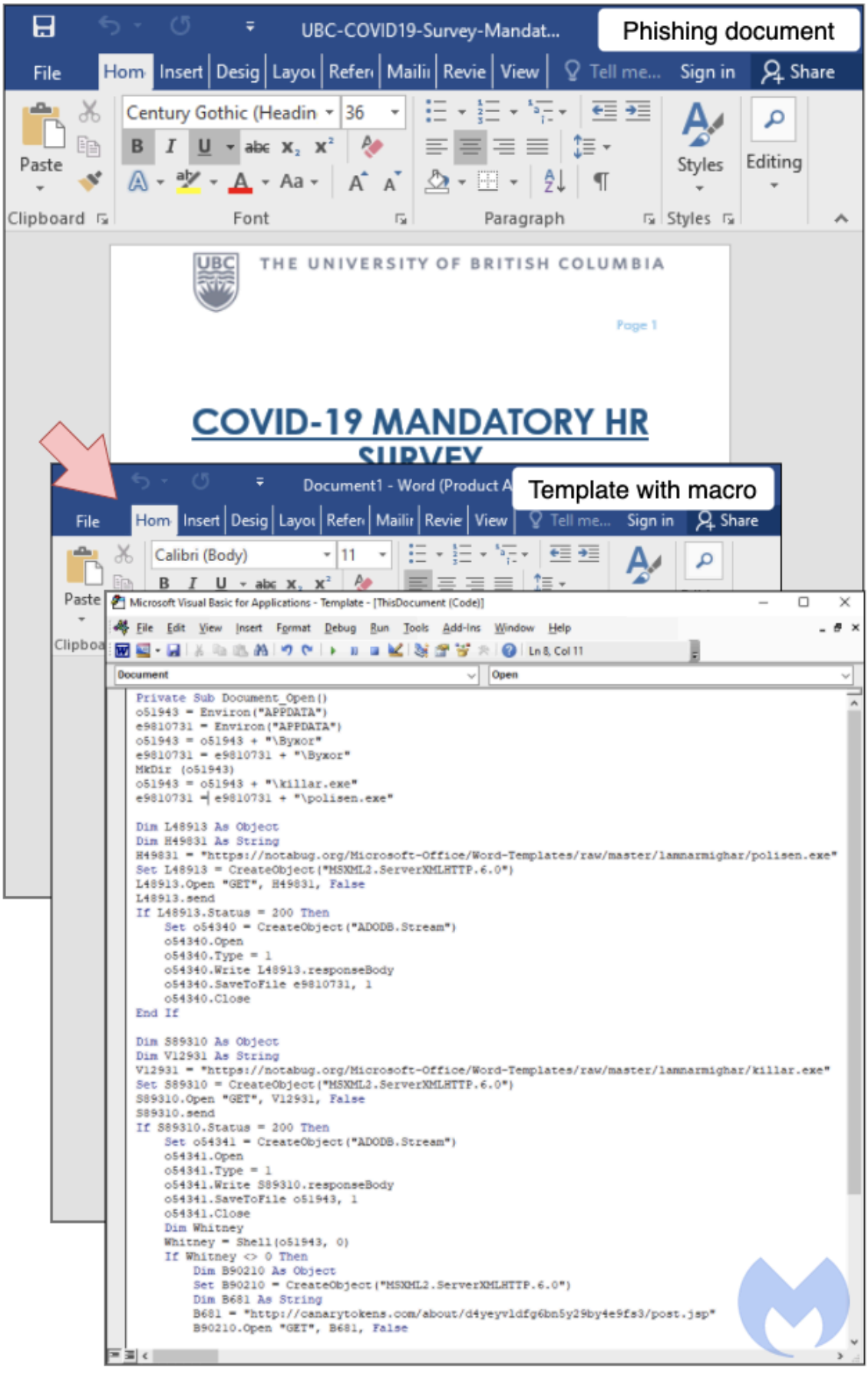
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

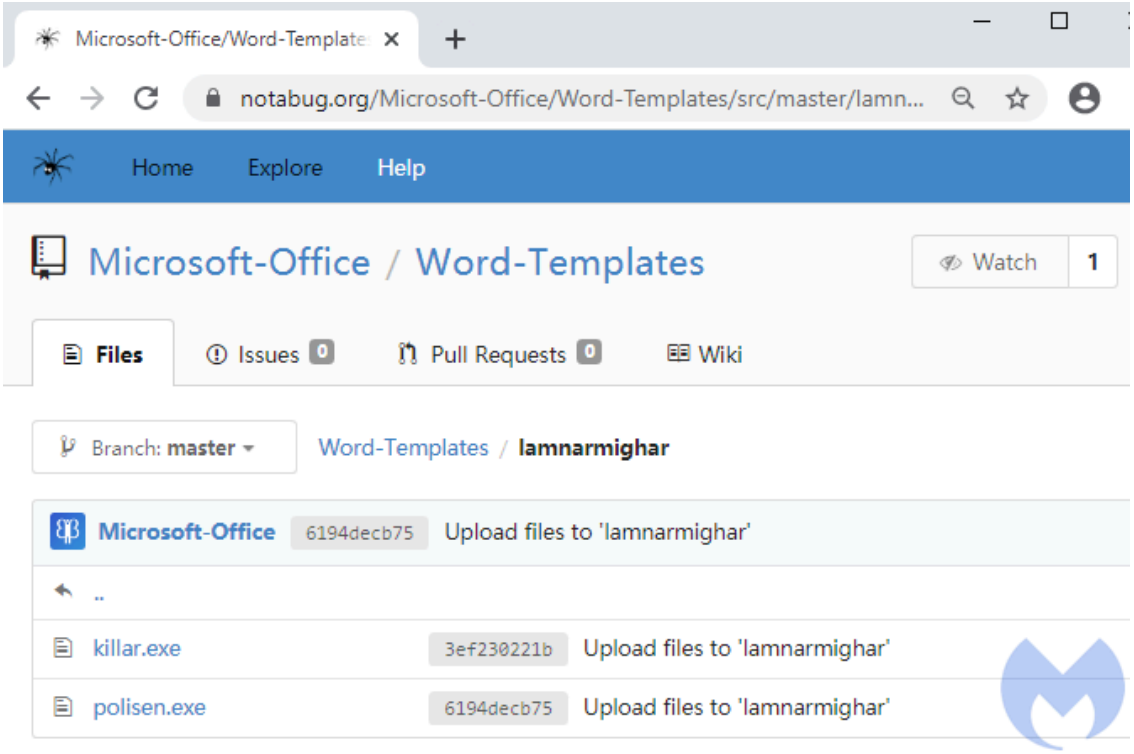
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



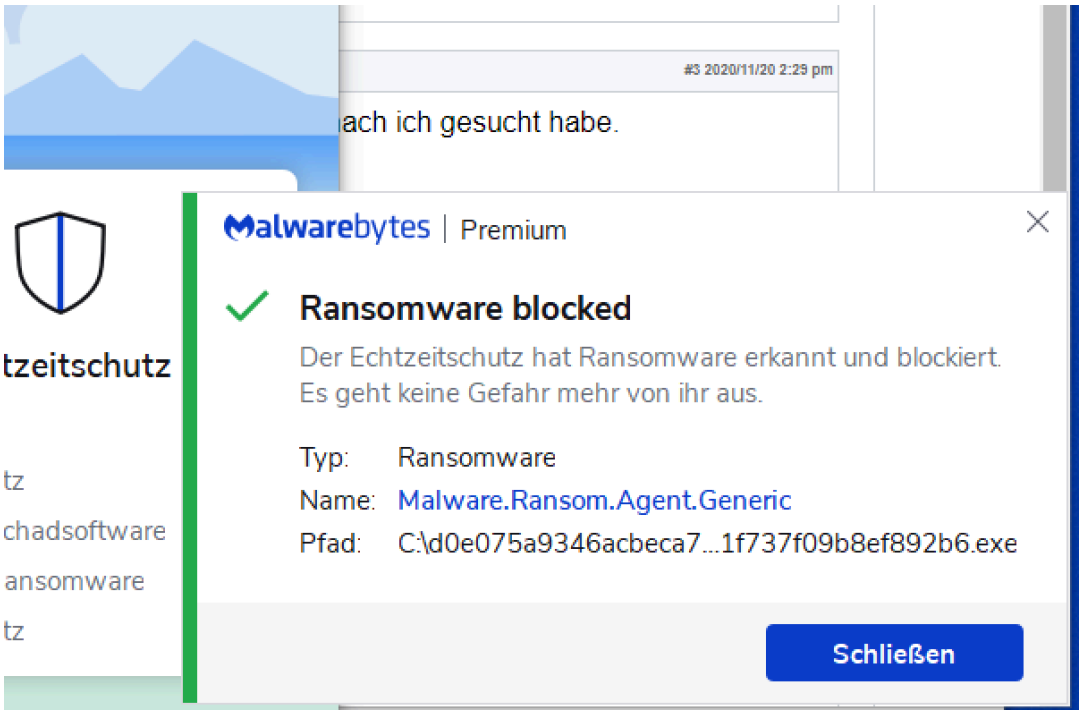
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

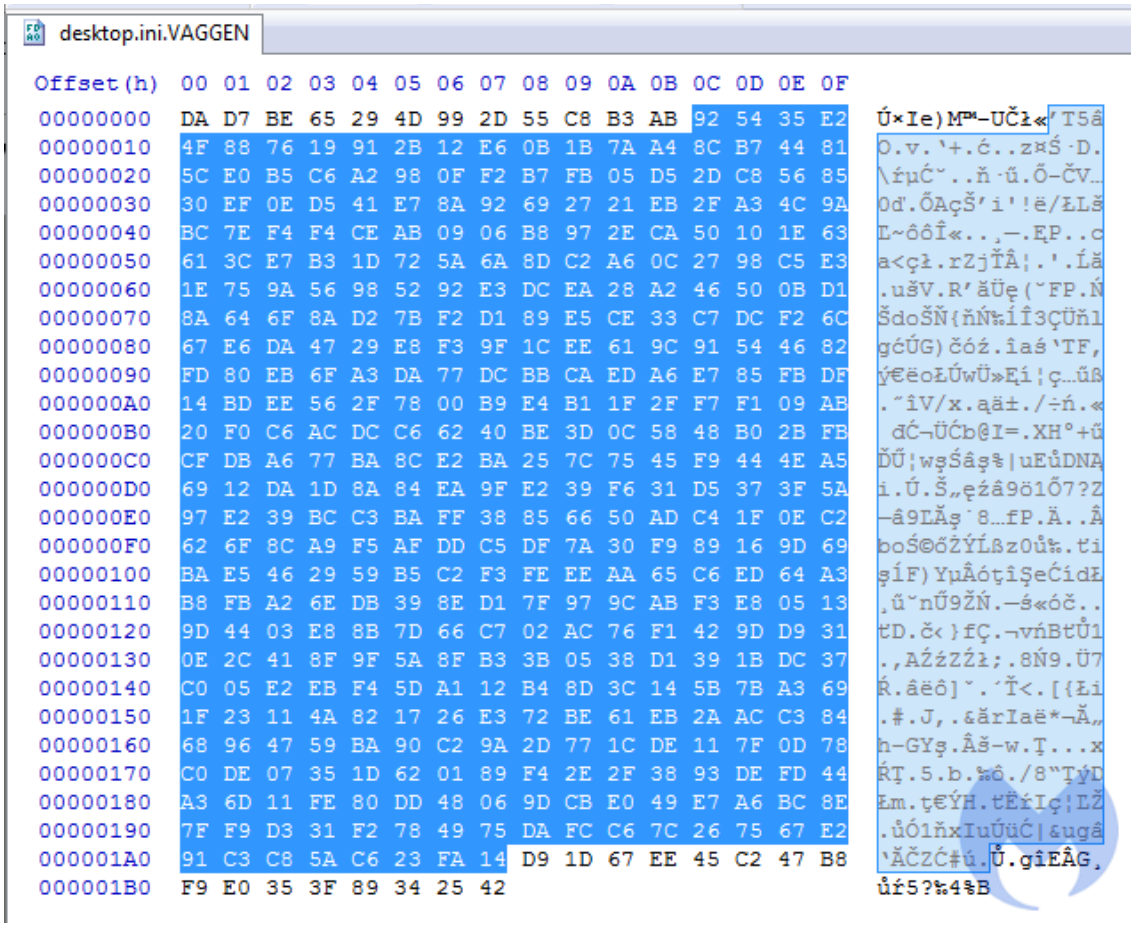
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

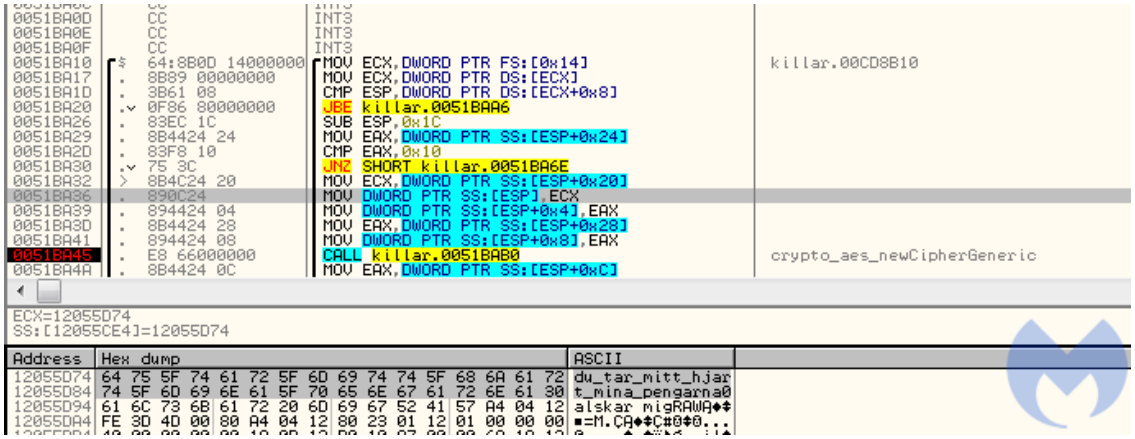


The content of the output file (with .VAGGEN extension) contains:

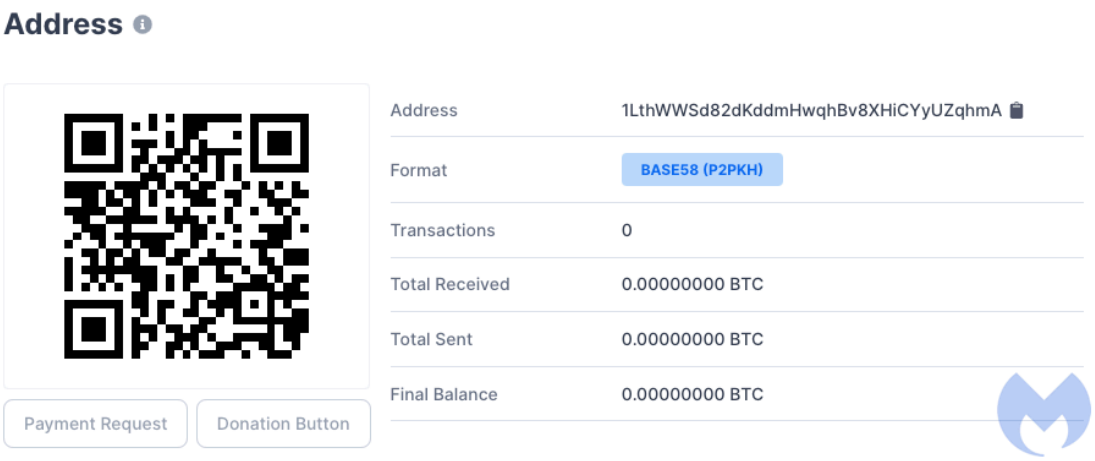
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

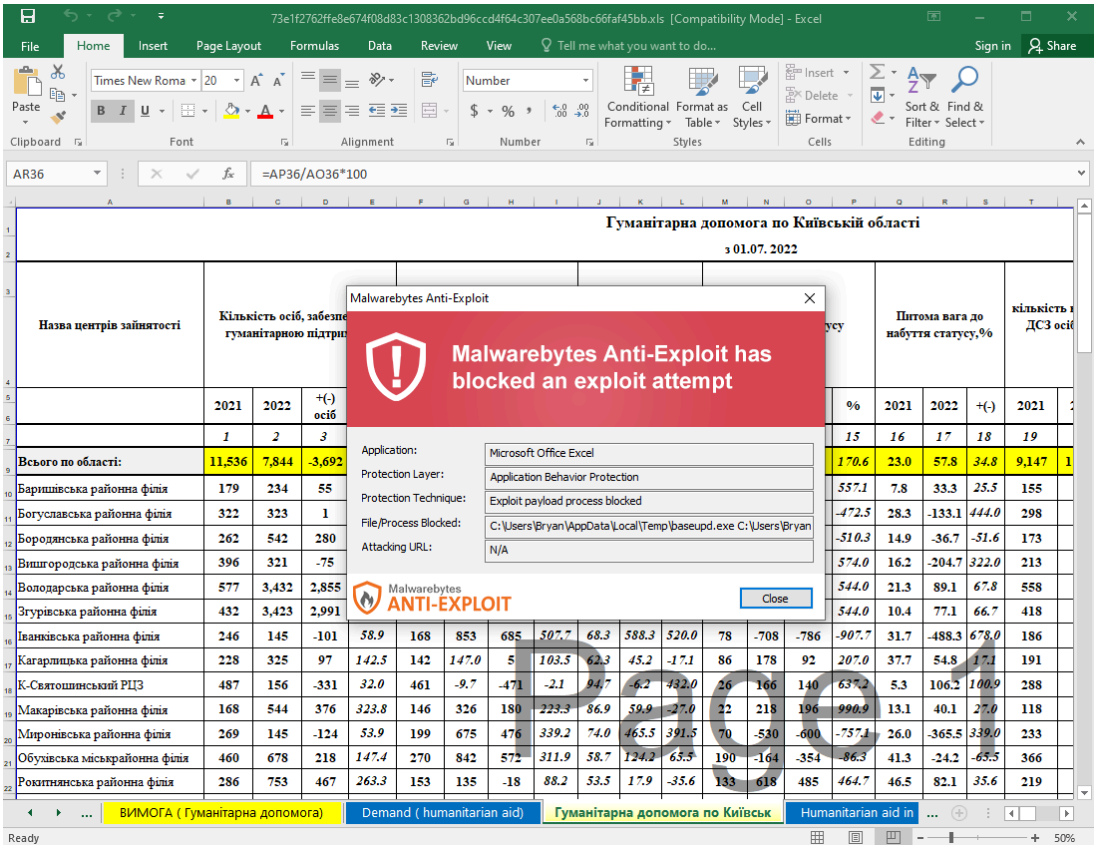
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

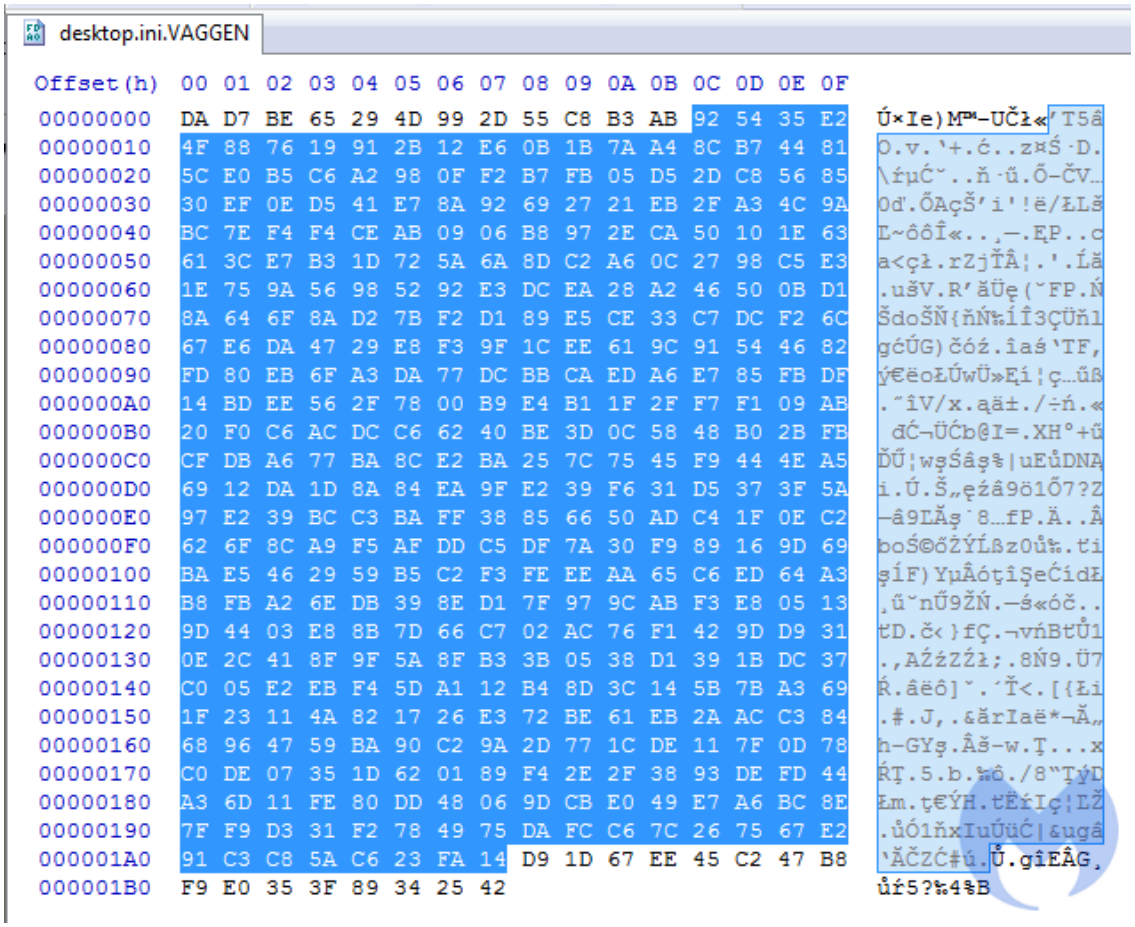
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

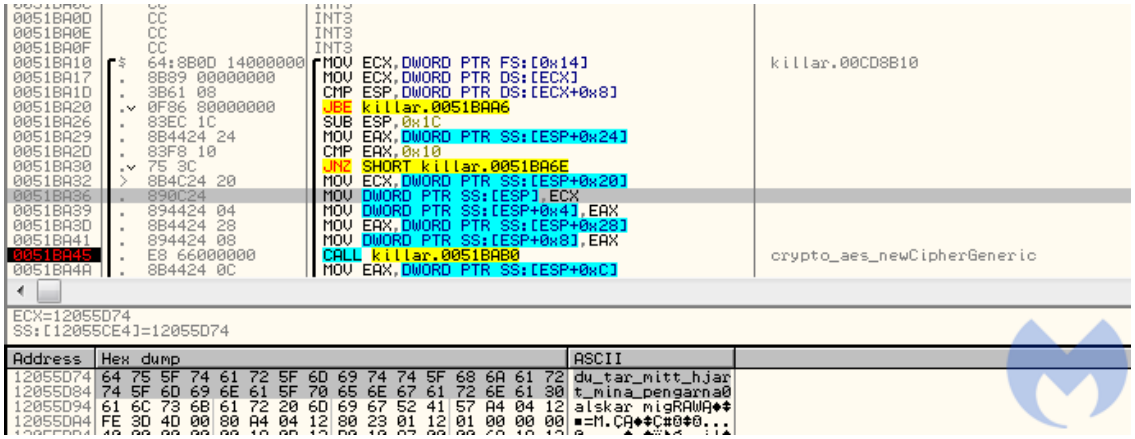


The content of the output file (with .VAGGEN extension) contains:

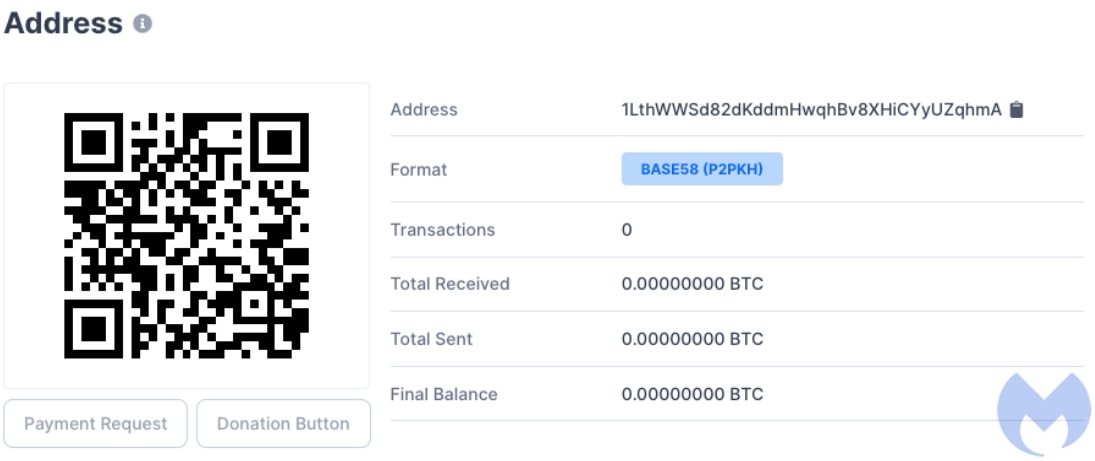
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

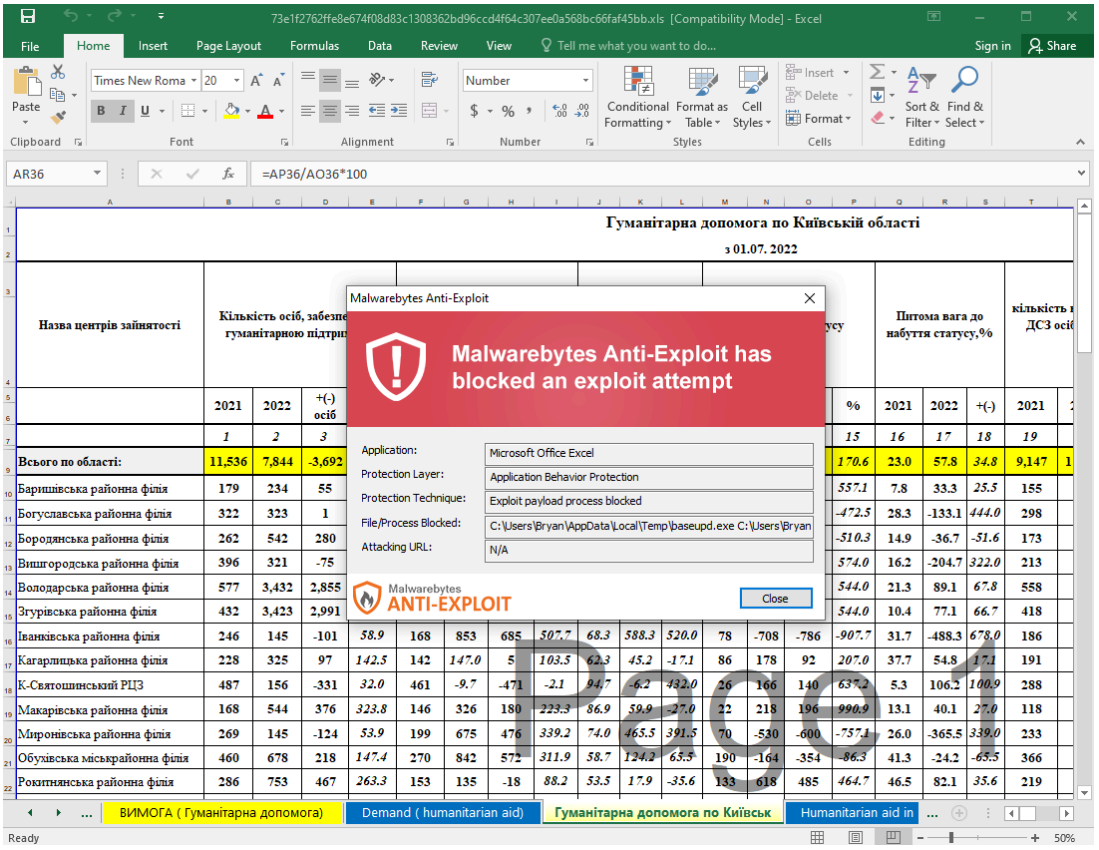
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

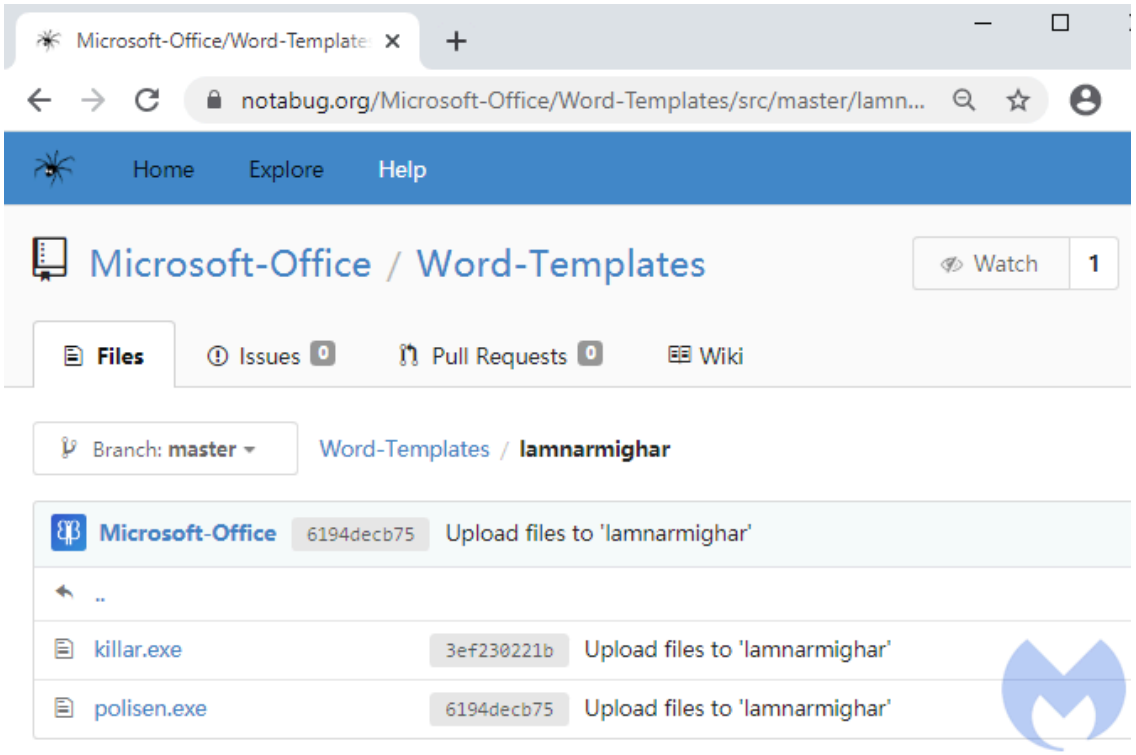
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



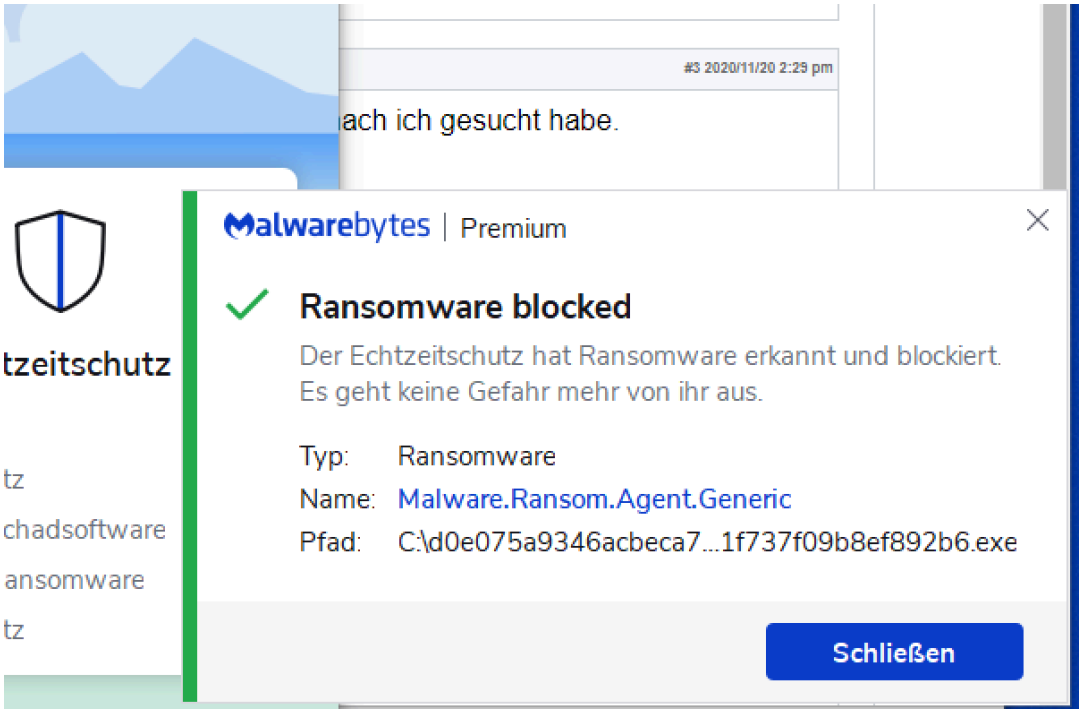
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

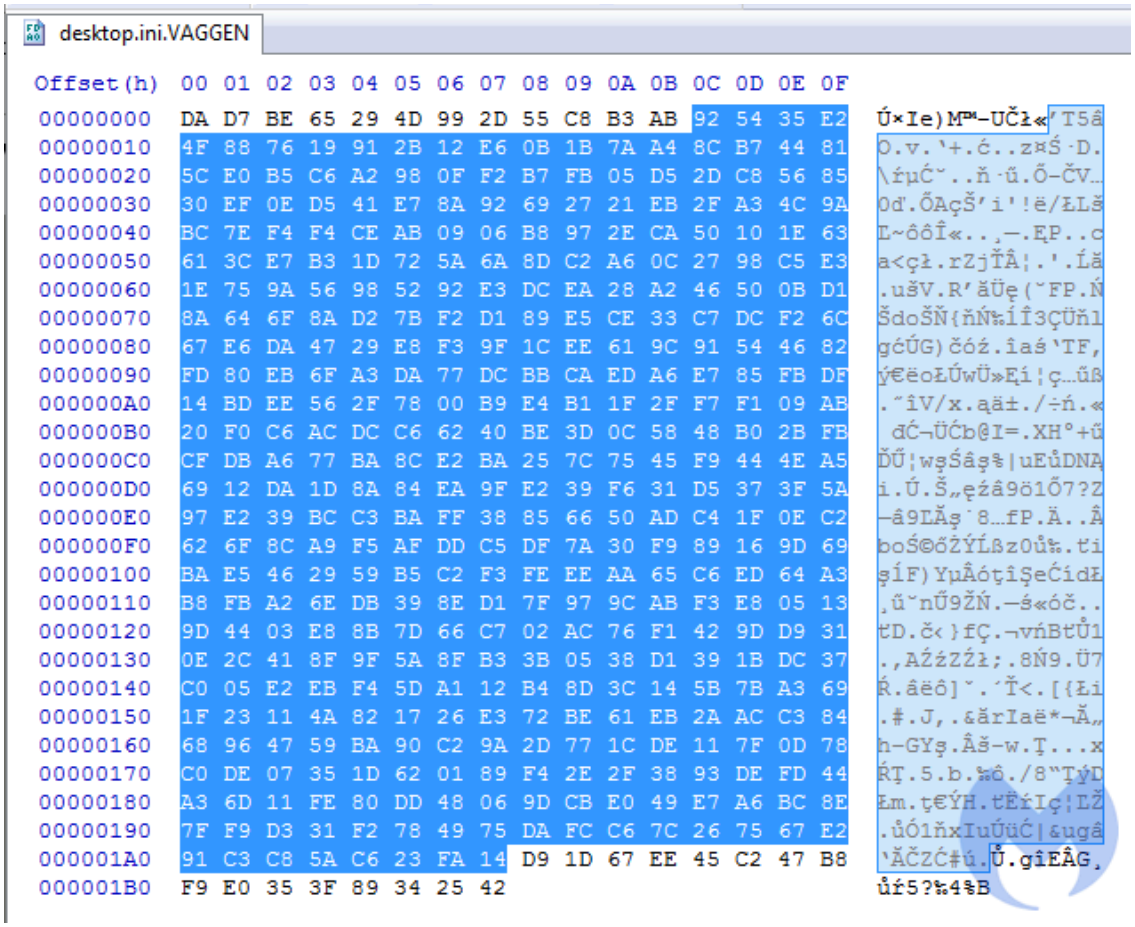
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

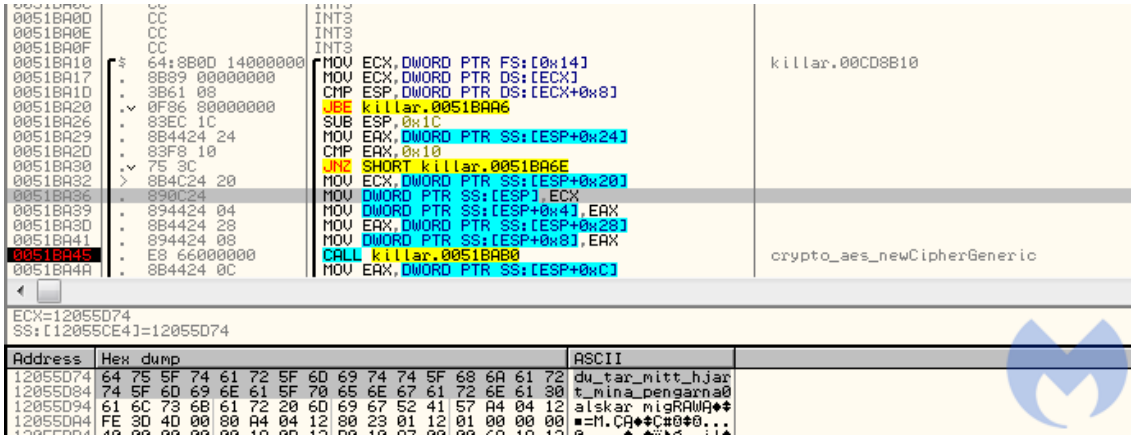


The content of the output file (with .VAGGEN extension) contains:

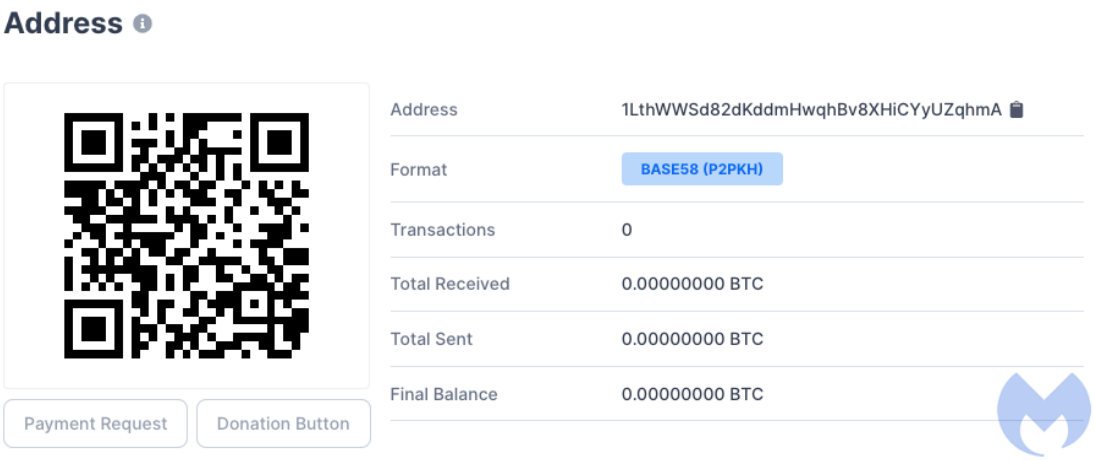
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

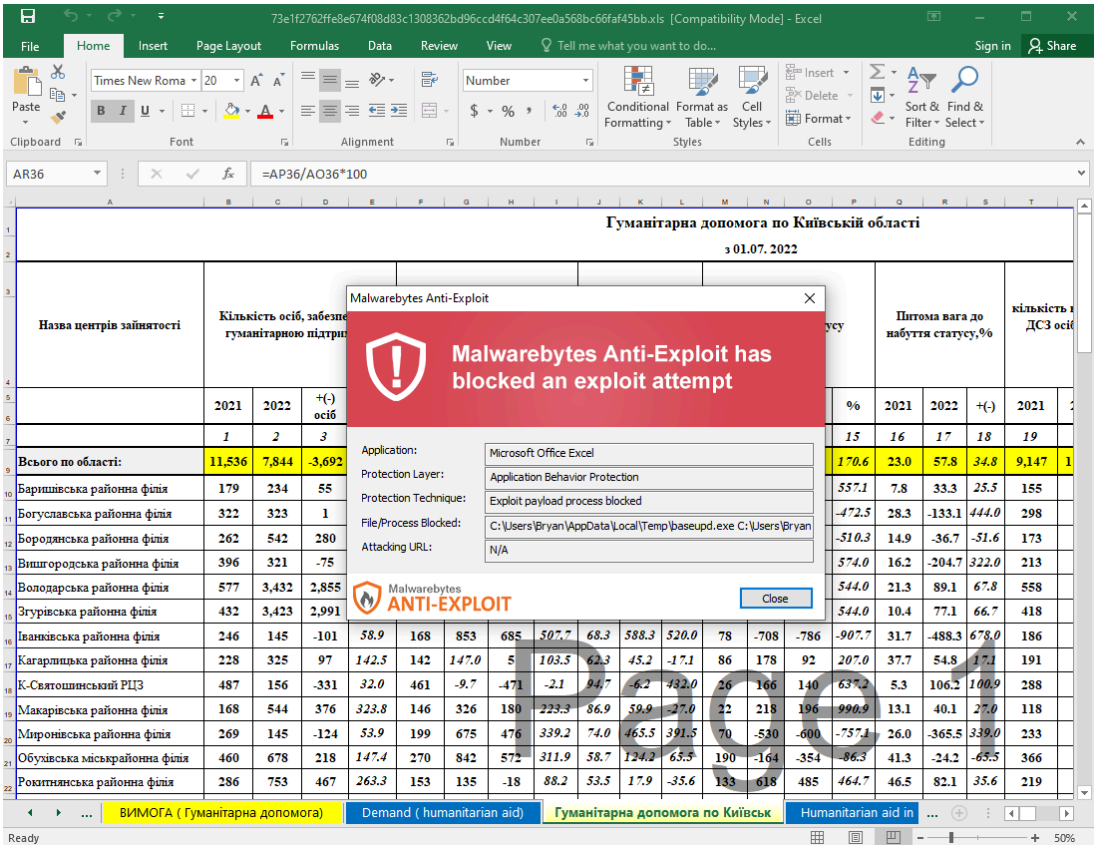
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

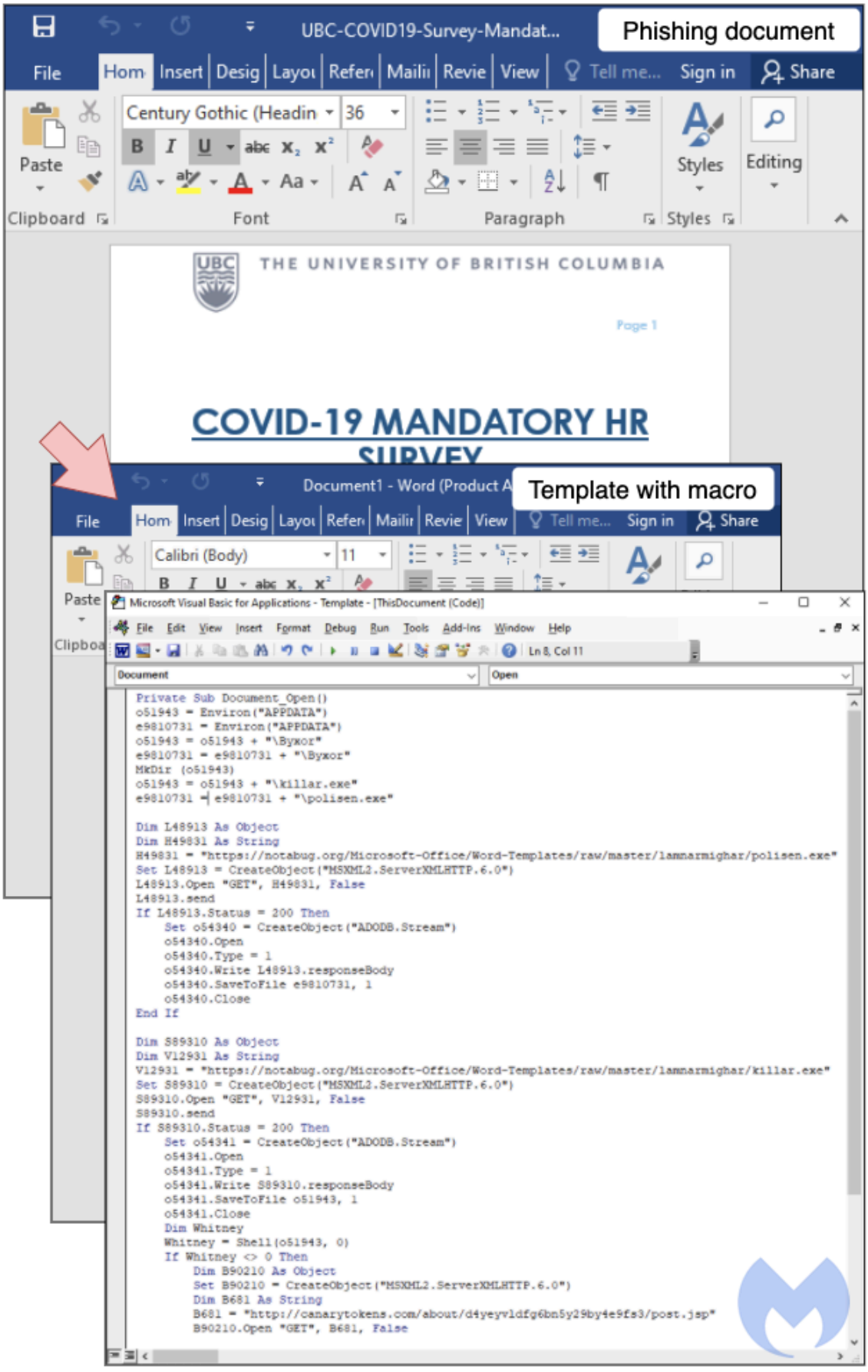
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

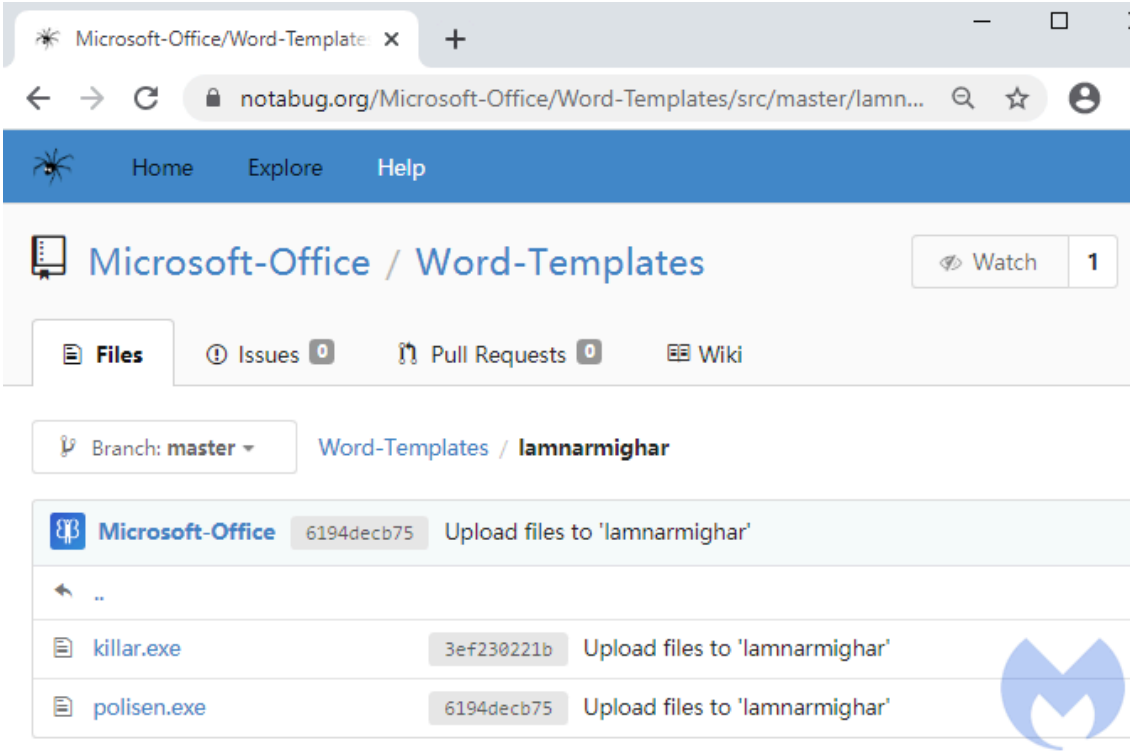
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



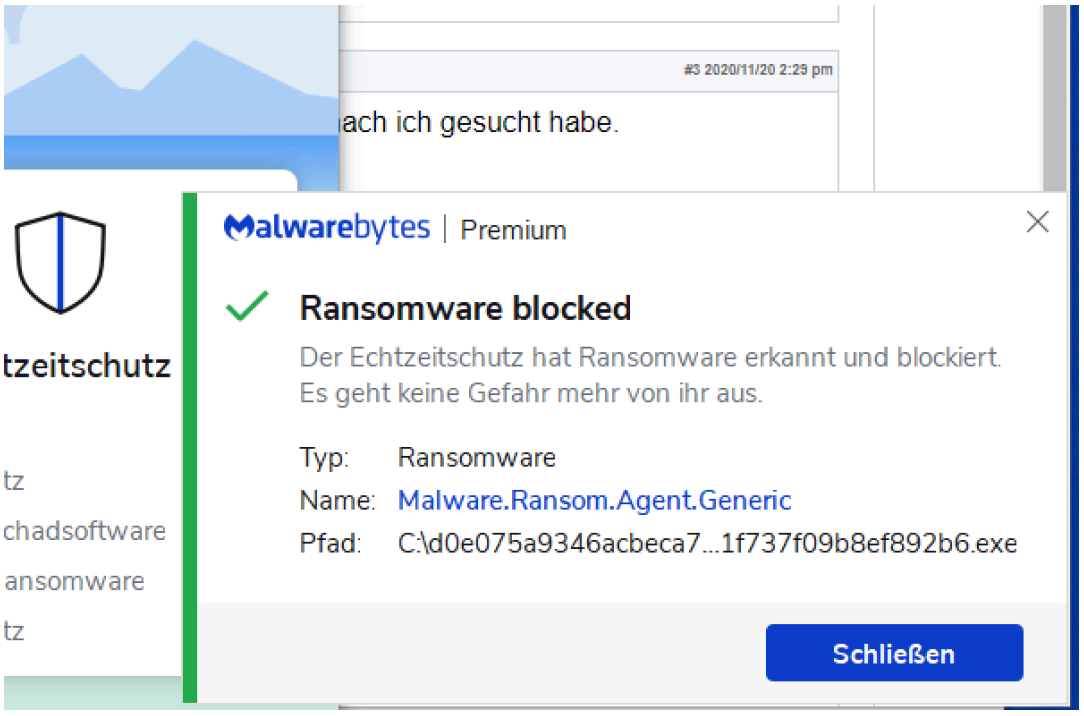
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

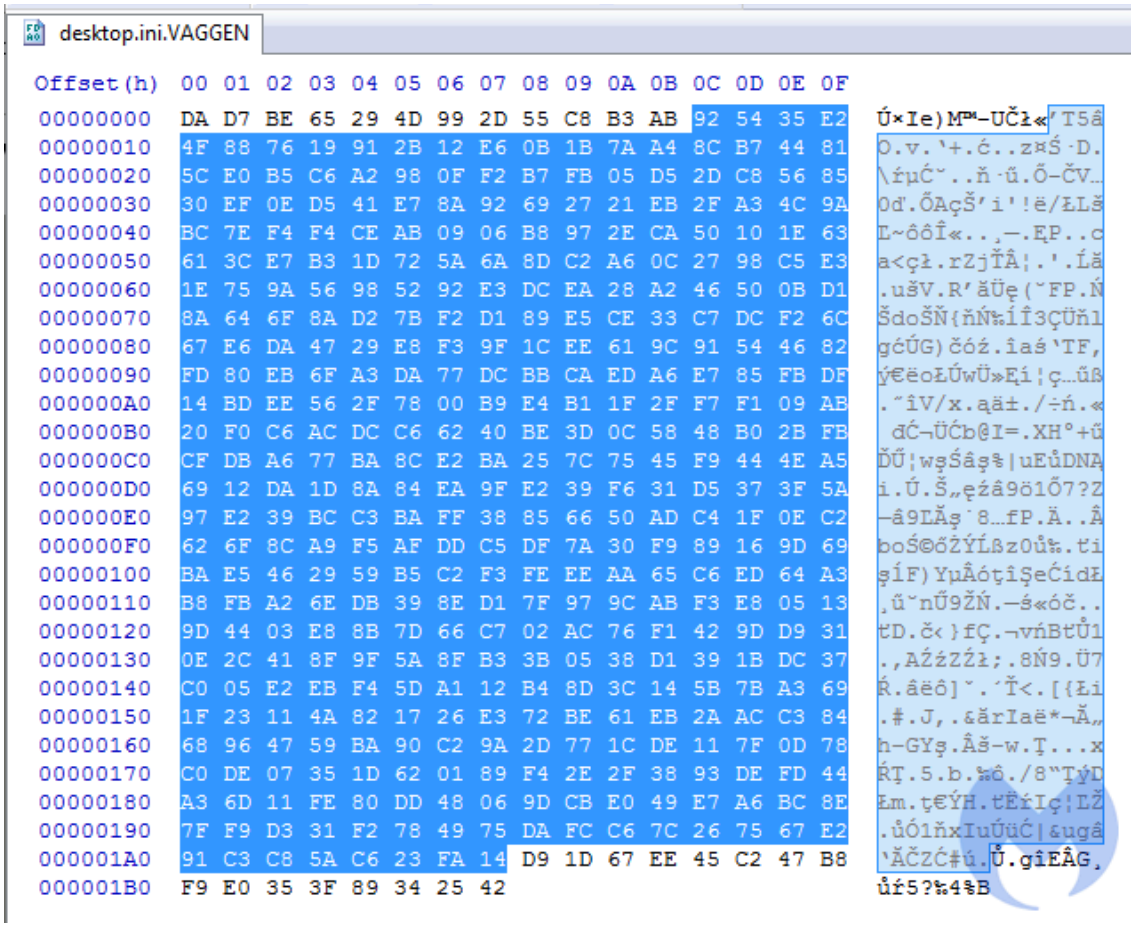
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

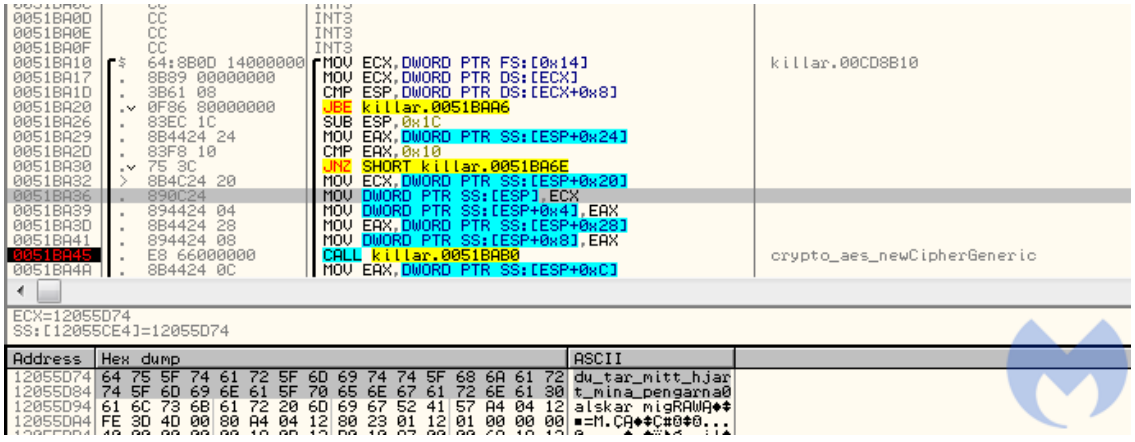


The content of the output file (with .VAGGEN extension) contains:

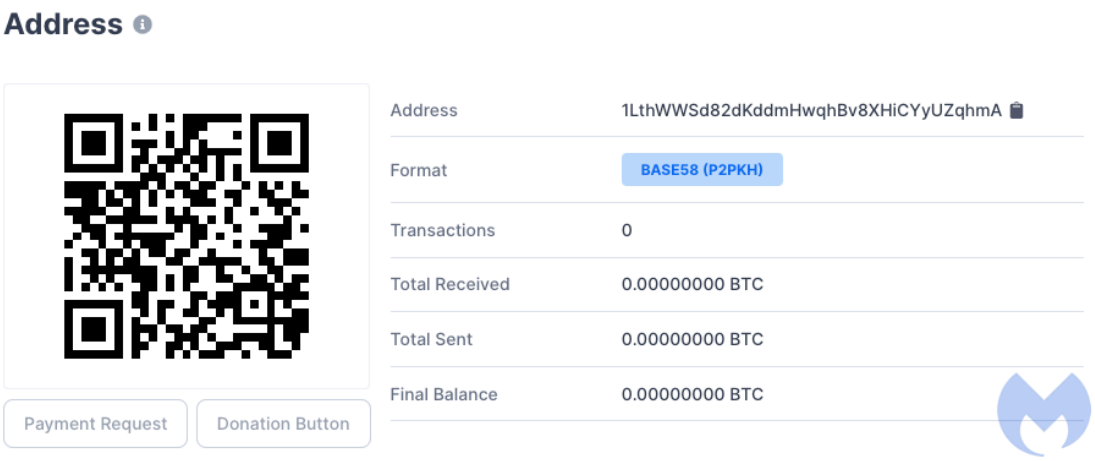
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

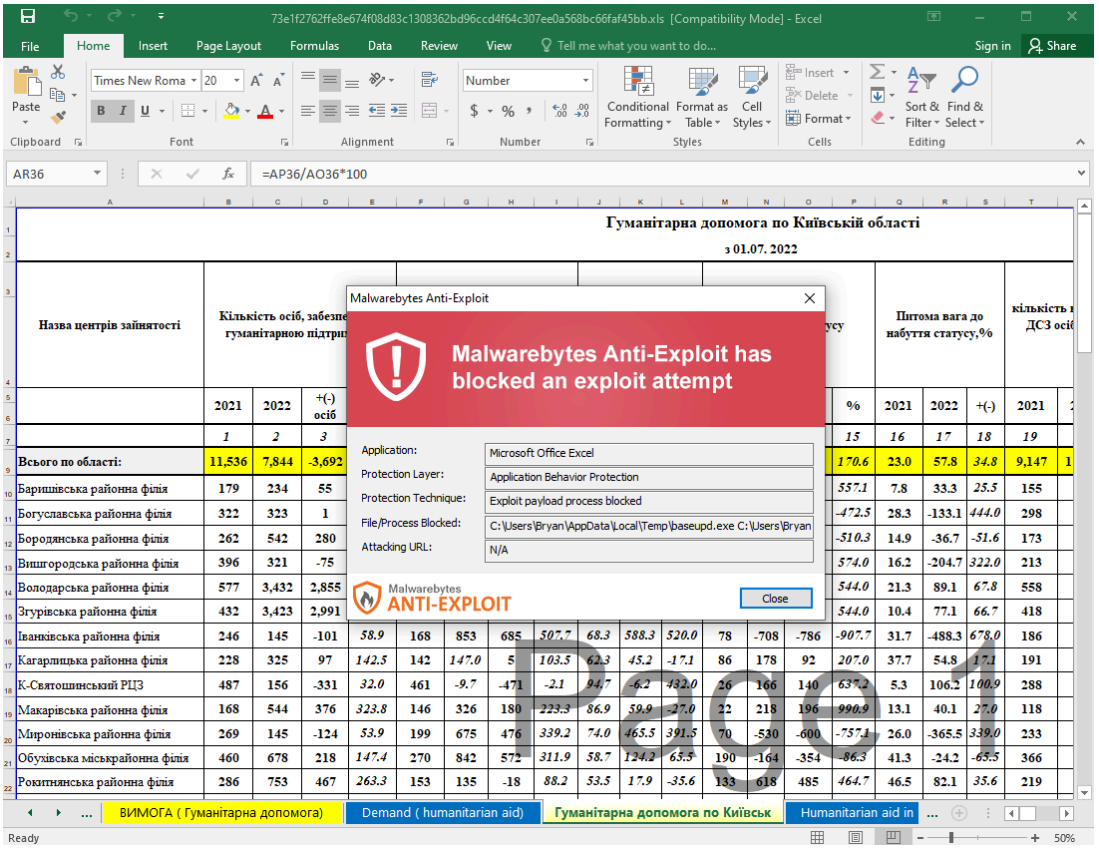
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

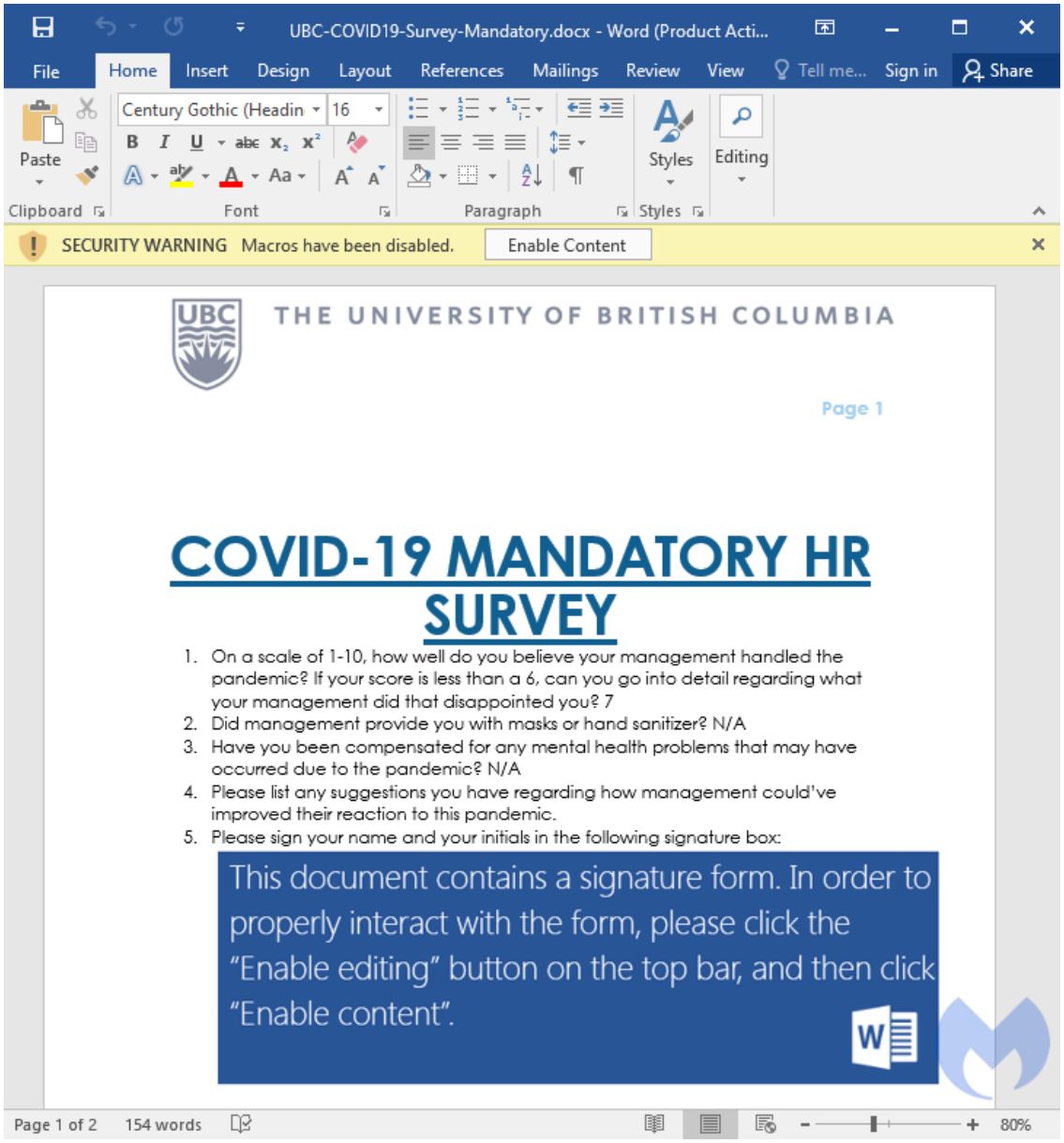
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

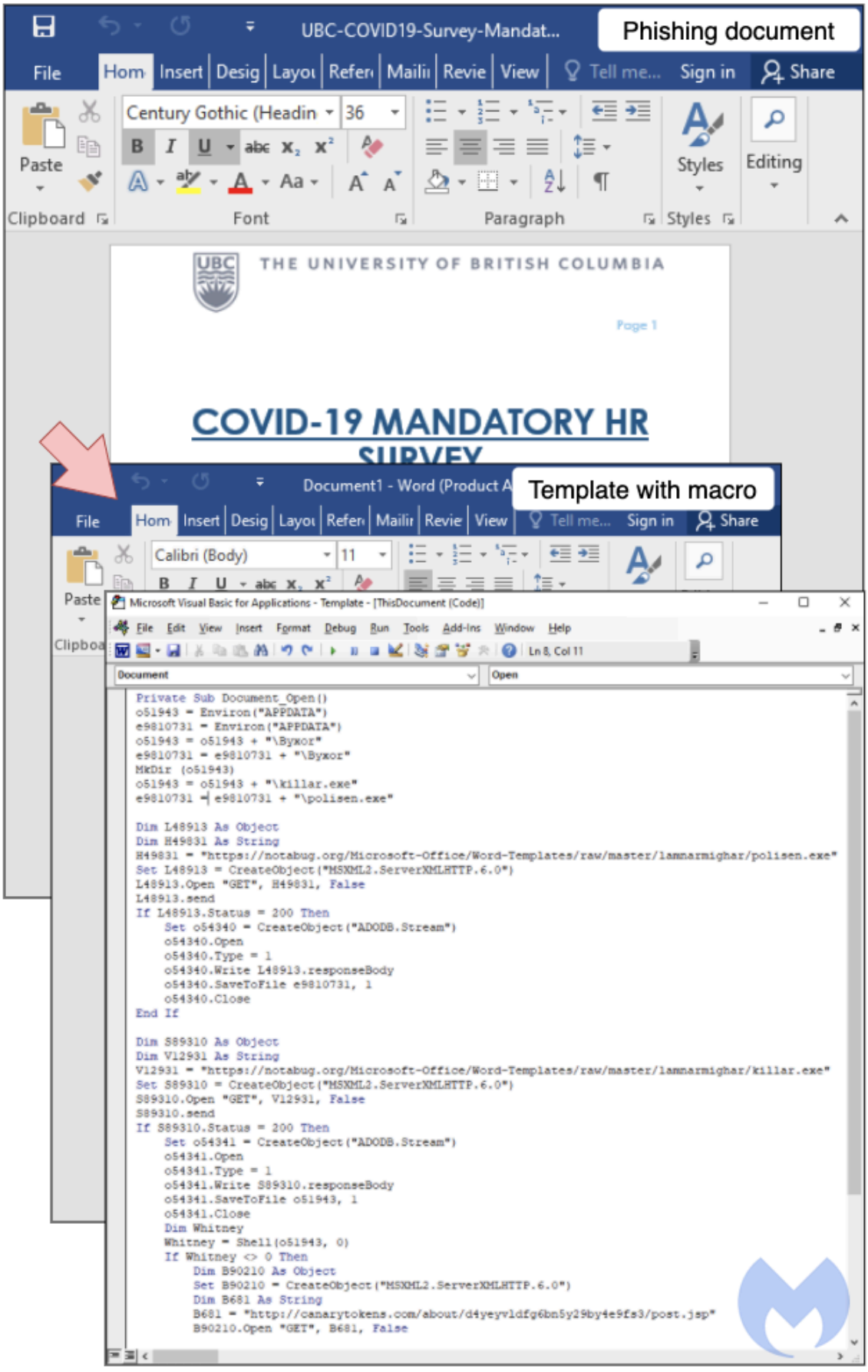
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

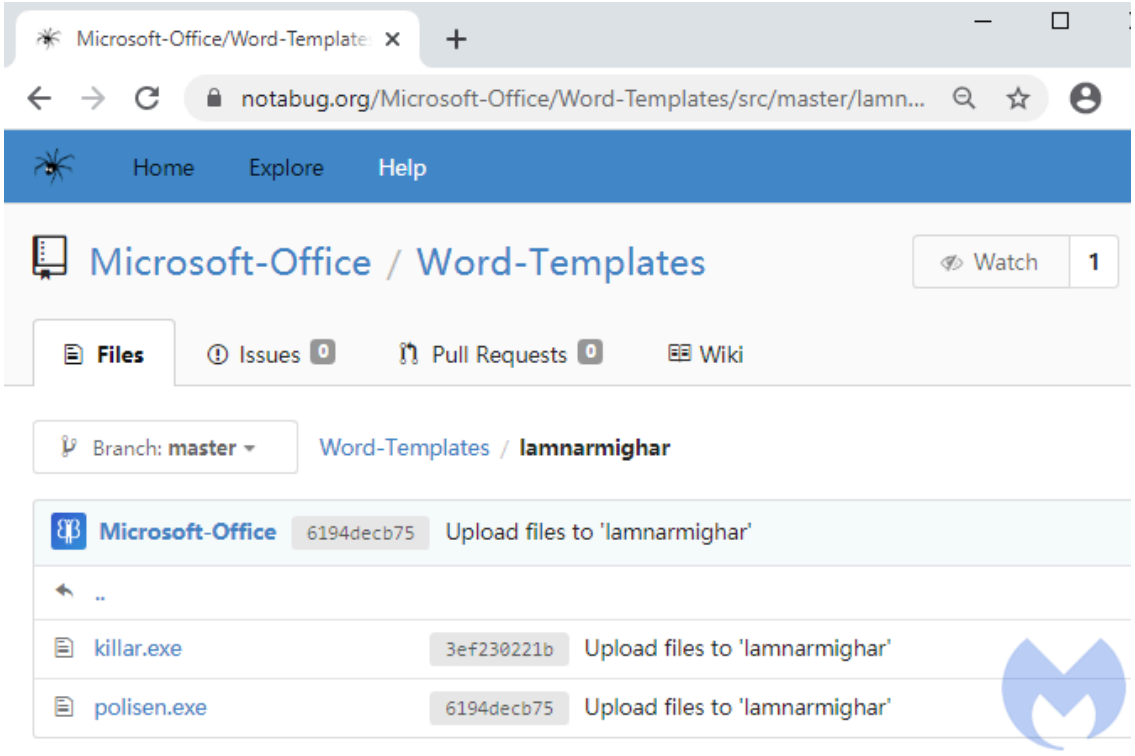
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



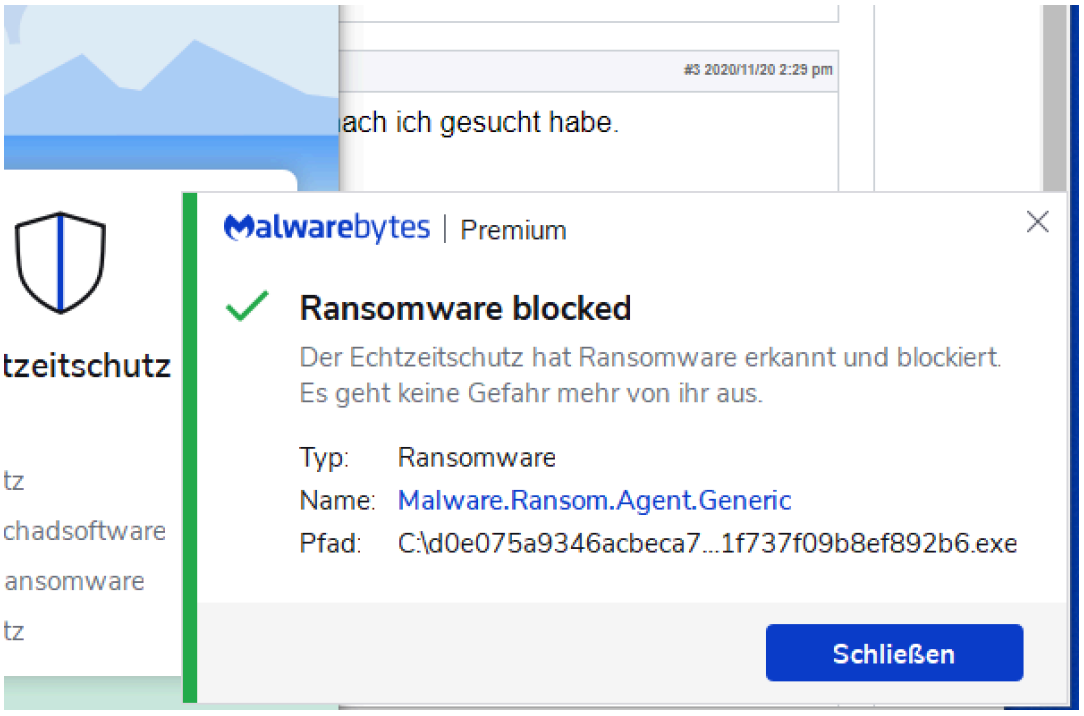
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

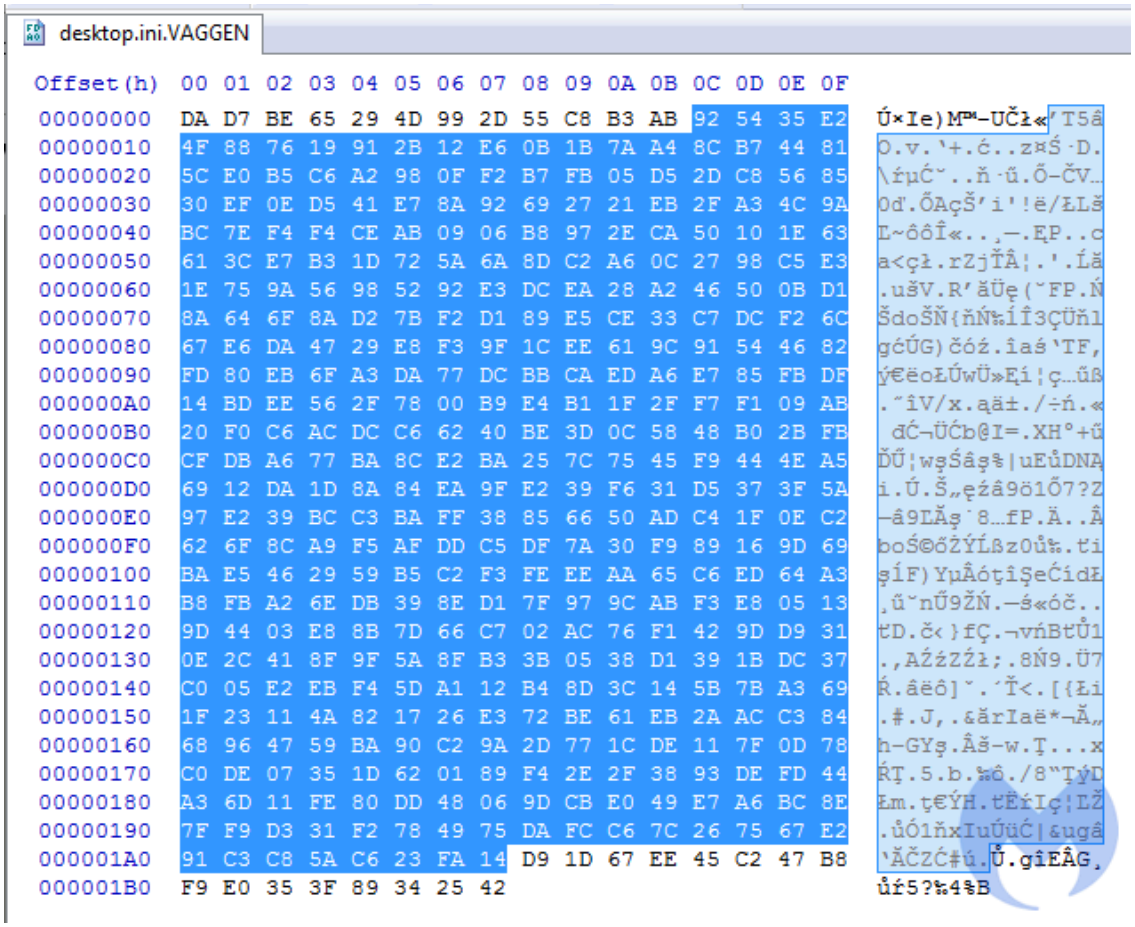
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



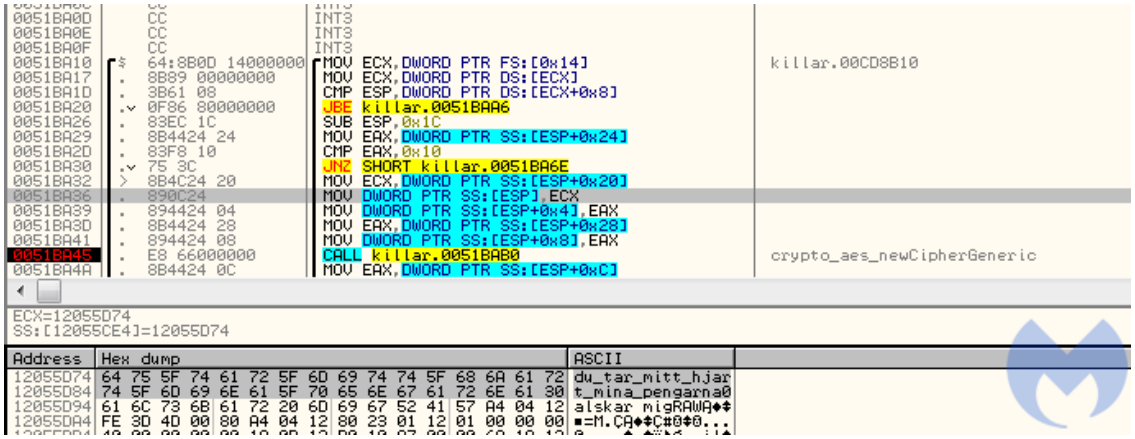
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

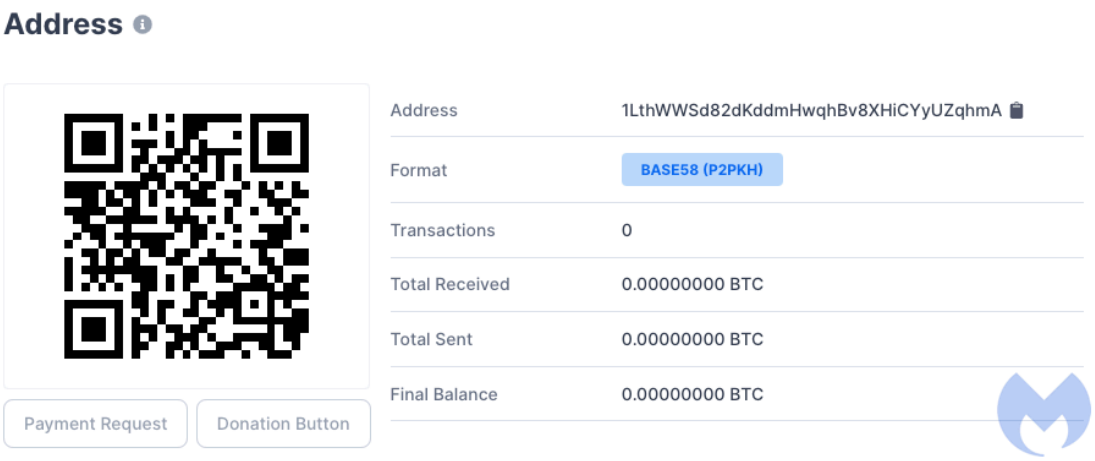
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

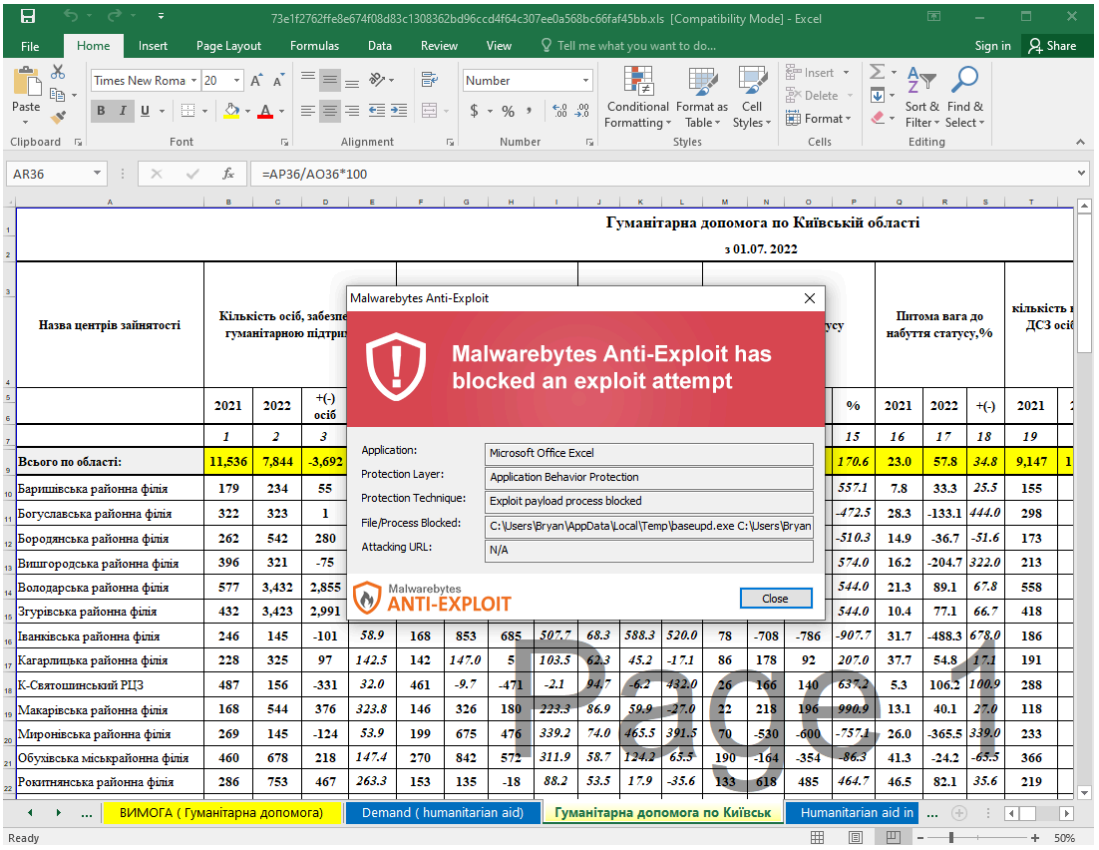
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

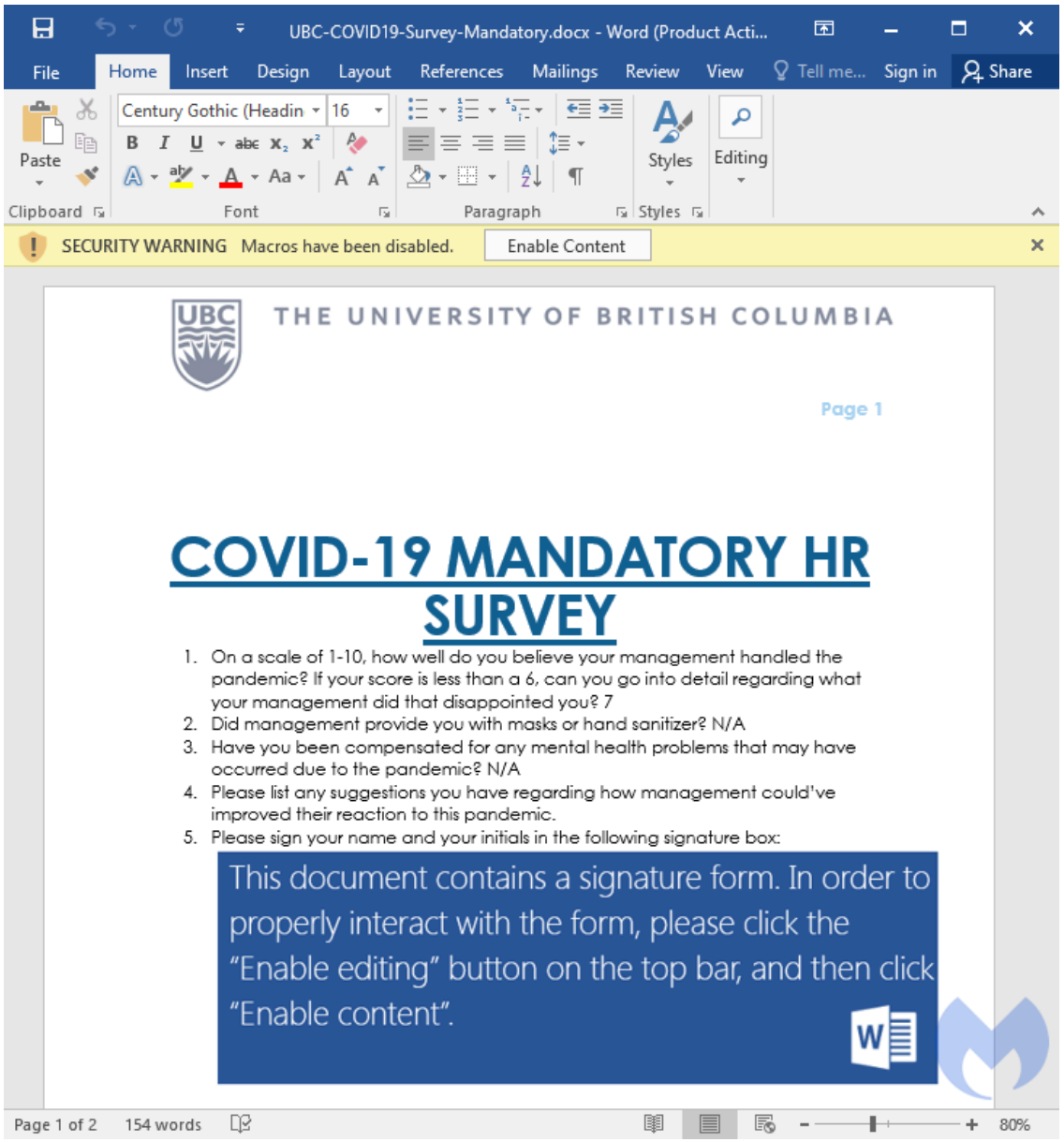
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

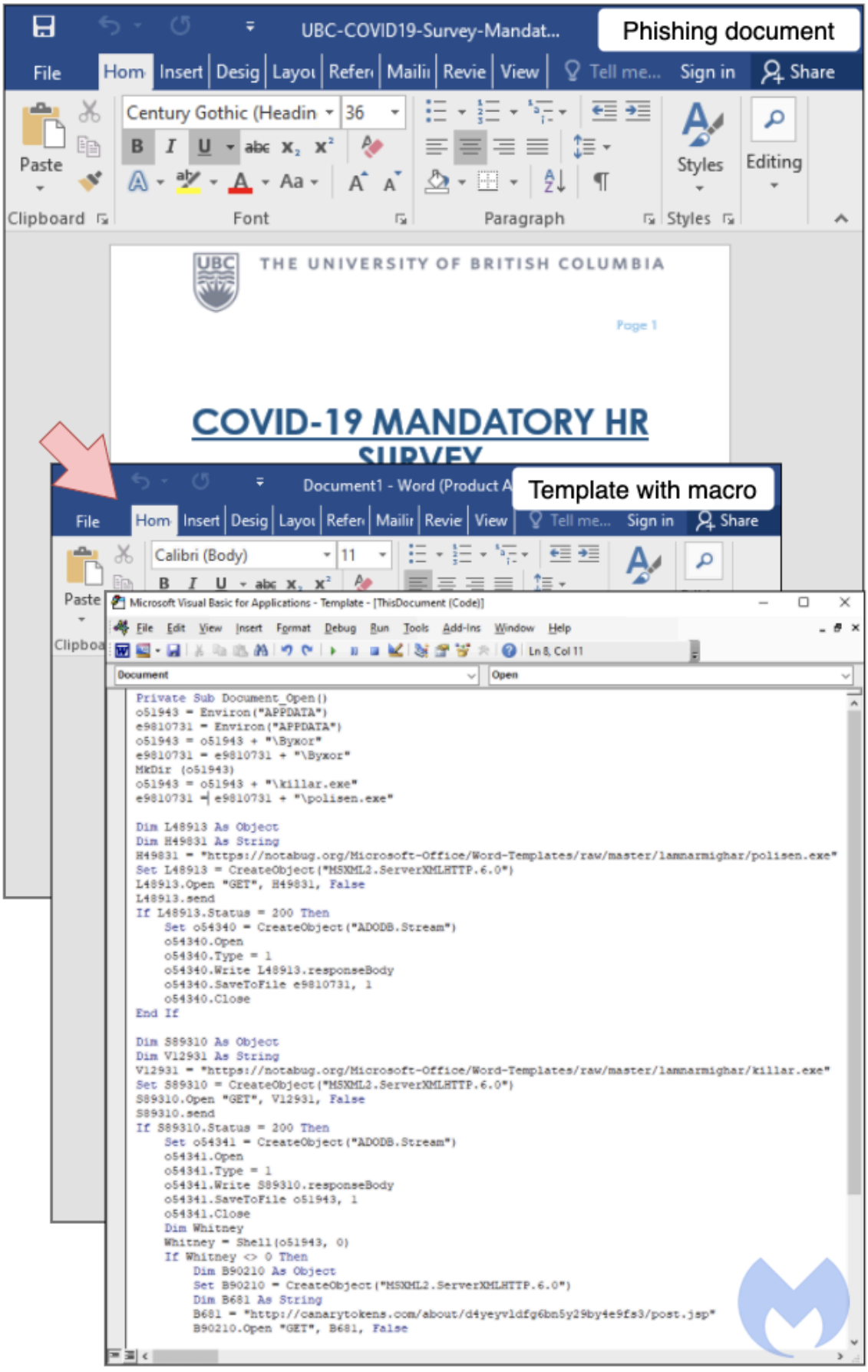
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

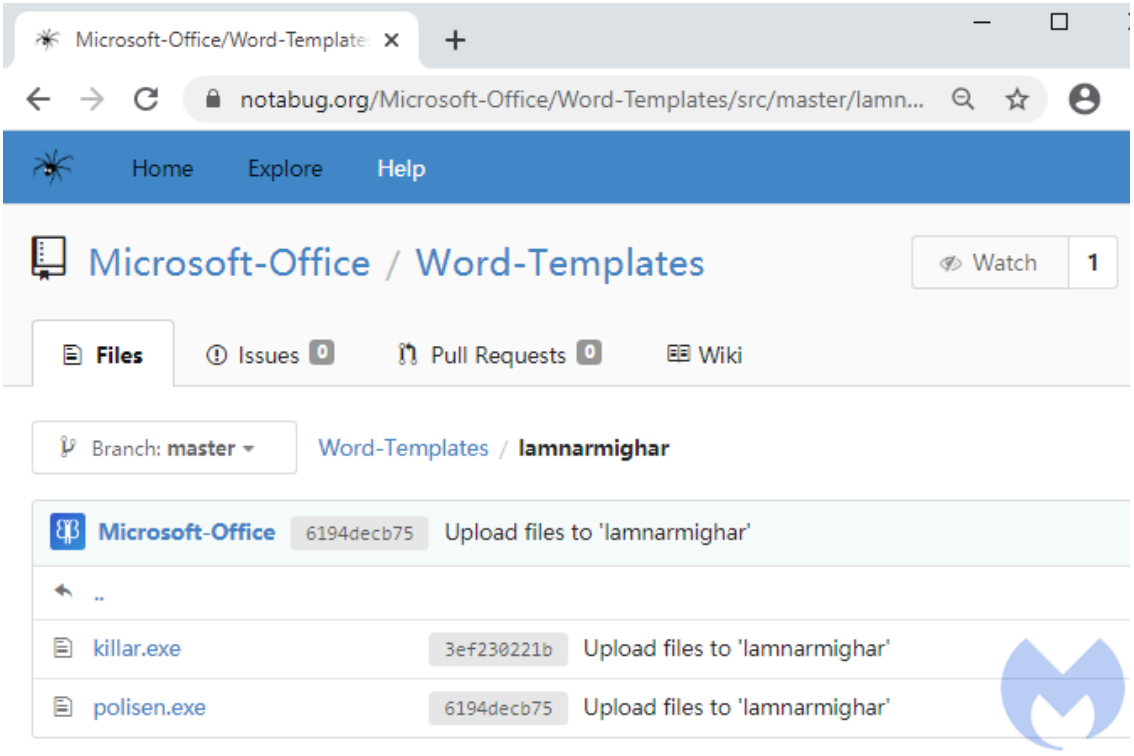
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



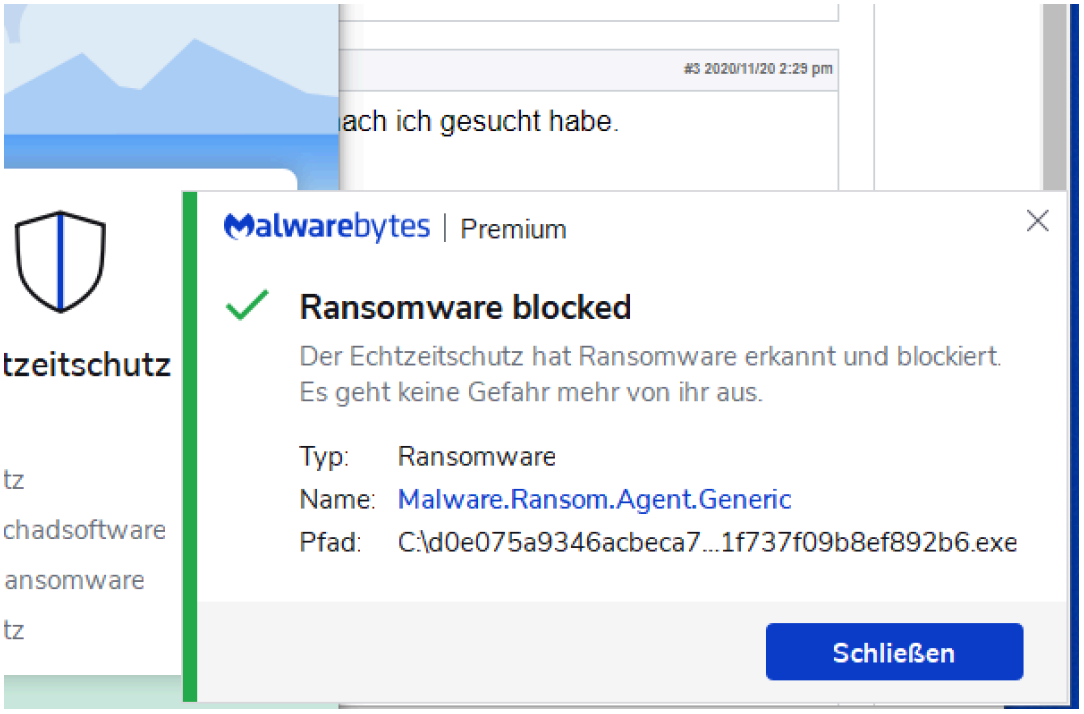
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

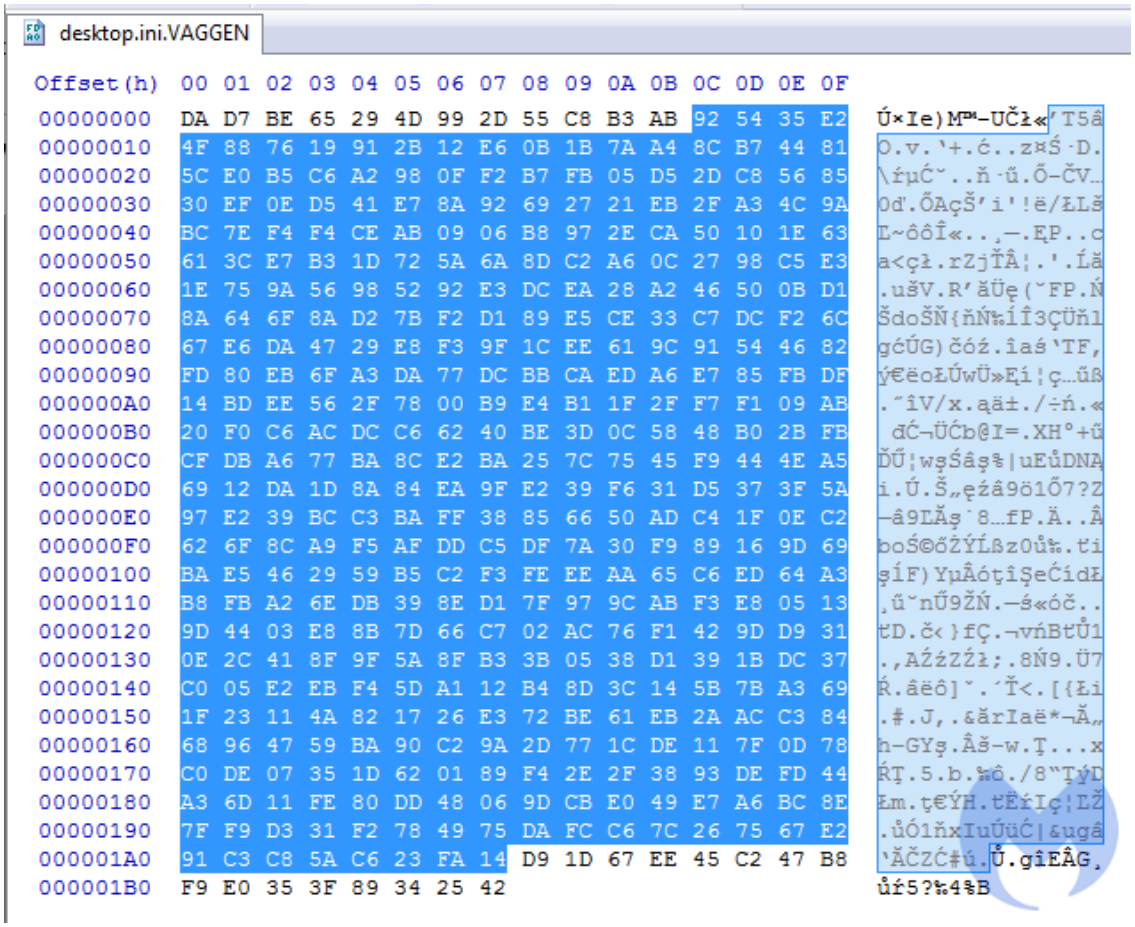
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

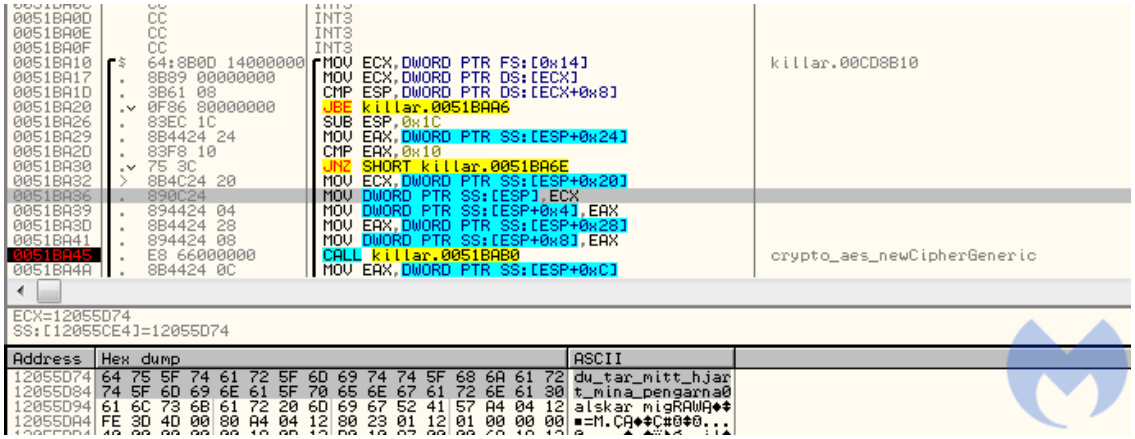


The content of the output file (with .VAGGEN extension) contains:

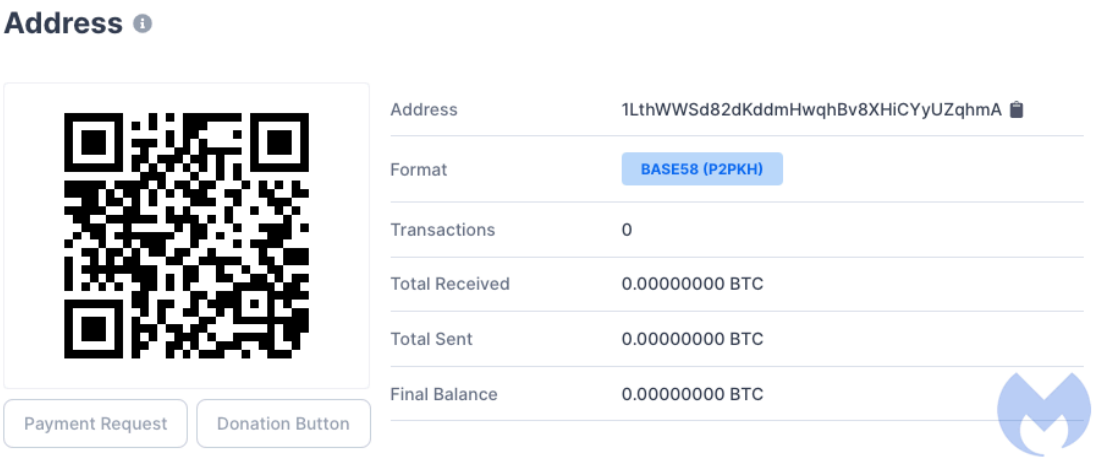
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

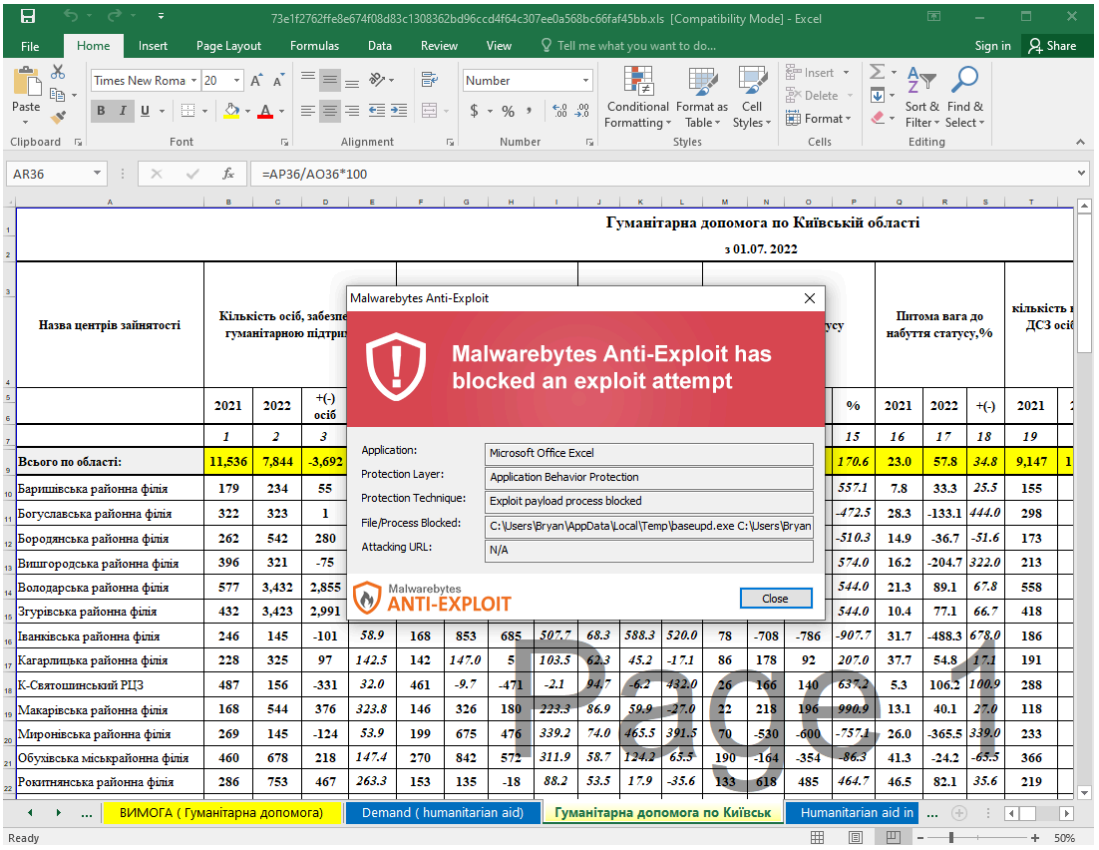
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

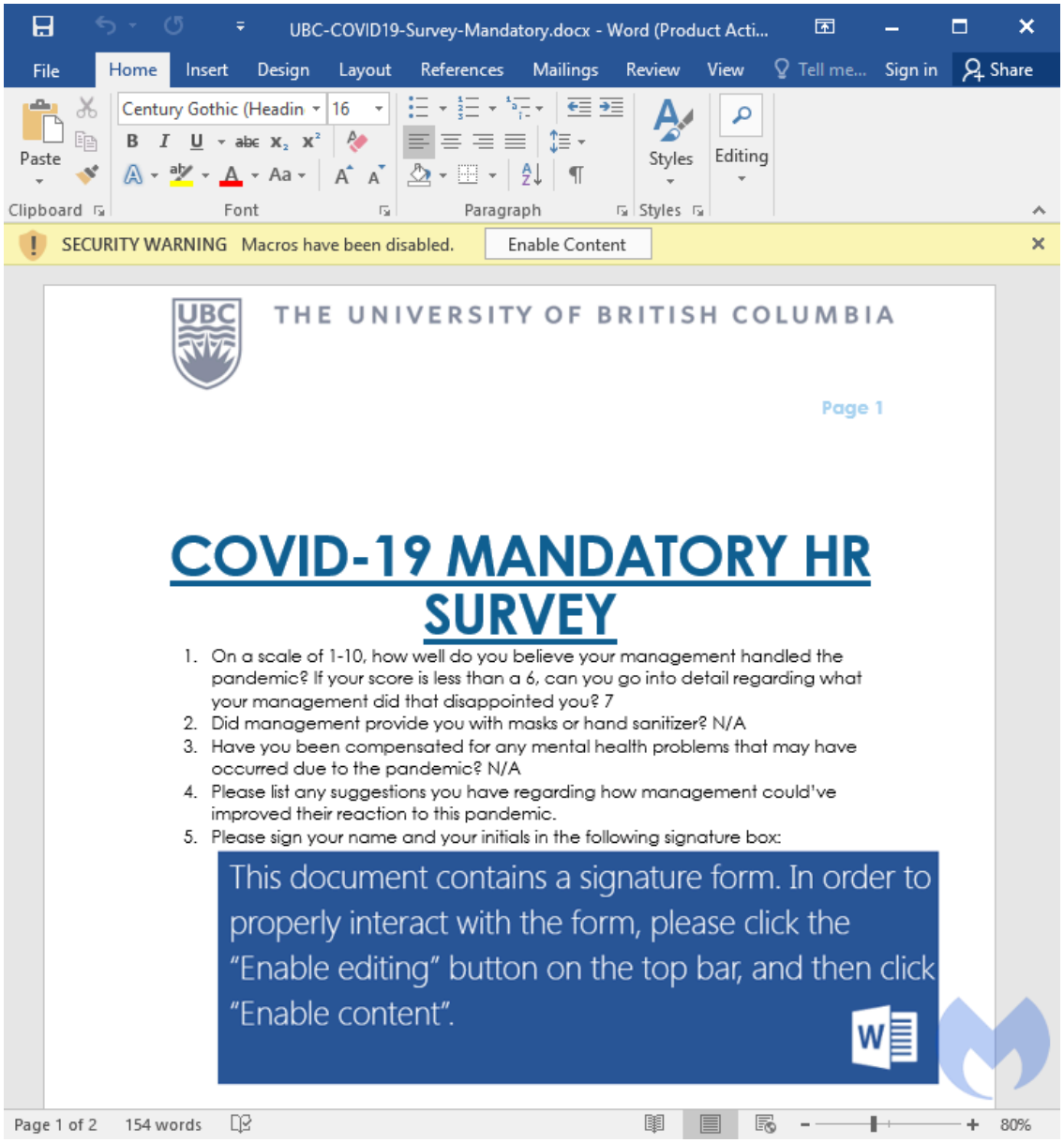
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

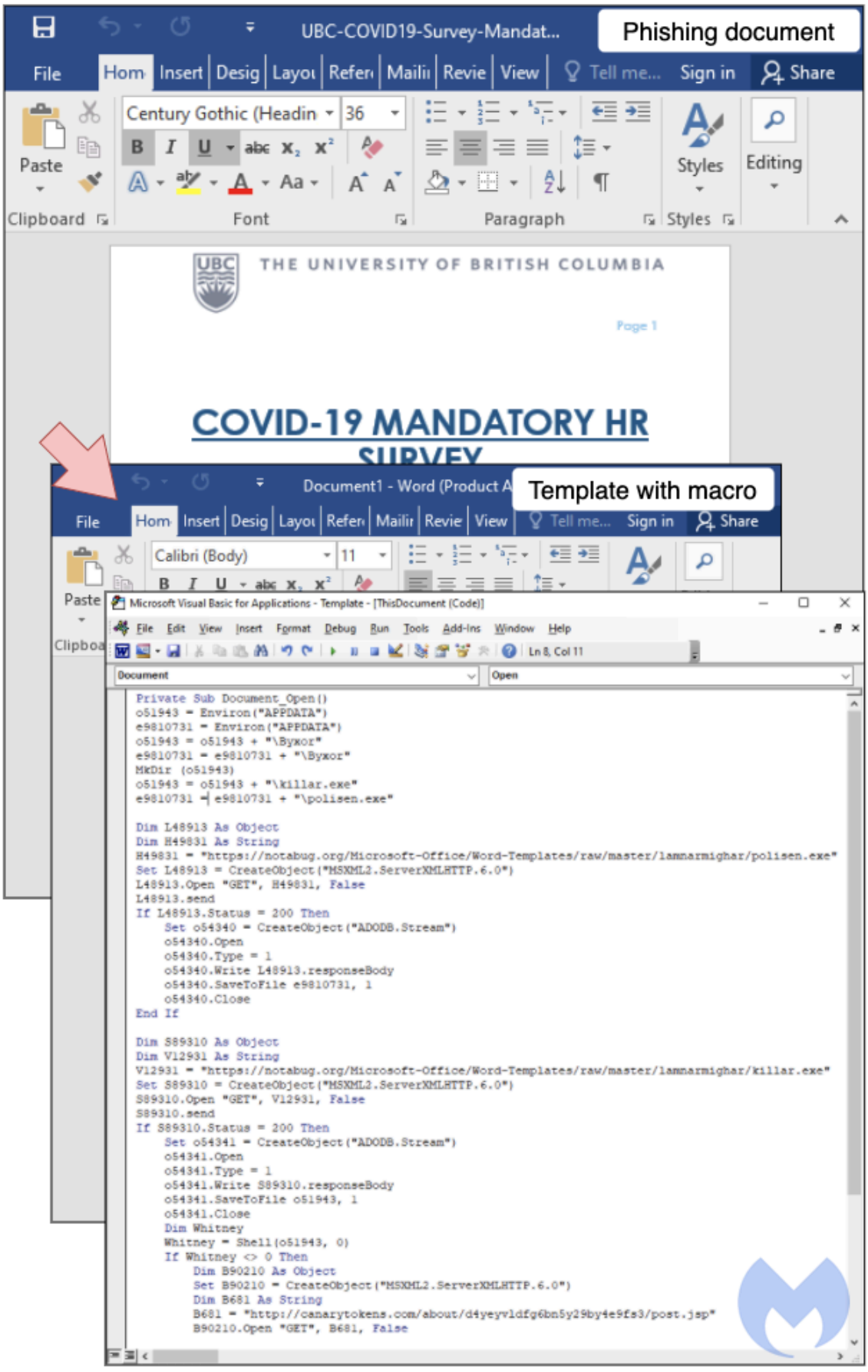
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

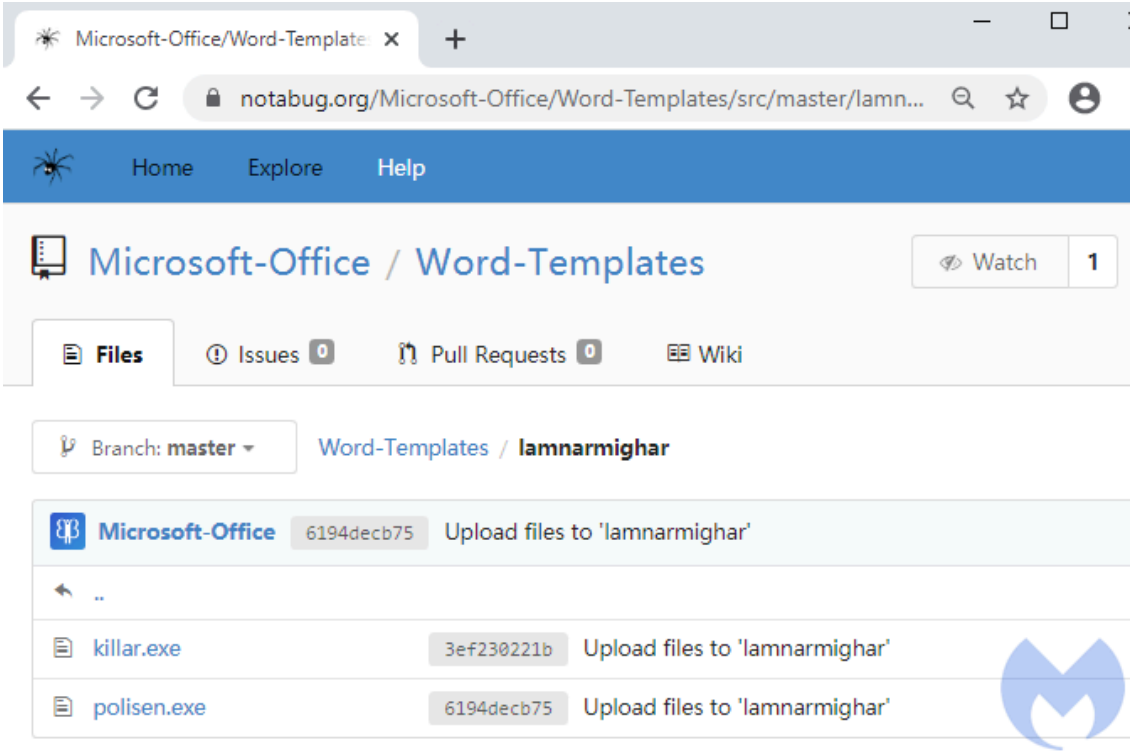
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



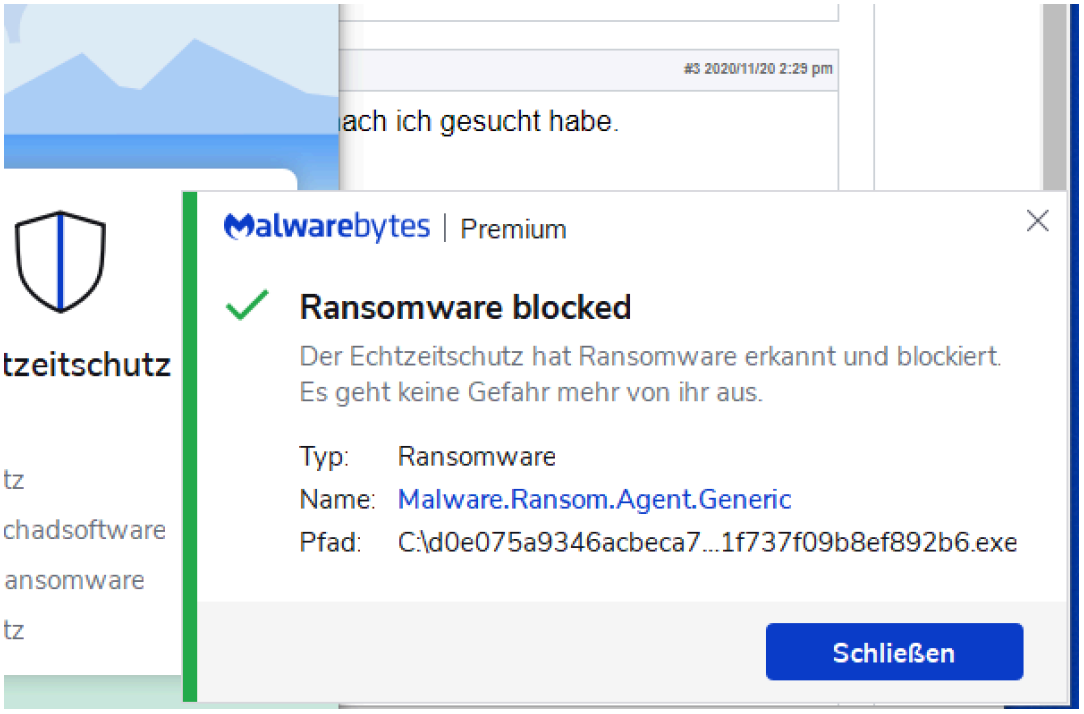
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

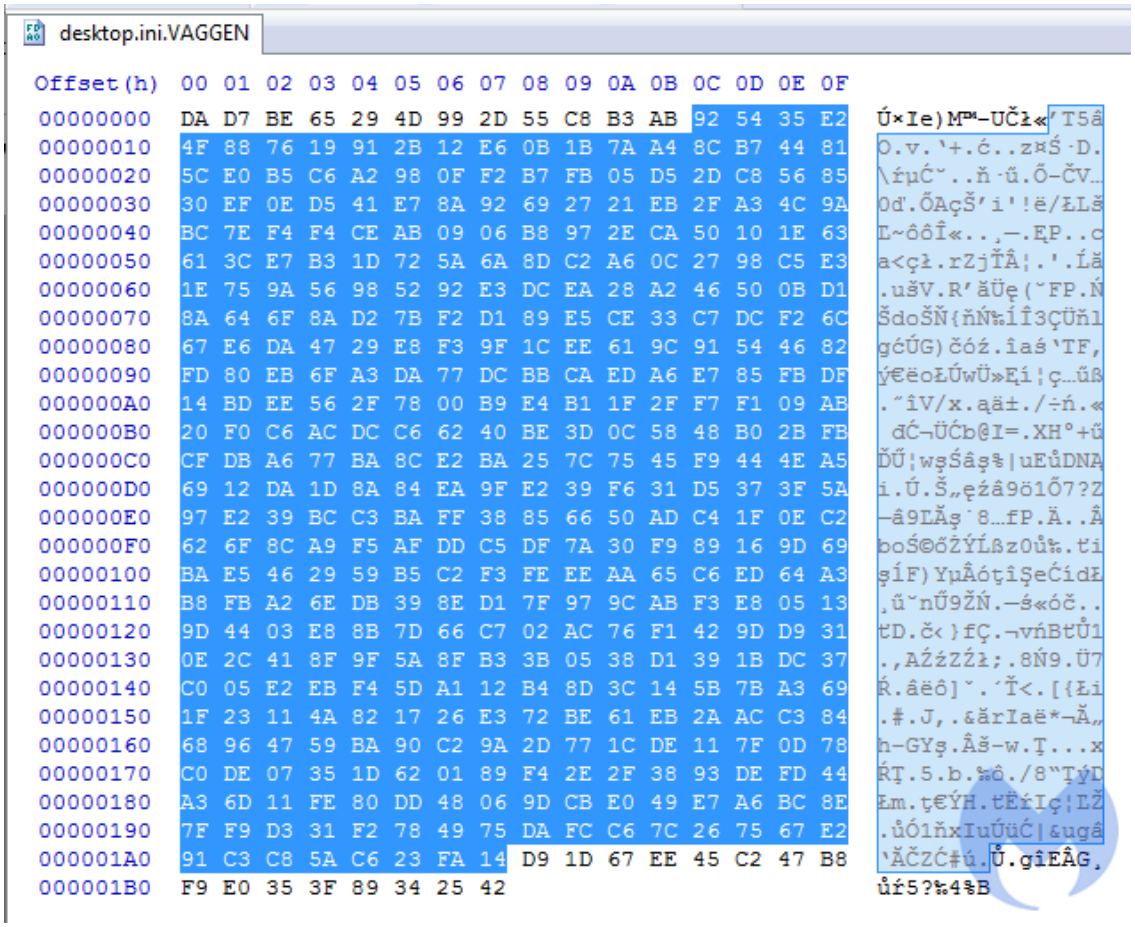
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

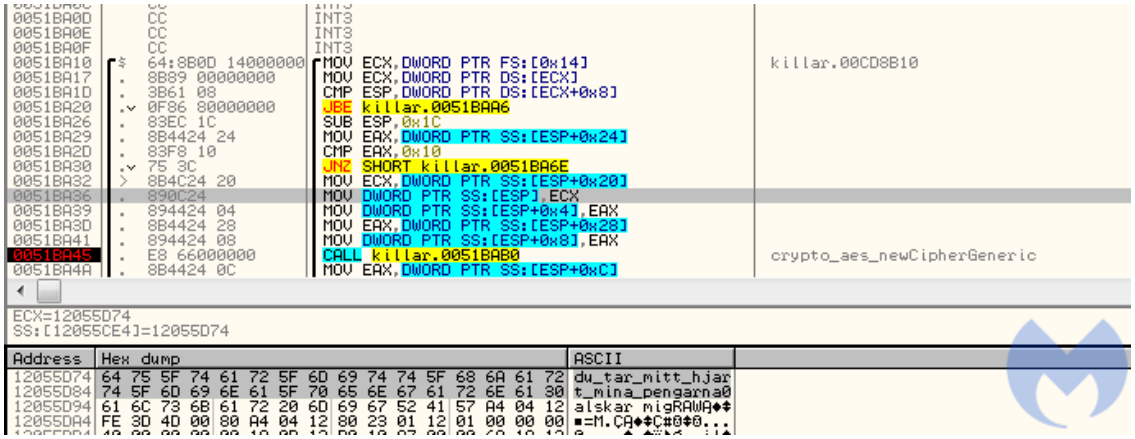


The content of the output file (with .VAGGEN extension) contains:

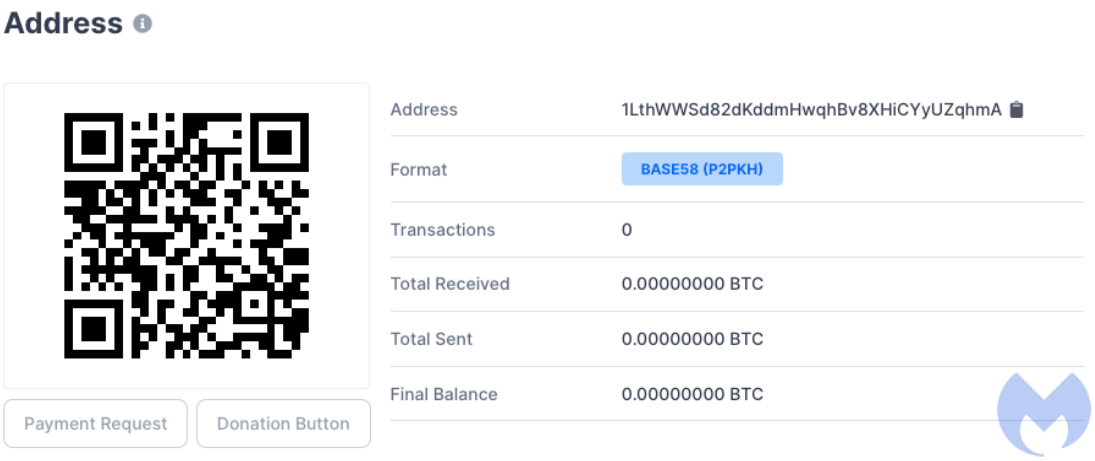
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

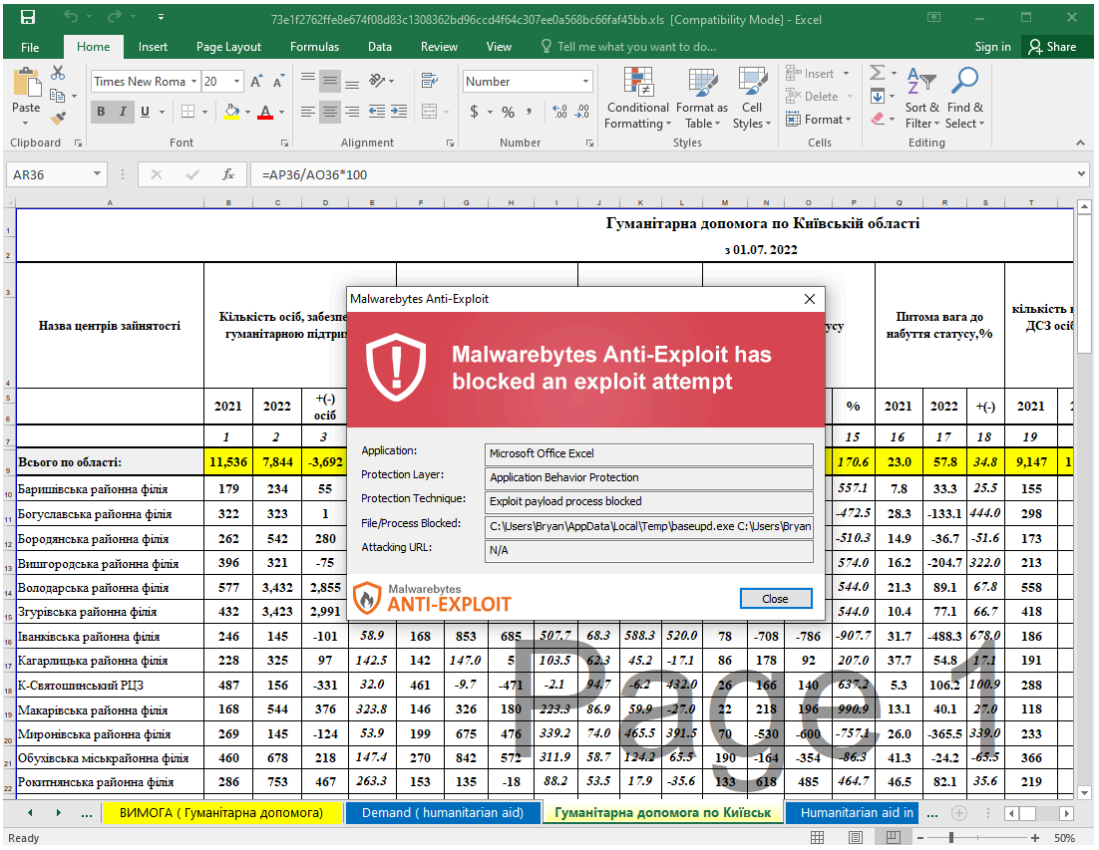
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
46  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
47    runtime_morestack_noctxt();
48  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
49  ((void (*)(void))loc_45CDA0)();
50  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
51  v38 = v4;
52  v30 = v5;
53  v31 = v6;
54  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
55  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
56  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
57  v32 = v10;
58  v11 = *(void (__cdecl **)(int))(v10 + 16);
59  v39 = v12;
60  v11(v12);
61  v29 = v13;
62  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
63  v40 = v15;
64  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
65  v17 = v27;
66  if ( v27 )
67  {
68    if ( v27 )
69      v17 = *(_DWORD *)(v27 + 4);
70    v43 = v17;
71    v44 = v28;
72    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
73  }
```



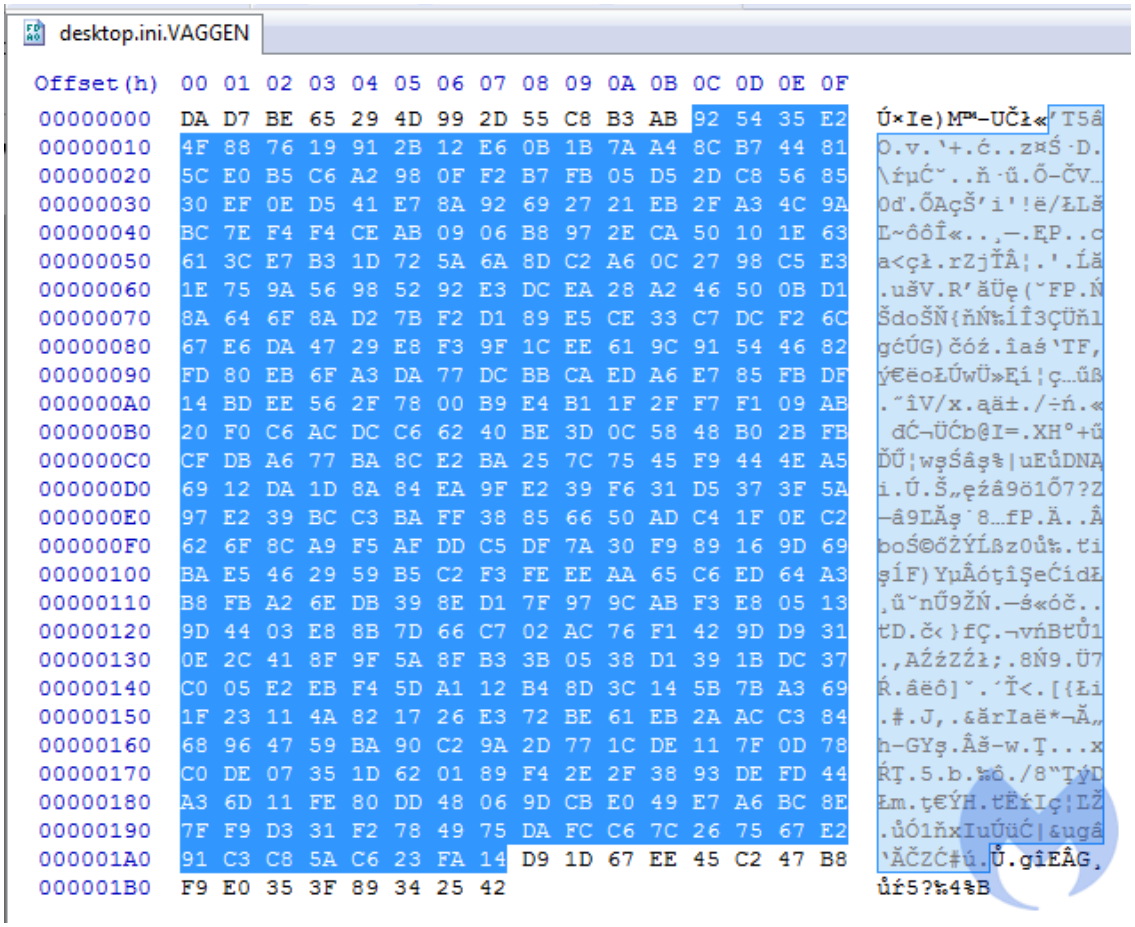
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



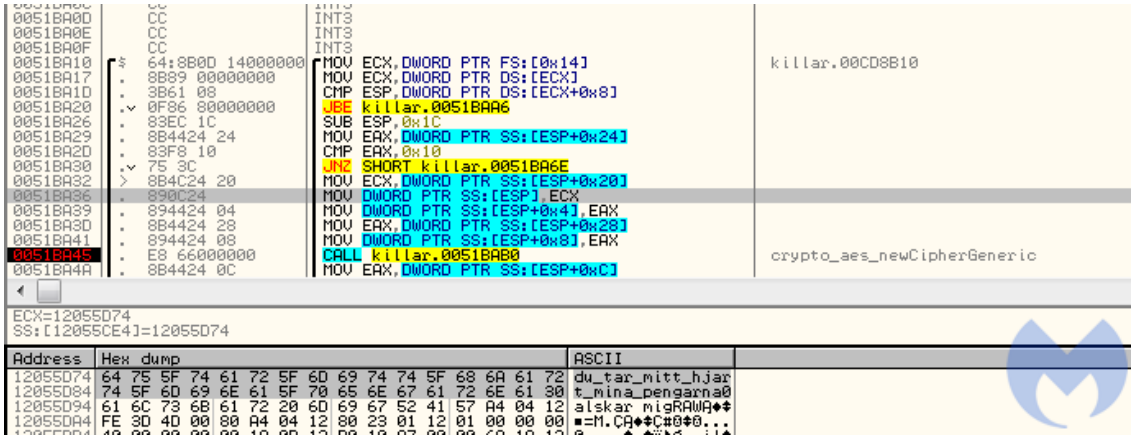
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

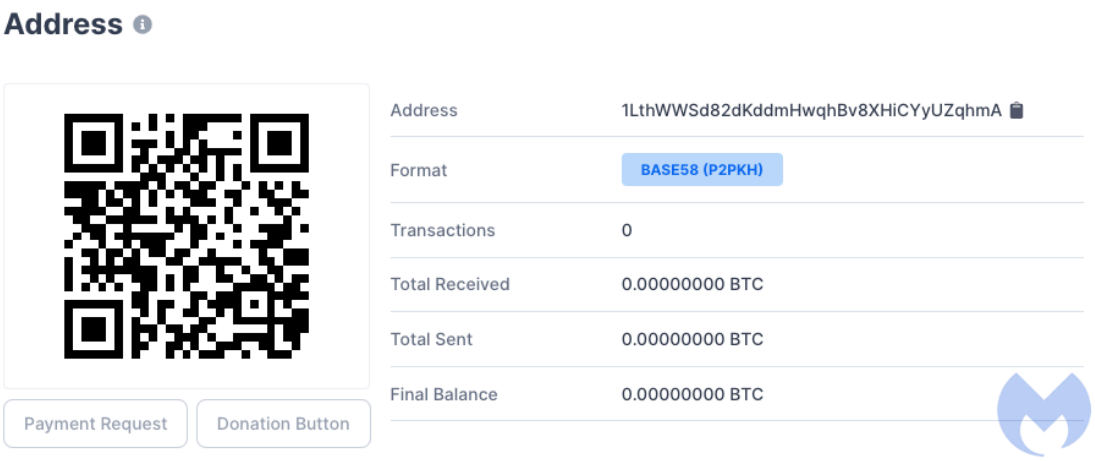
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

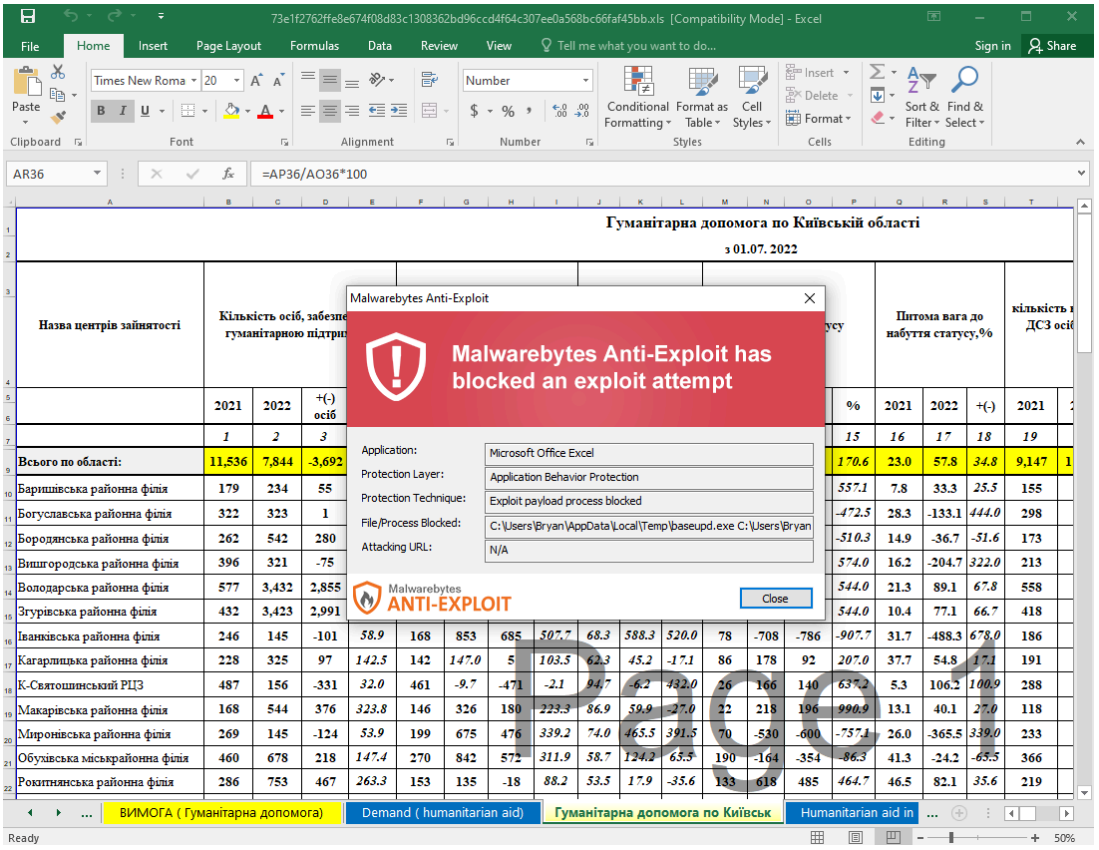
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

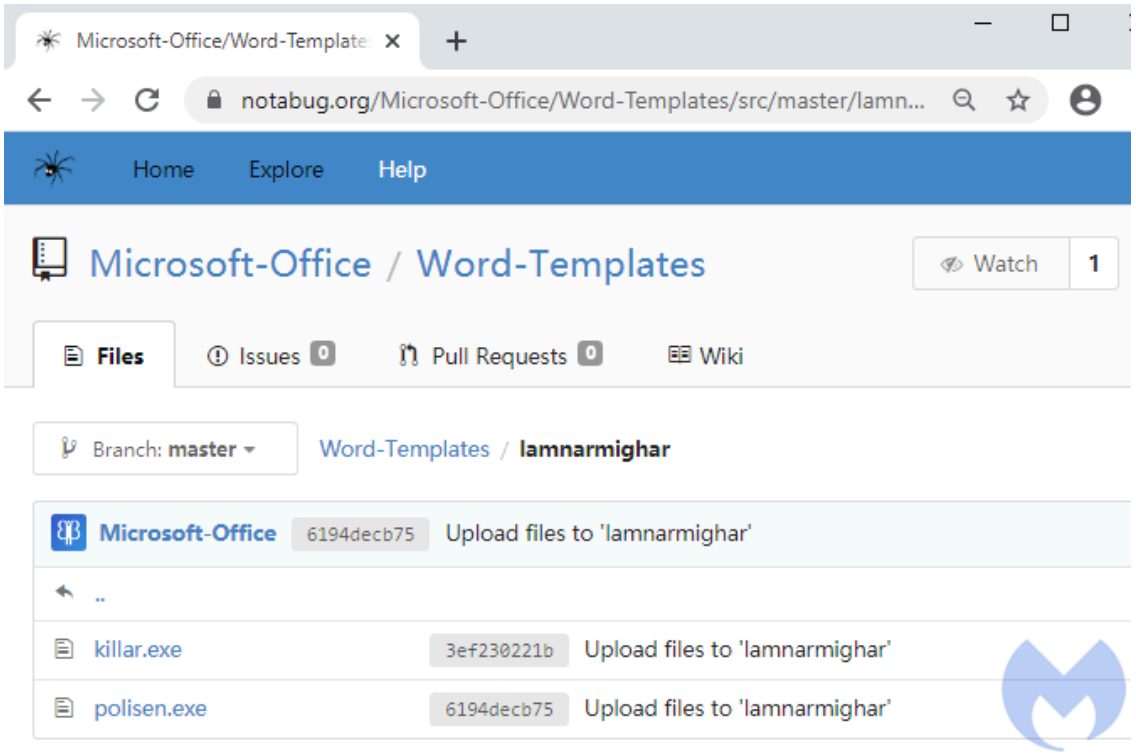
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



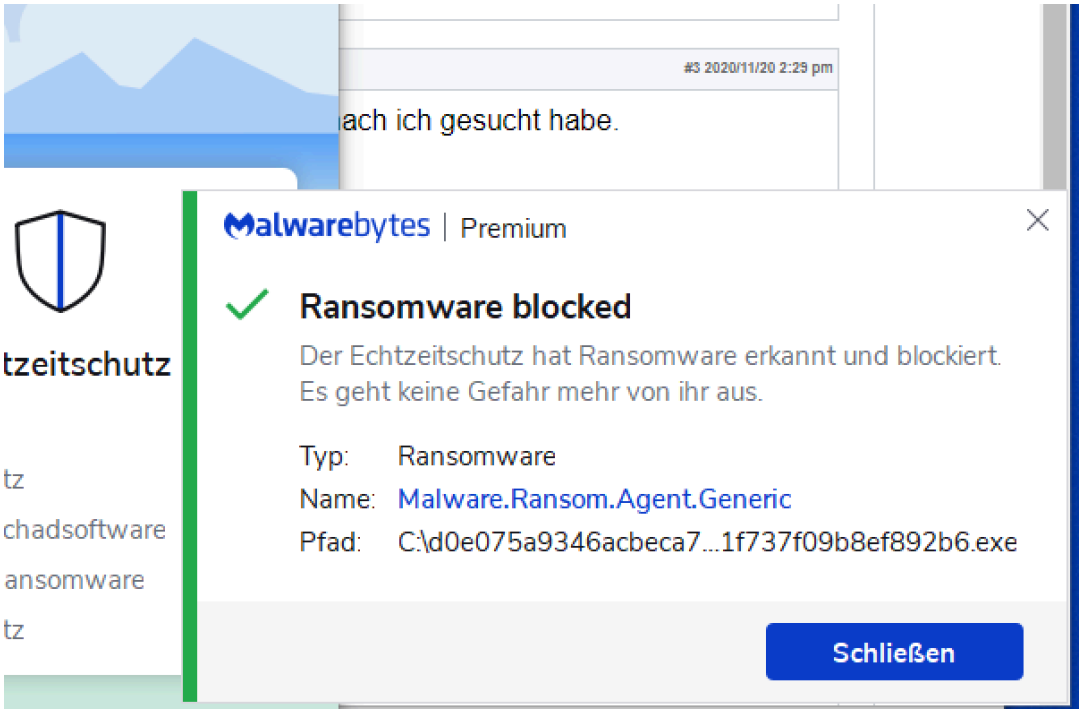
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

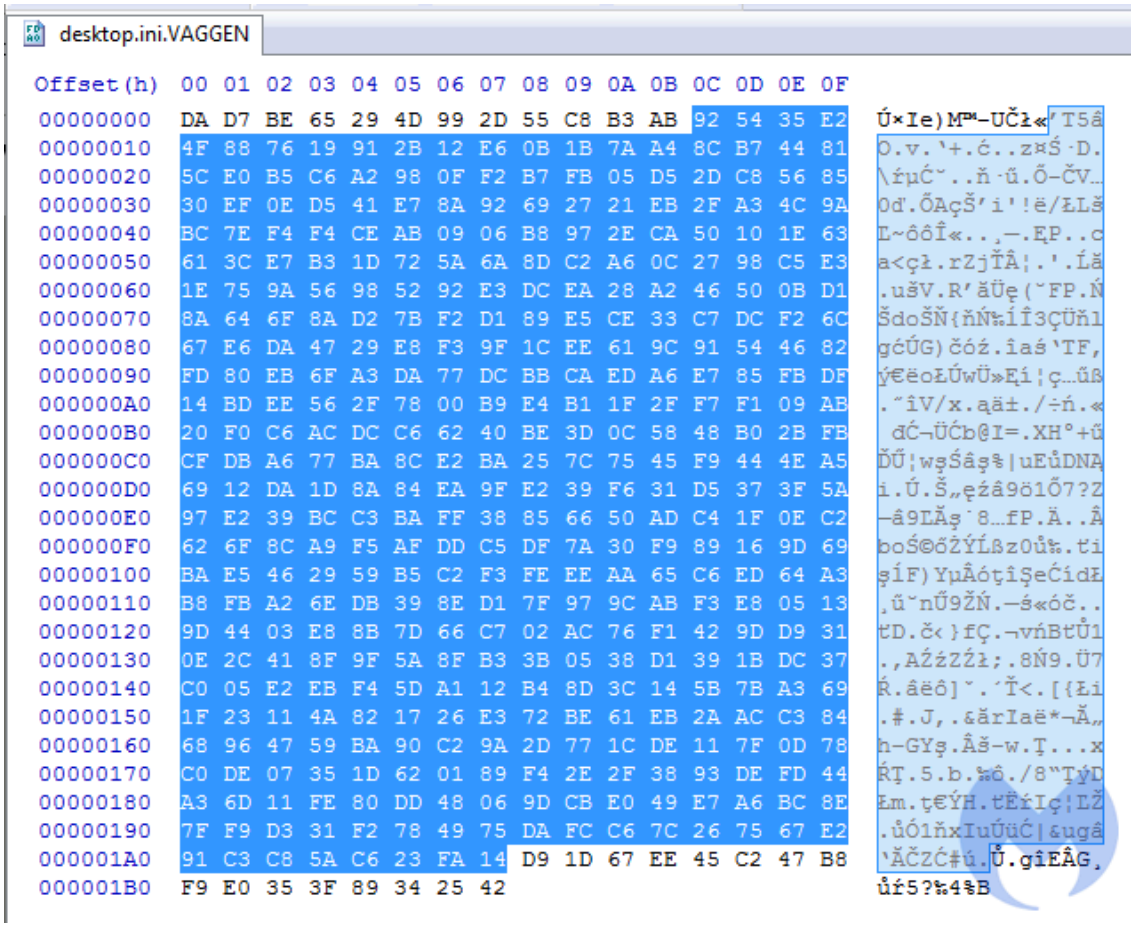
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

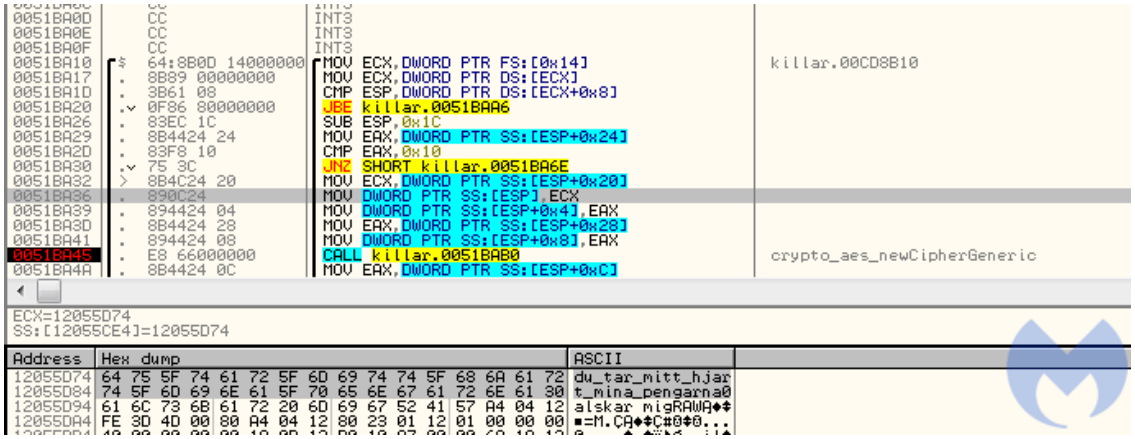


The content of the output file (with .VAGGEN extension) contains:

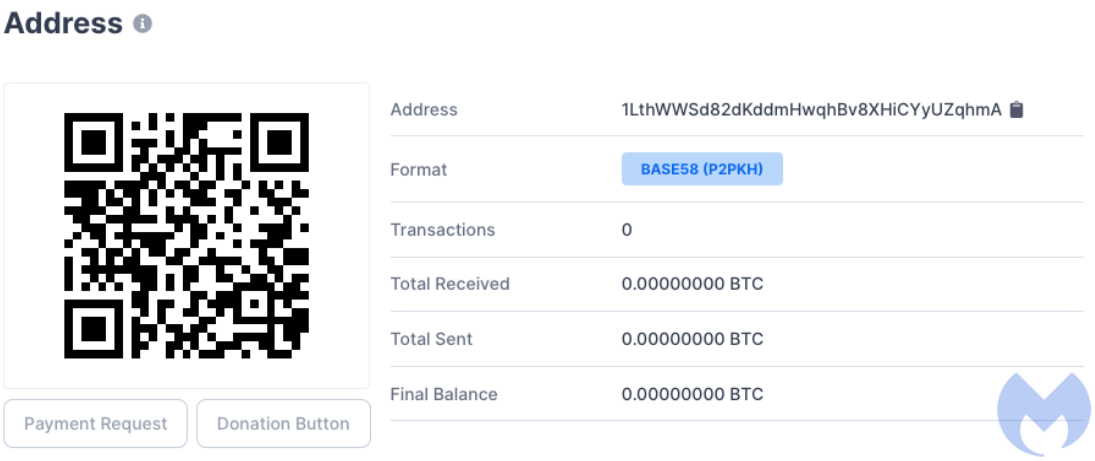
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

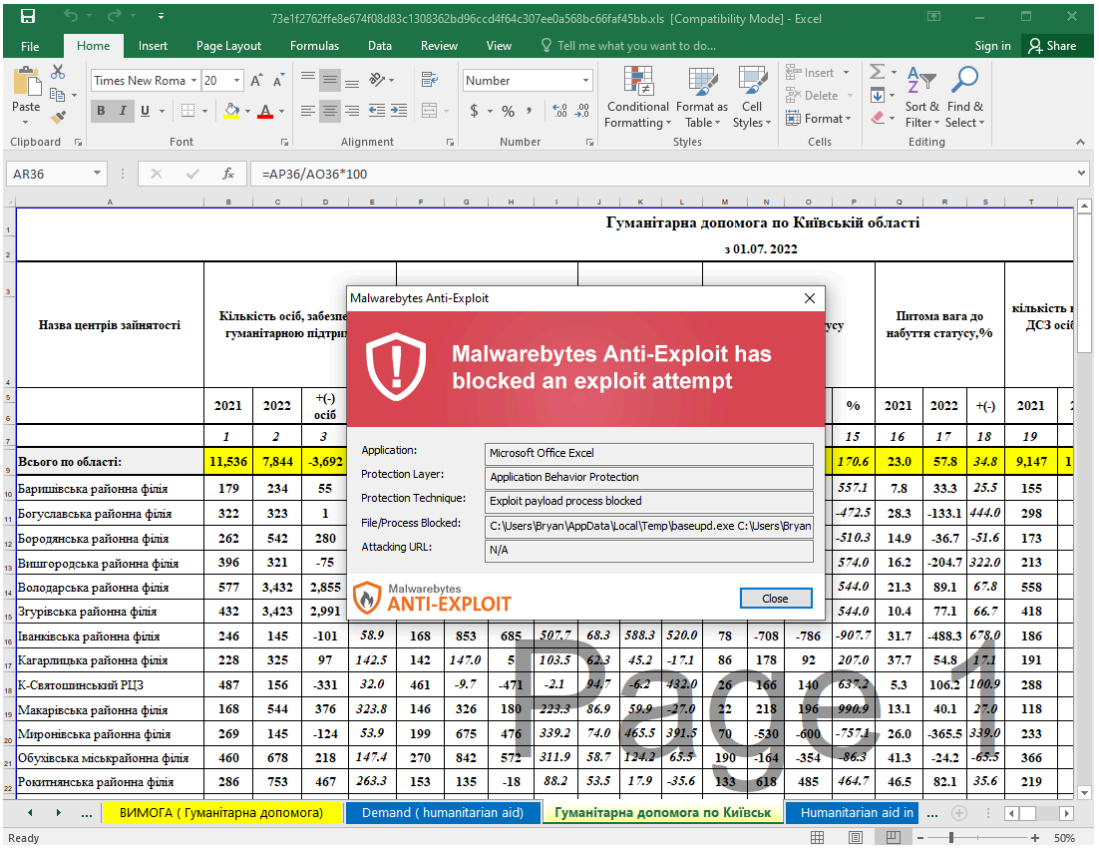
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

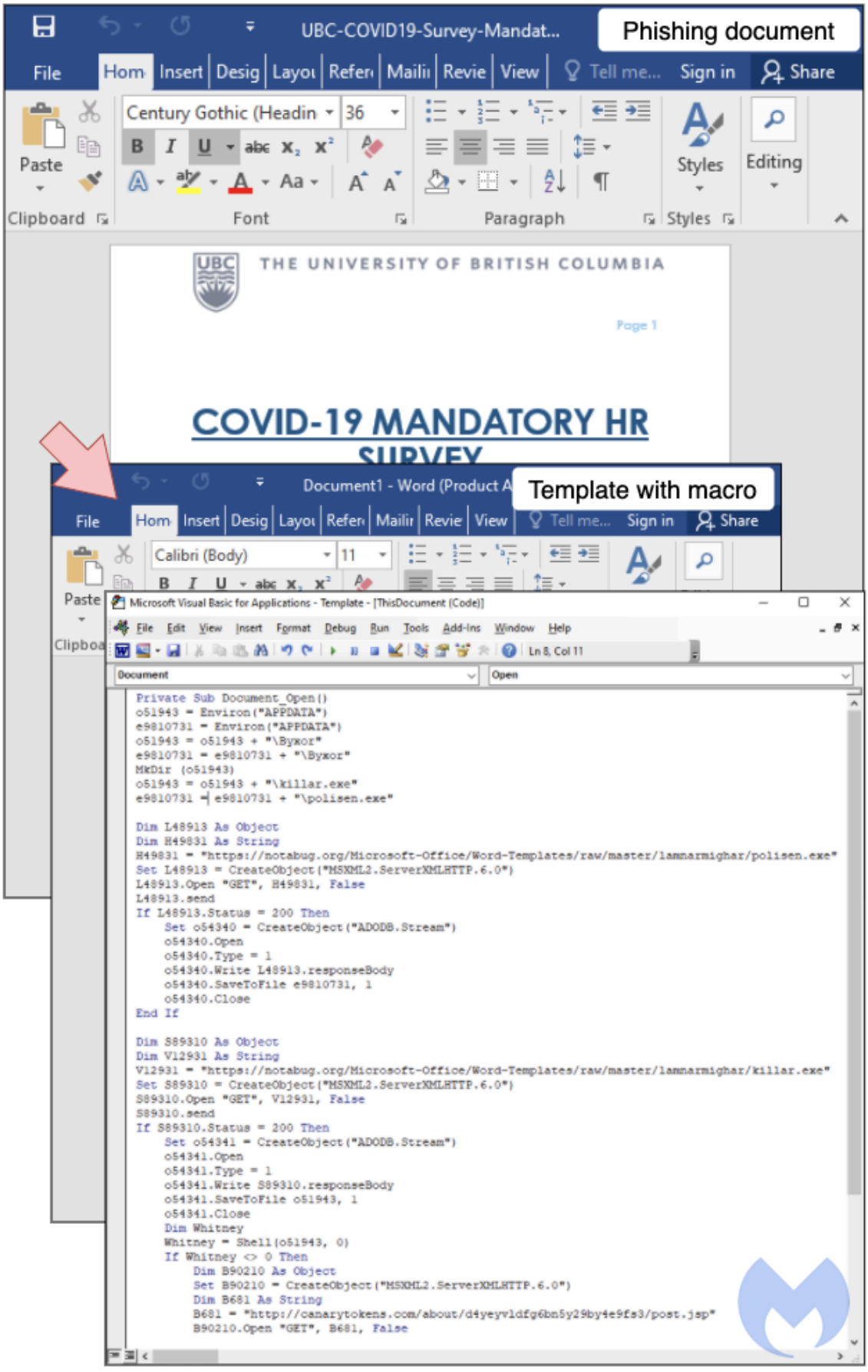
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

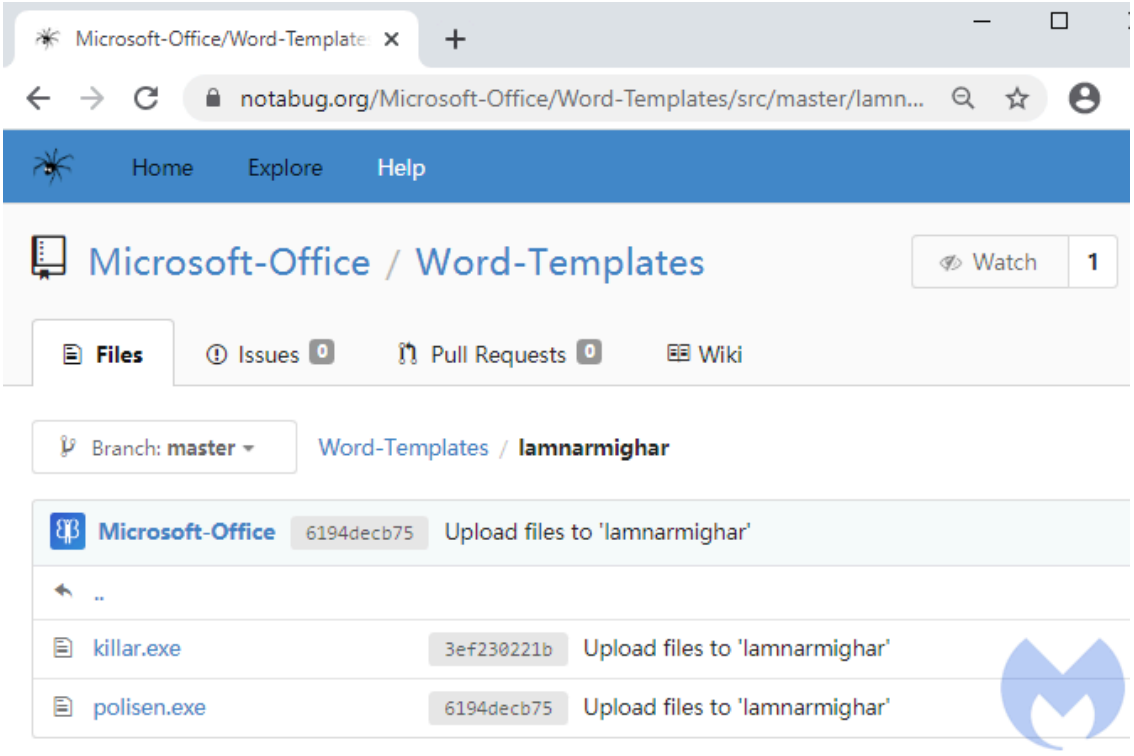
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



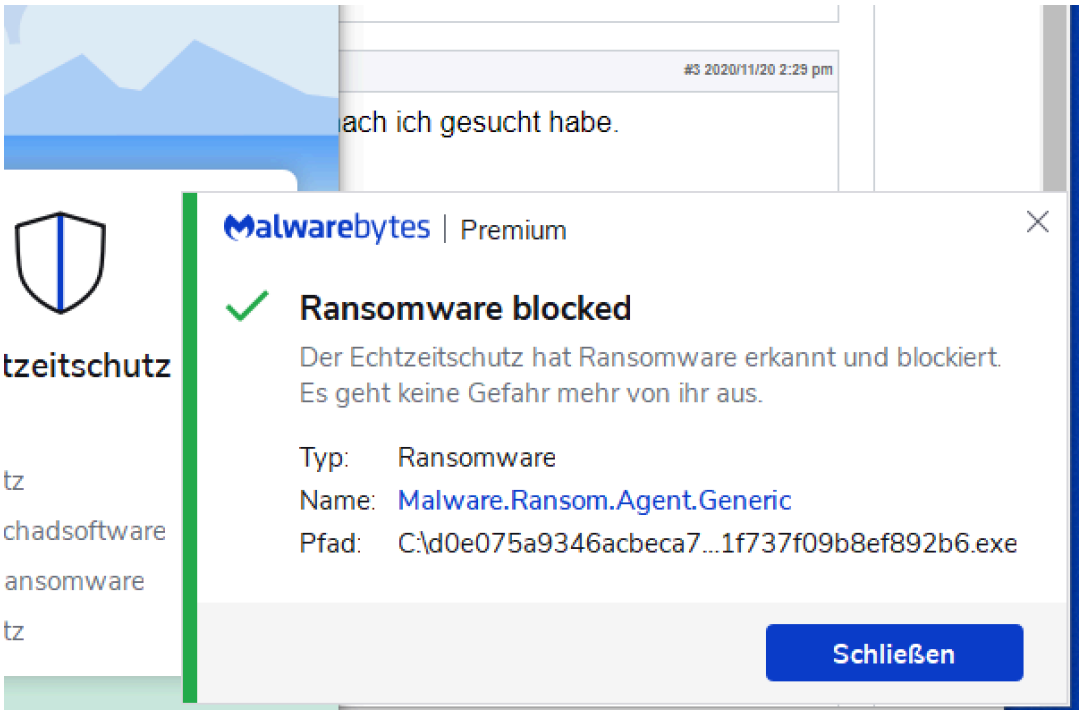
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

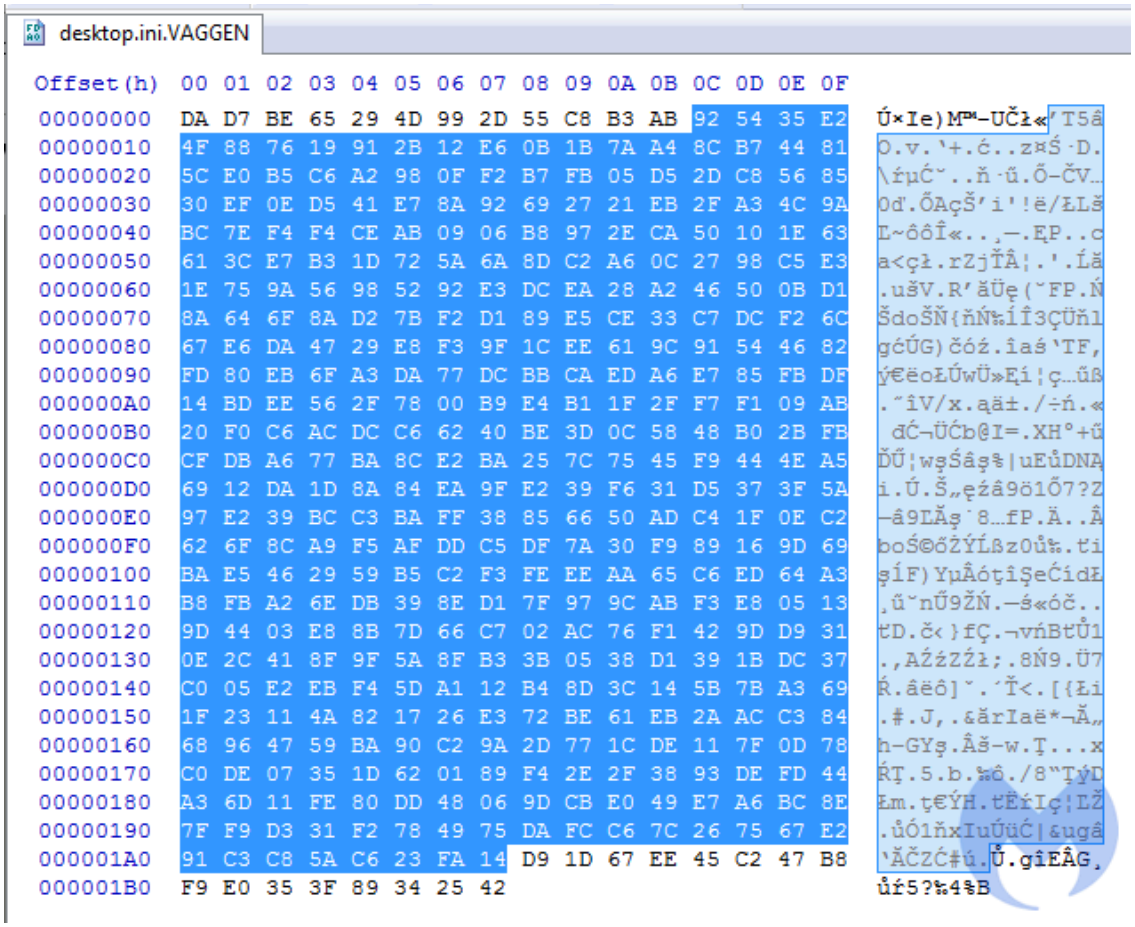
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

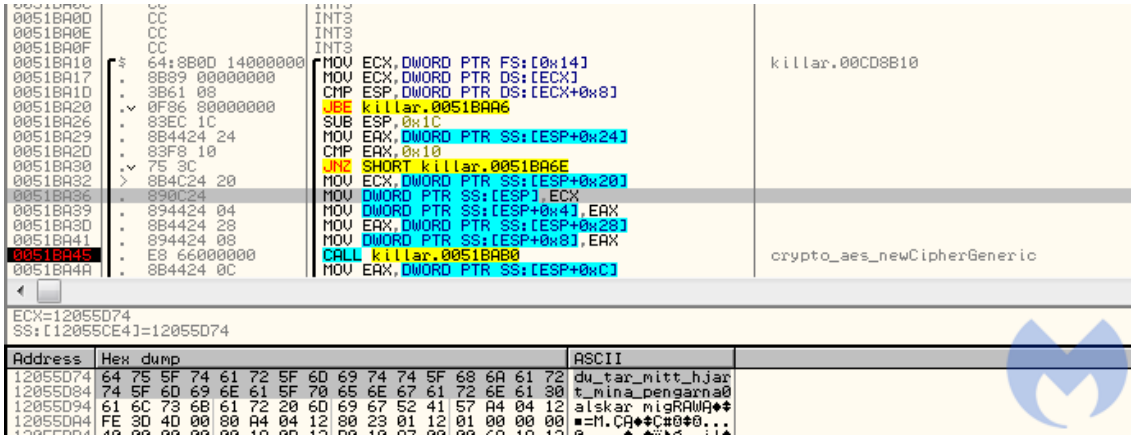


The content of the output file (with .VAGGEN extension) contains:

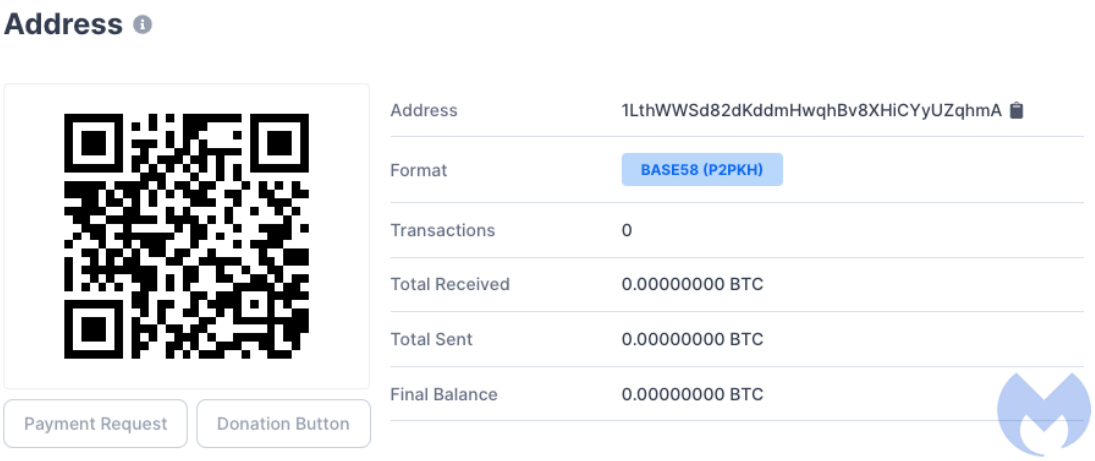
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

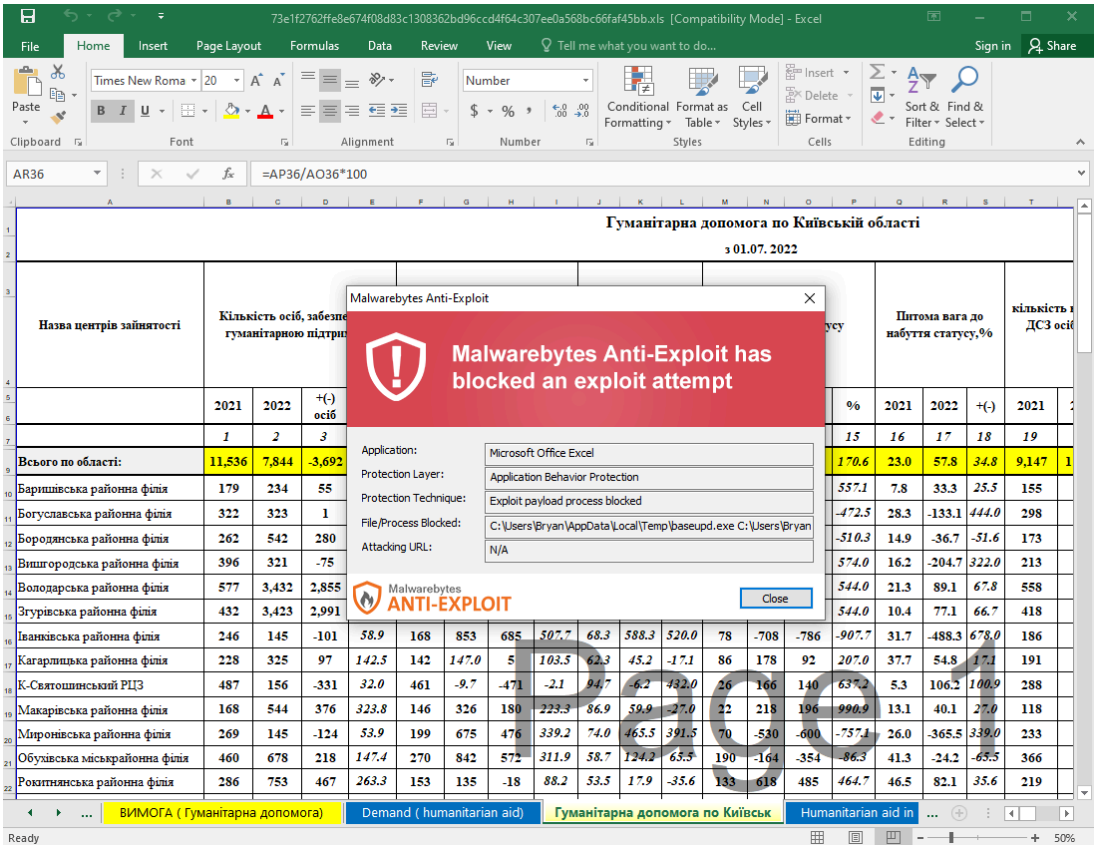
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

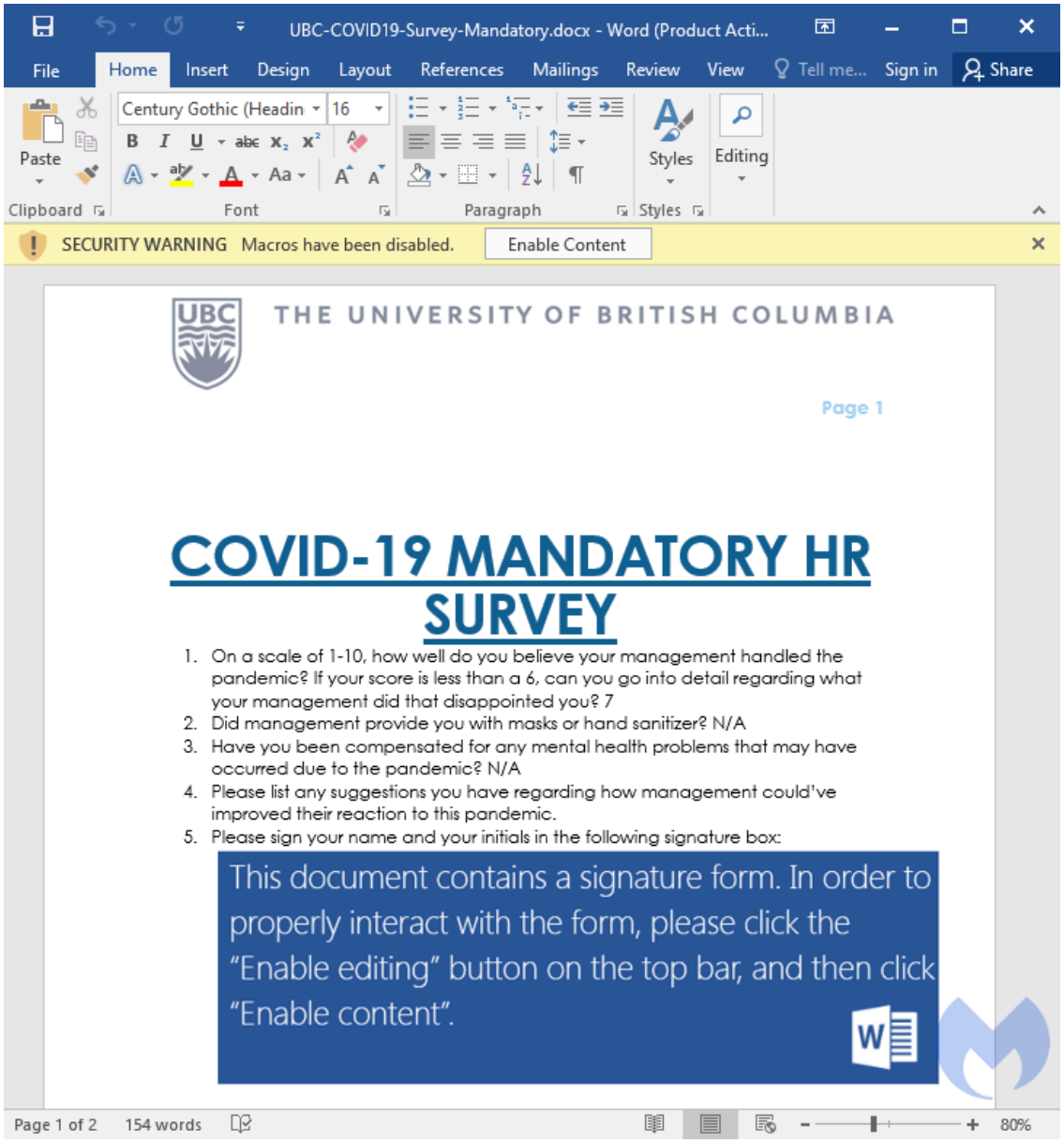
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

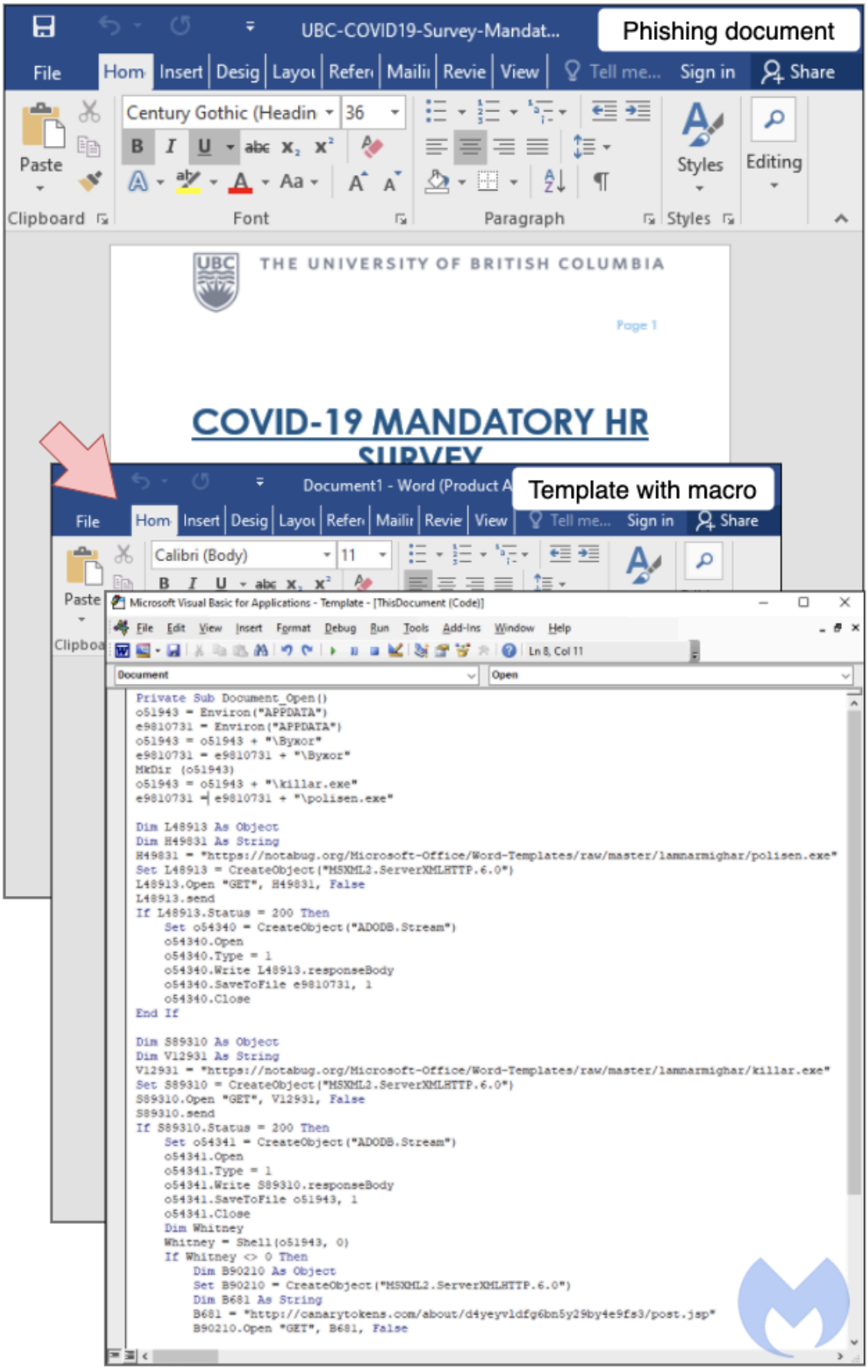
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

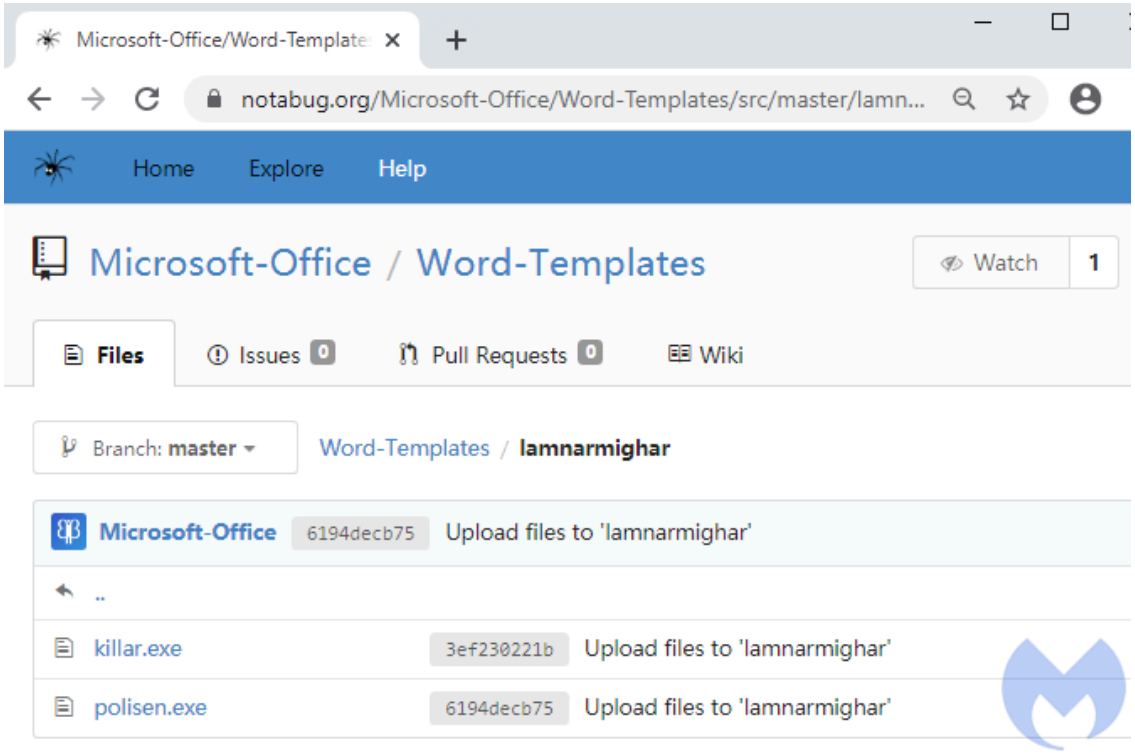
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



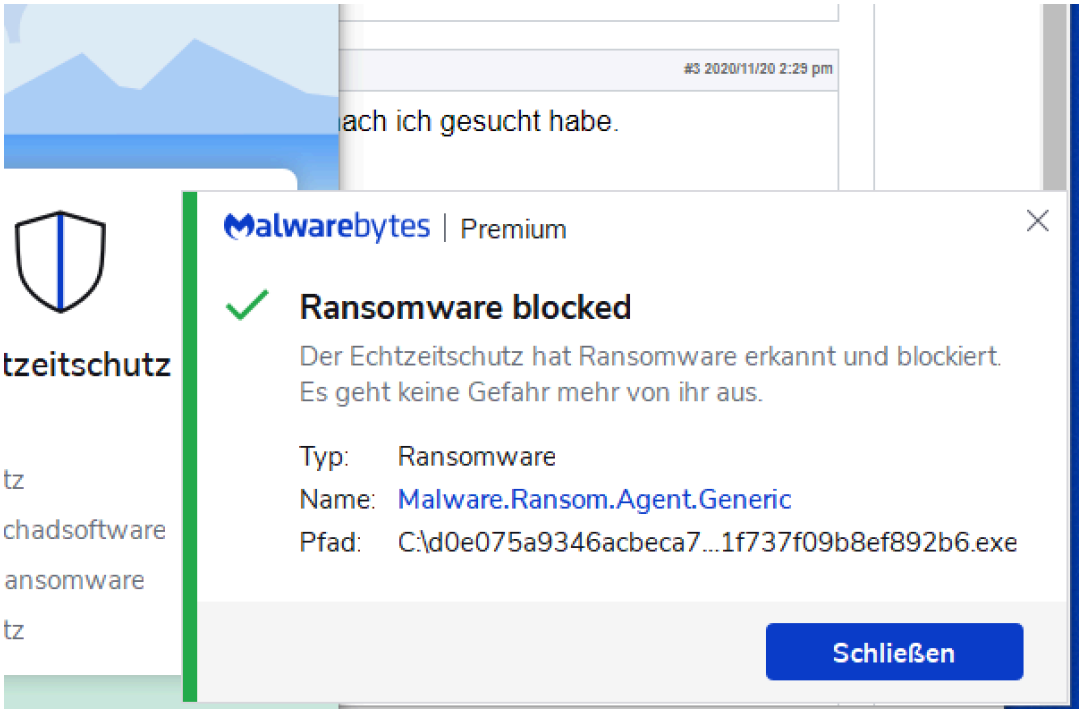
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

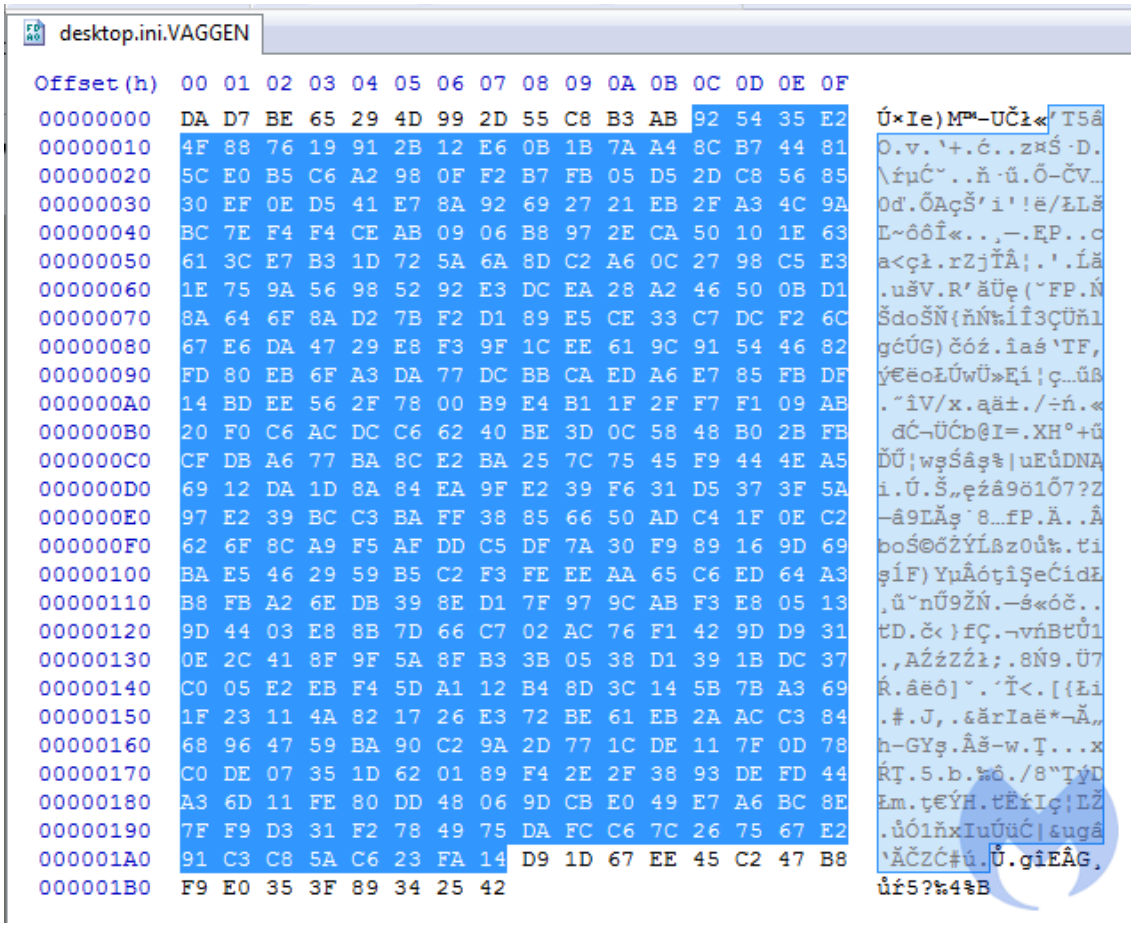
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

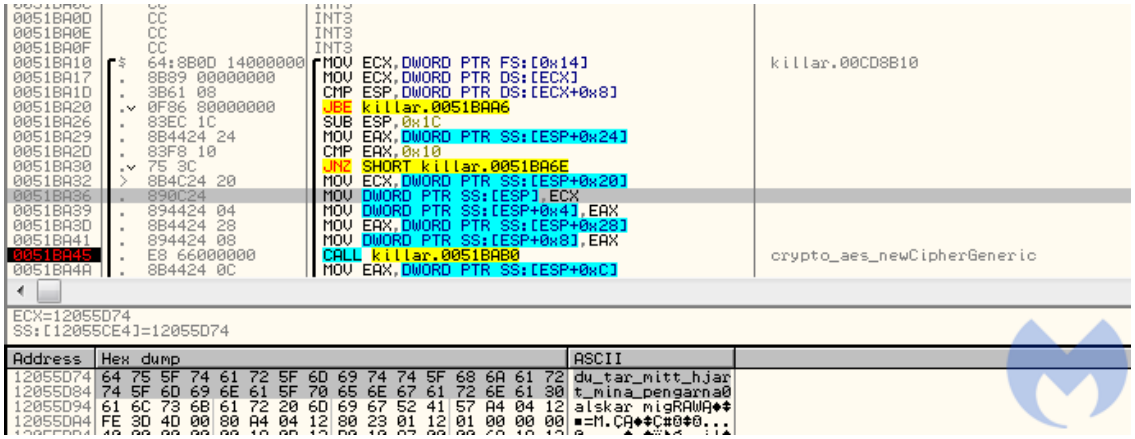


The content of the output file (with .VAGGEN extension) contains:

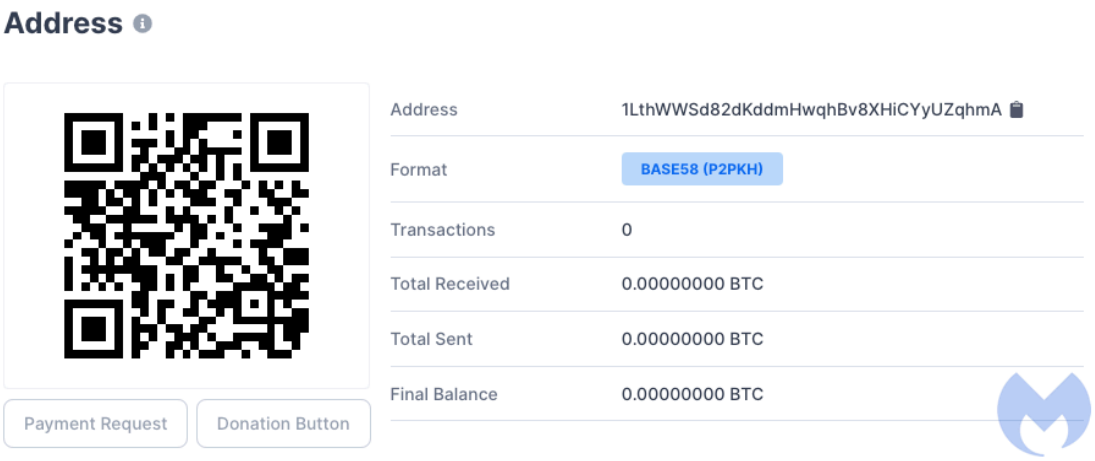
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

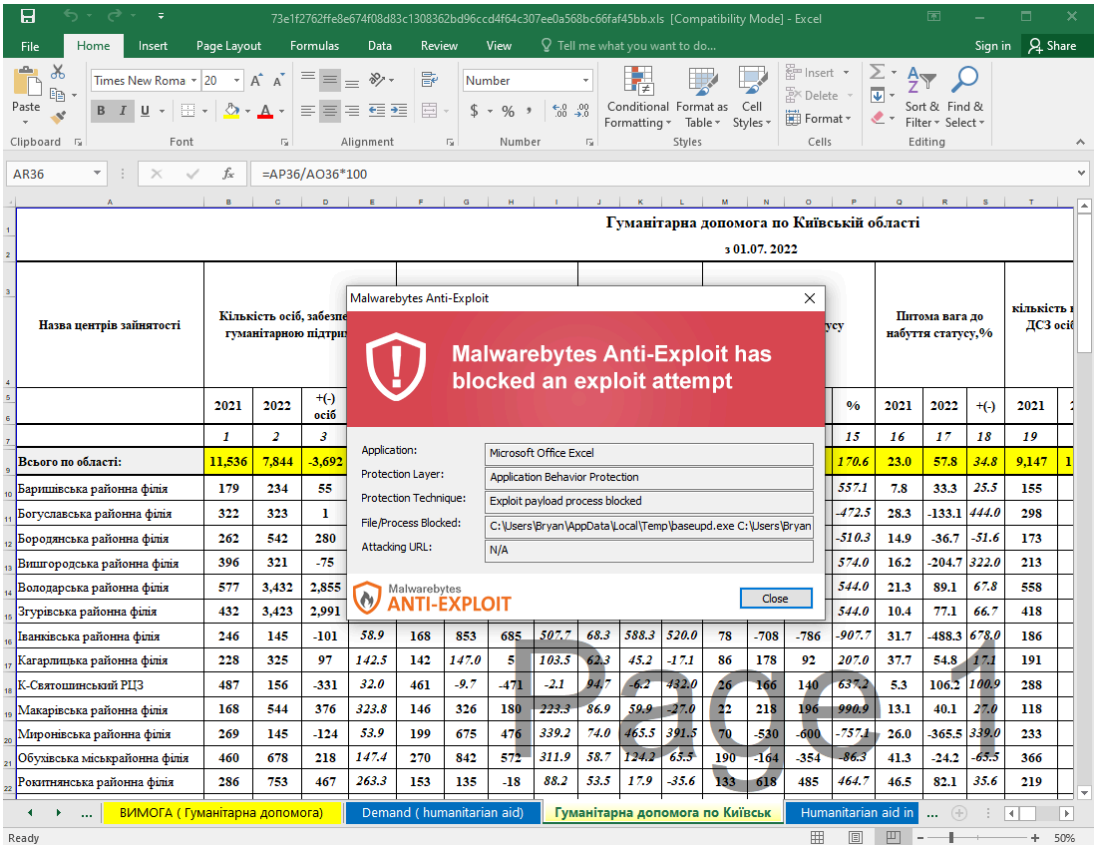
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

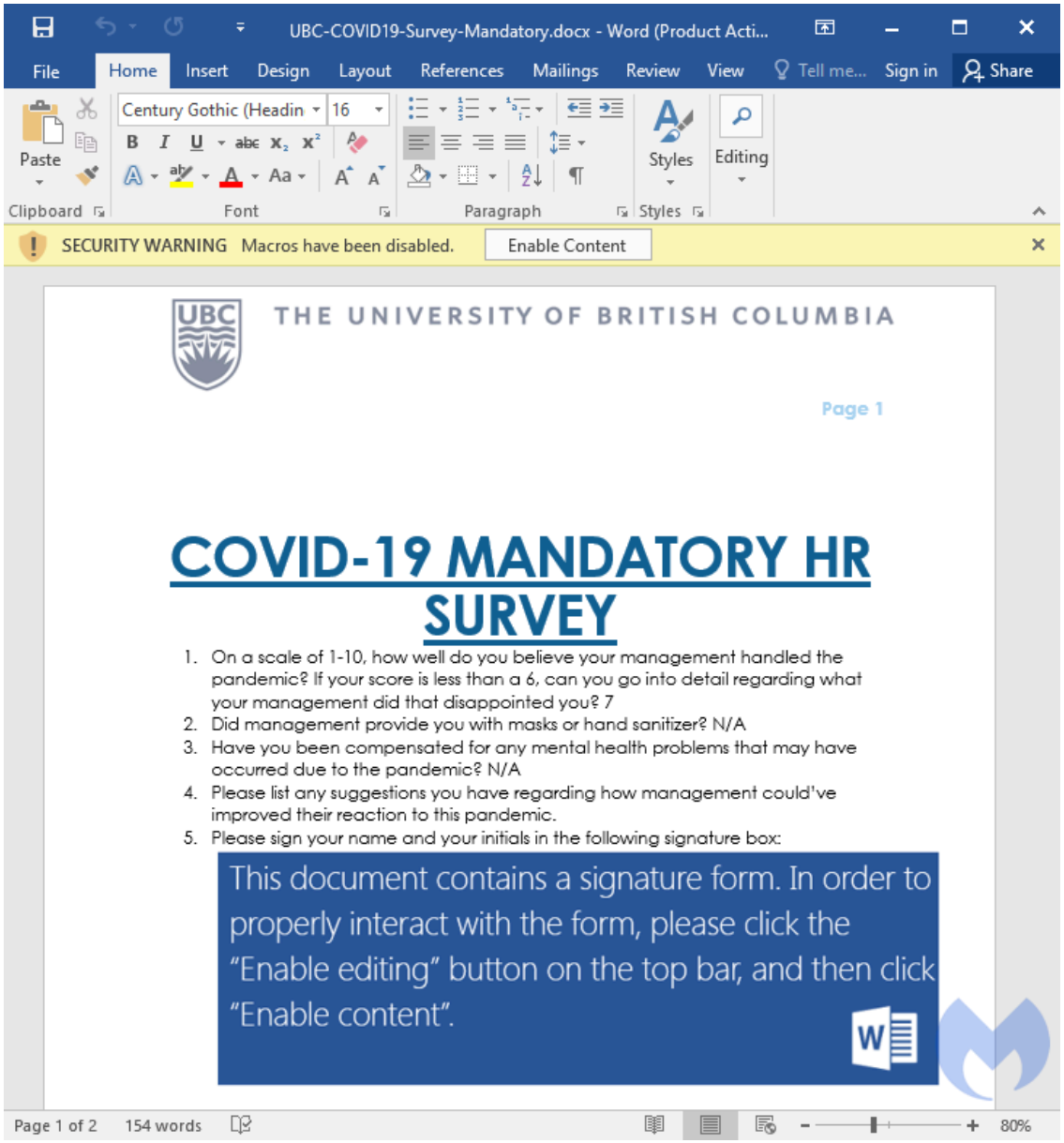
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

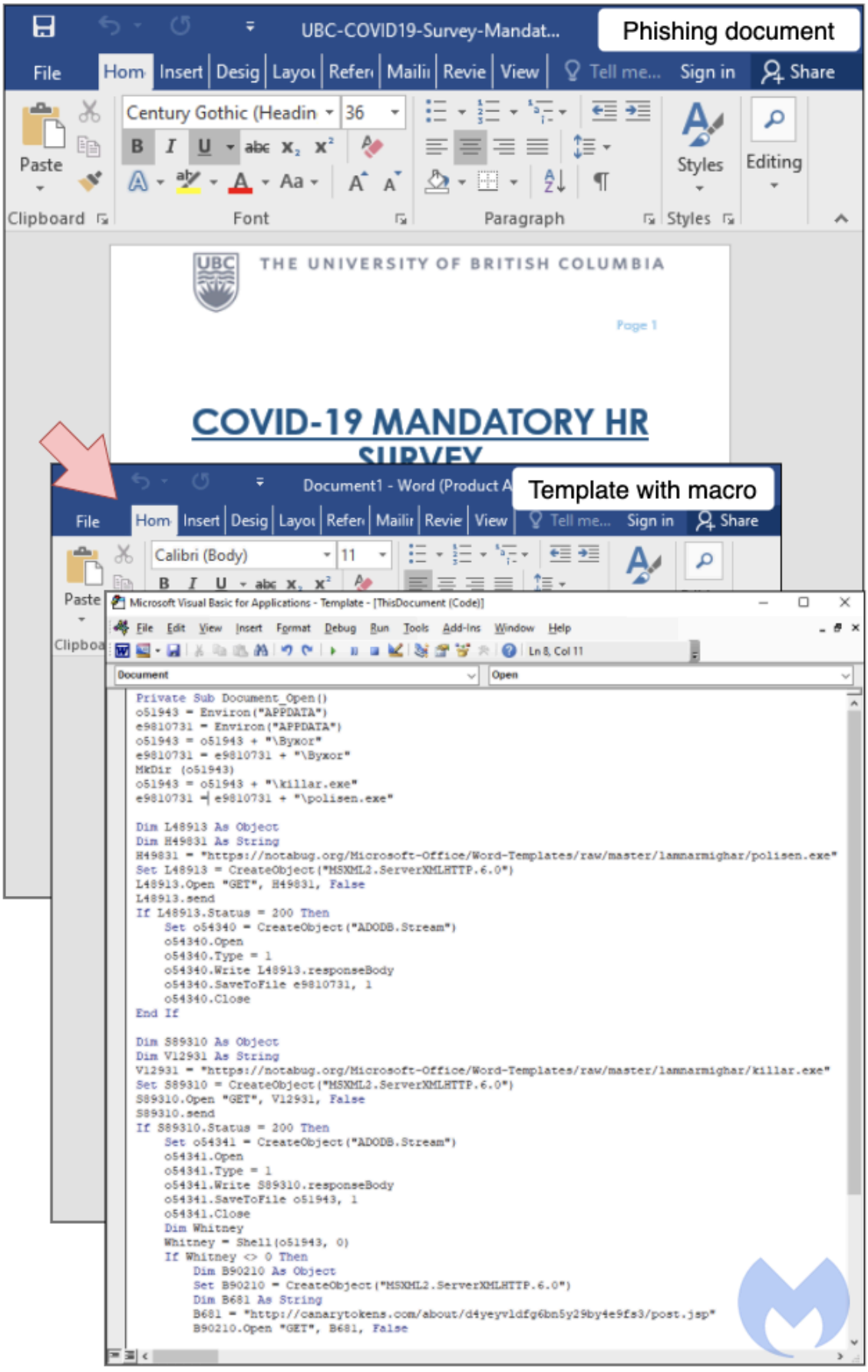
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

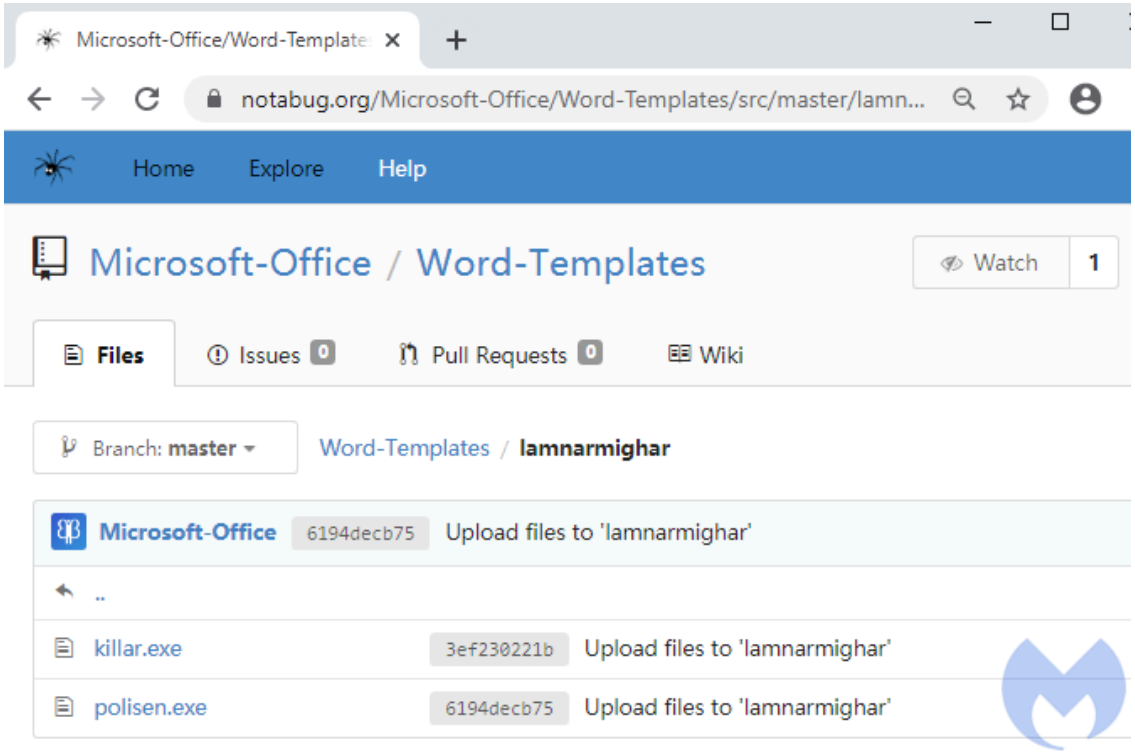
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



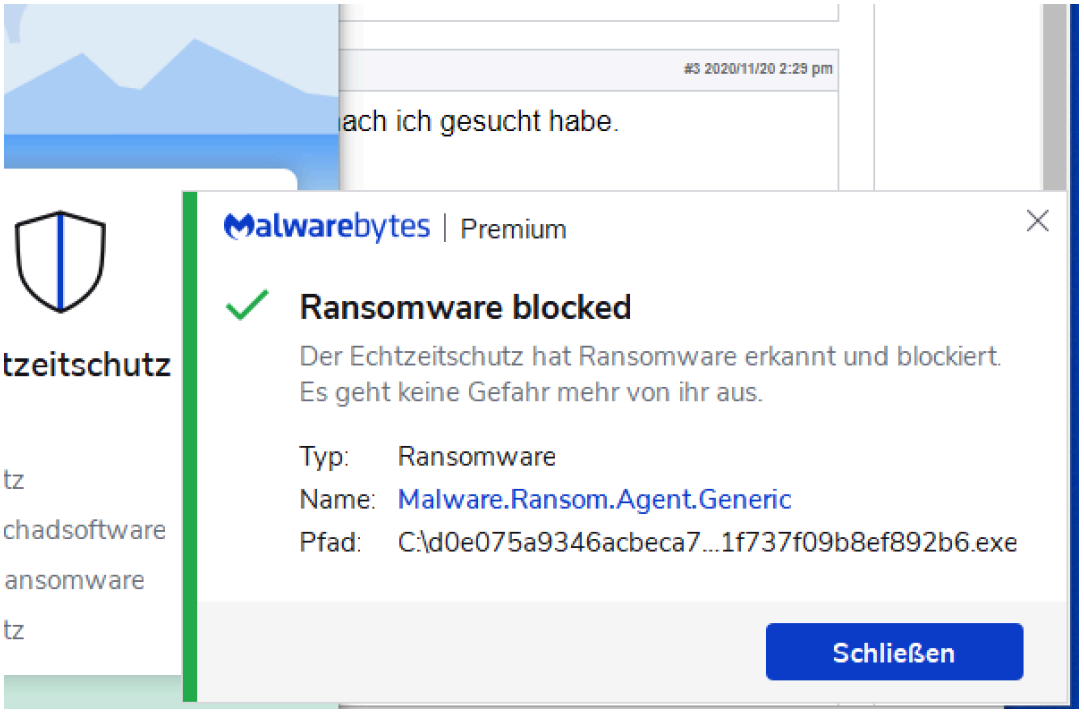
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

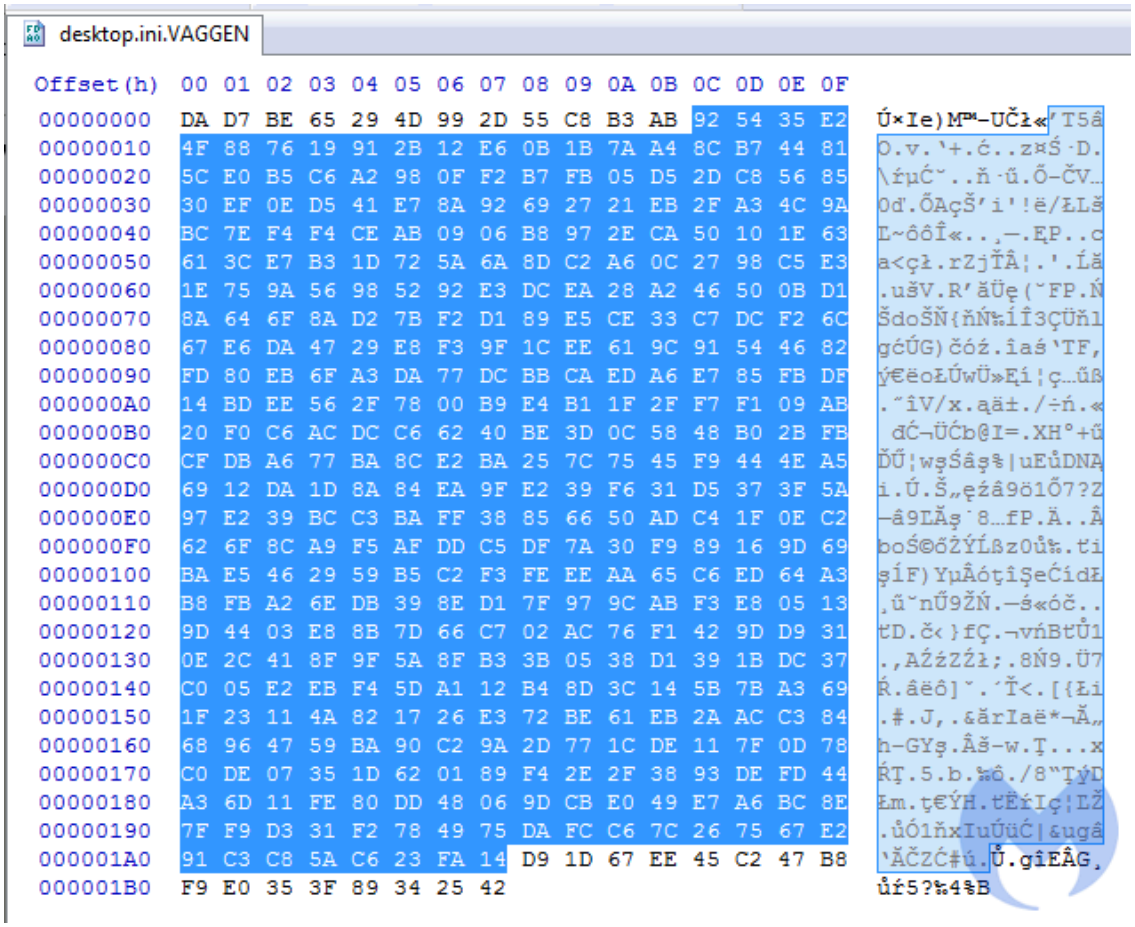
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

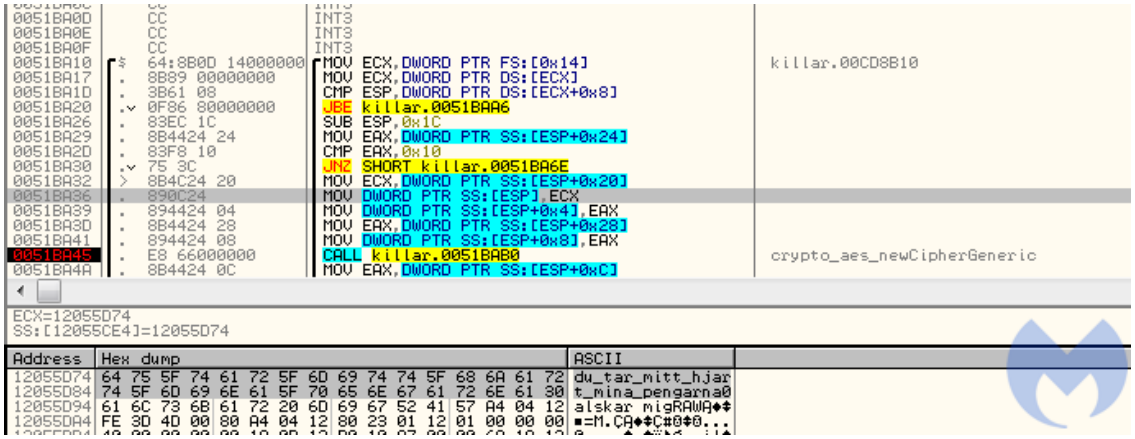


The content of the output file (with .VAGGEN extension) contains:

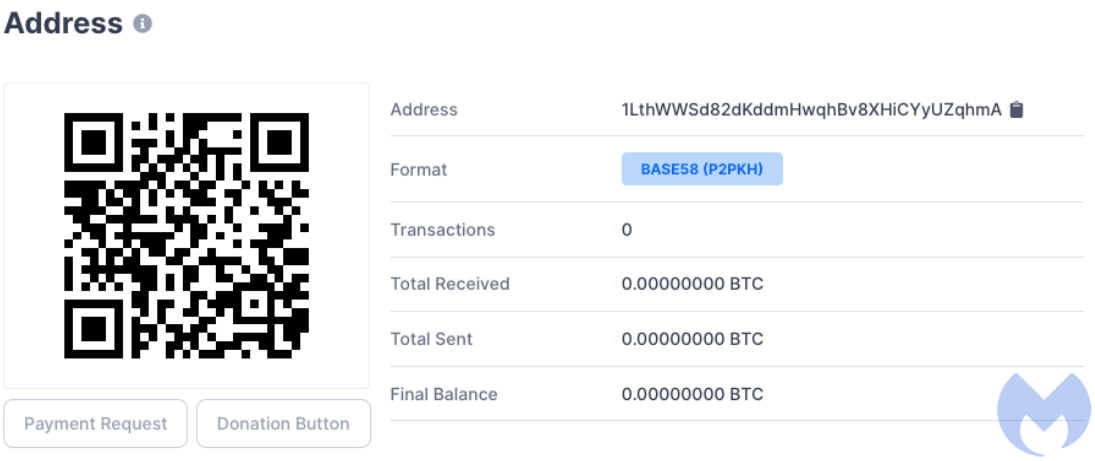
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

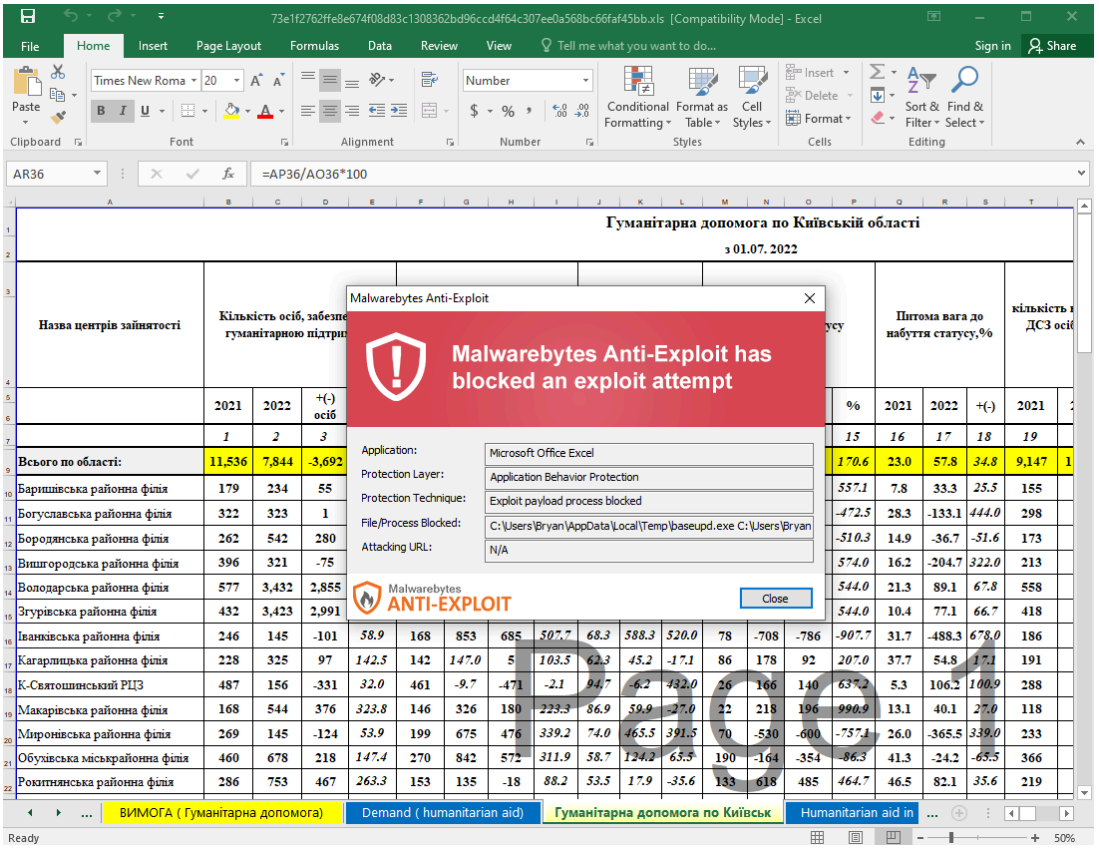
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

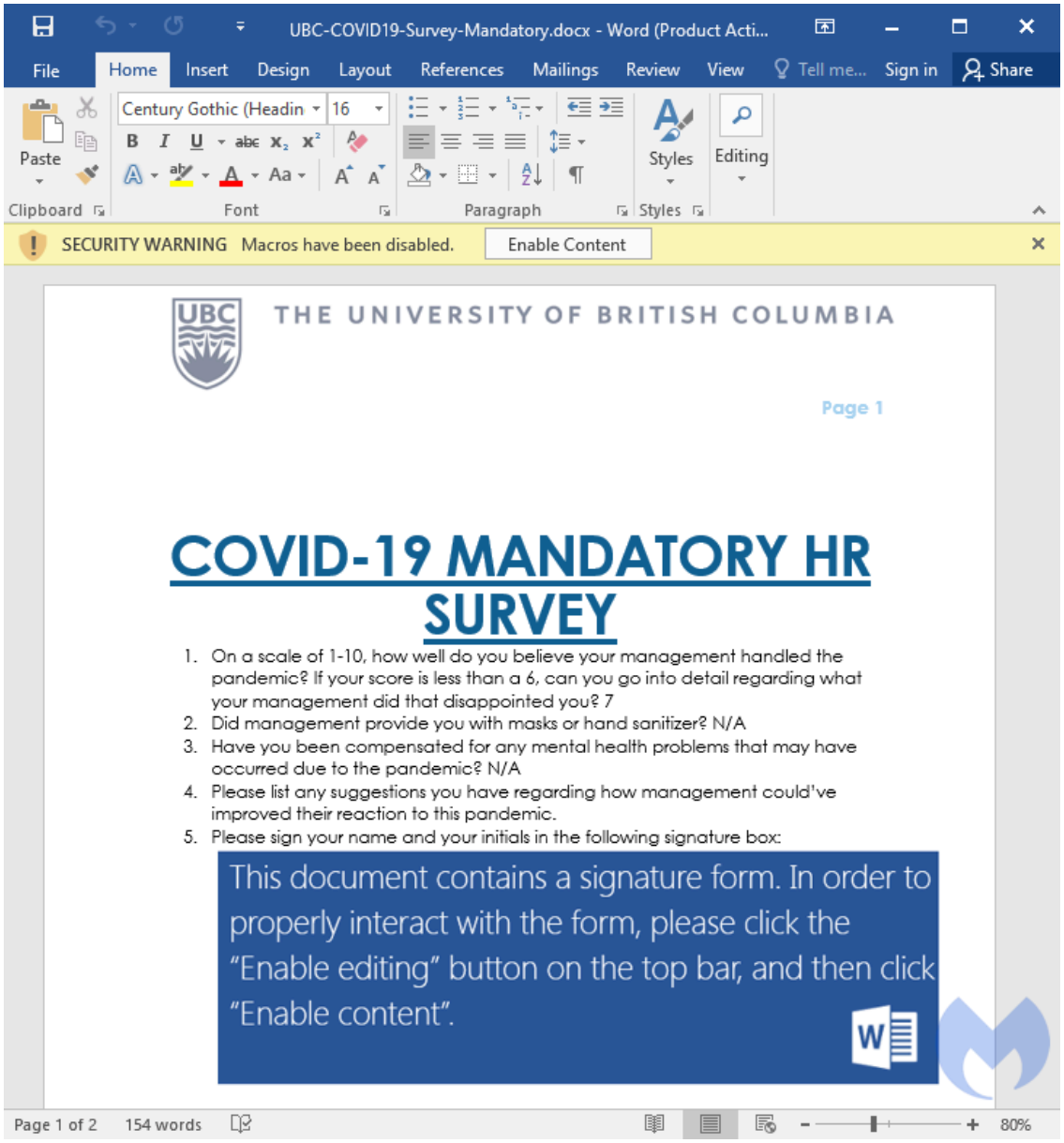
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

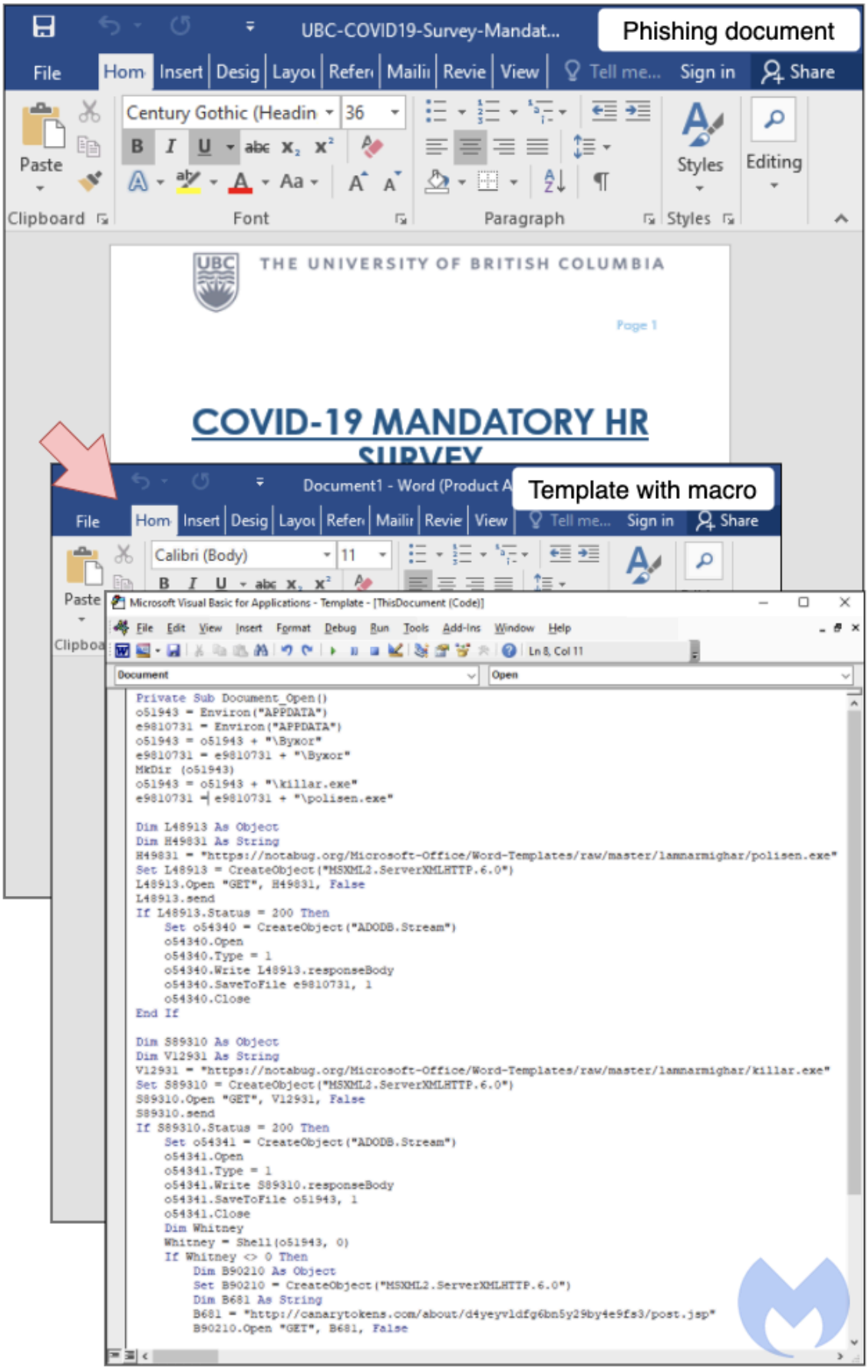
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

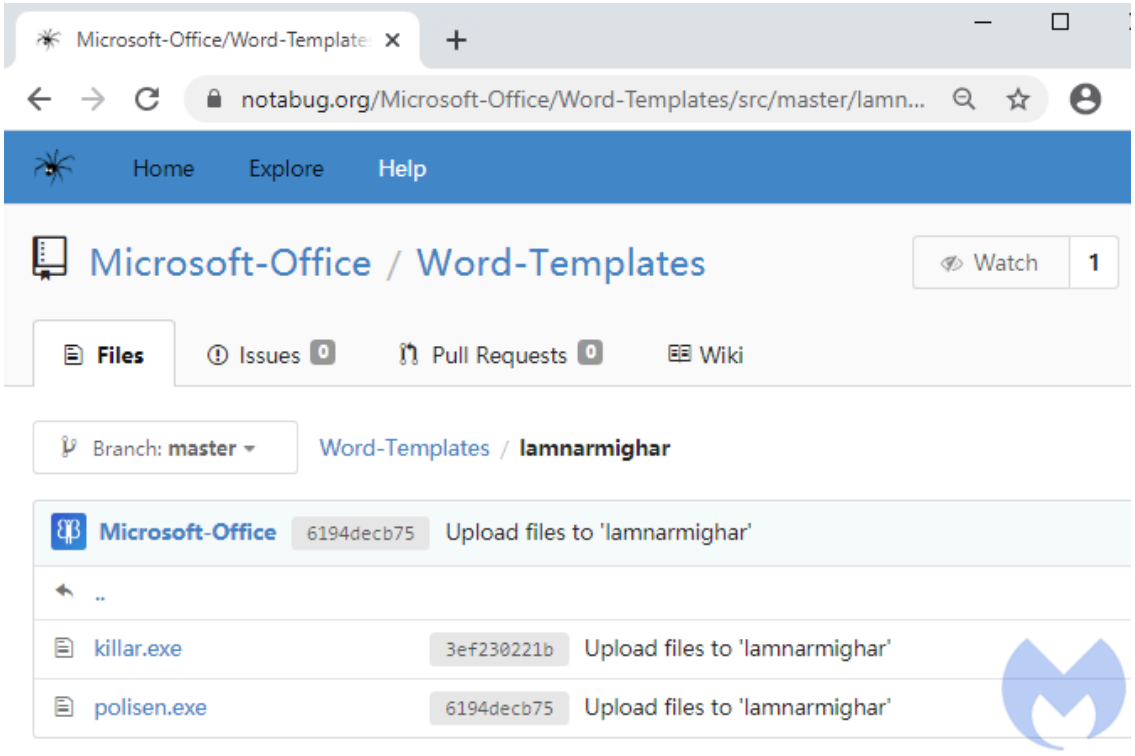
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



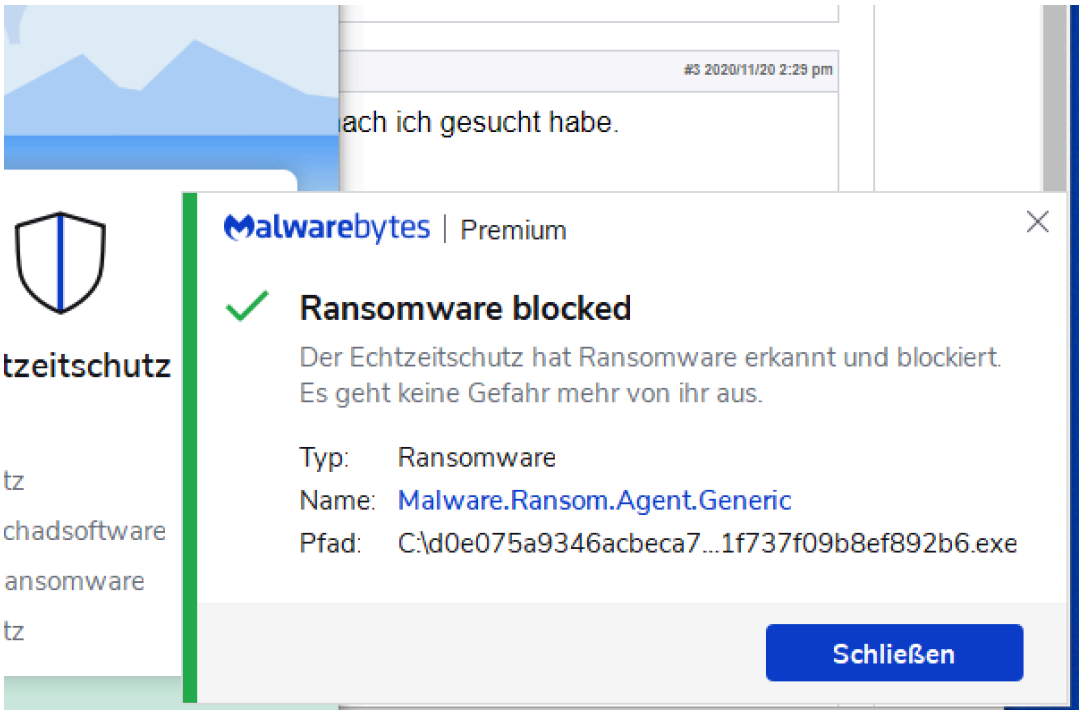
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

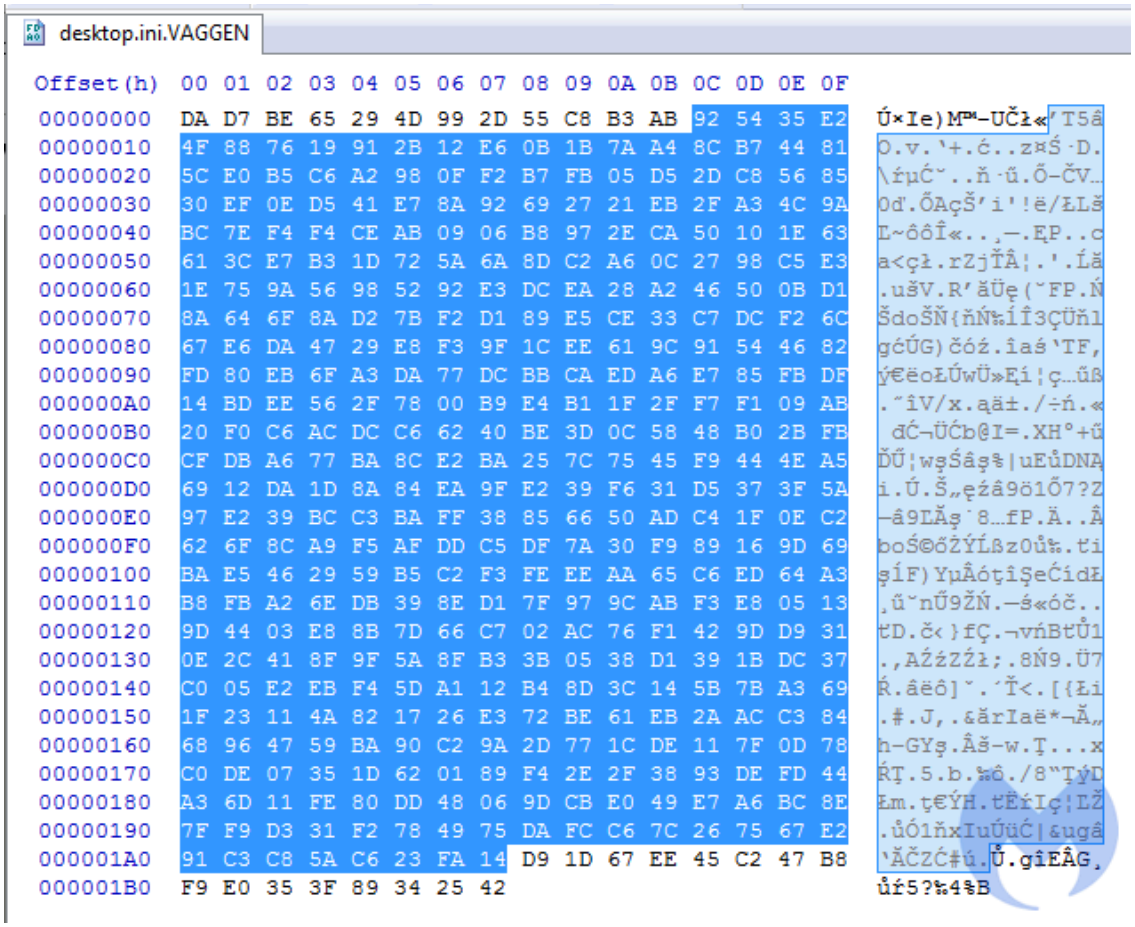
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

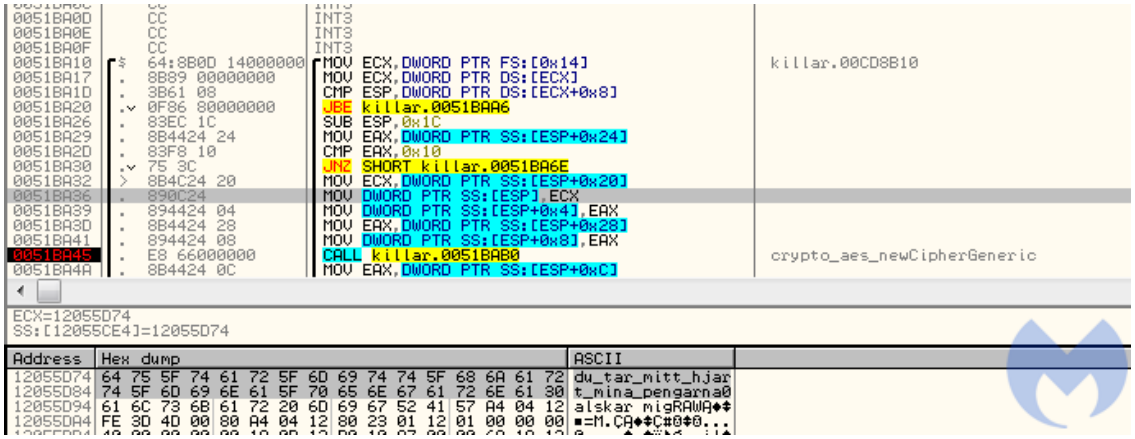


The content of the output file (with .VAGGEN extension) contains:

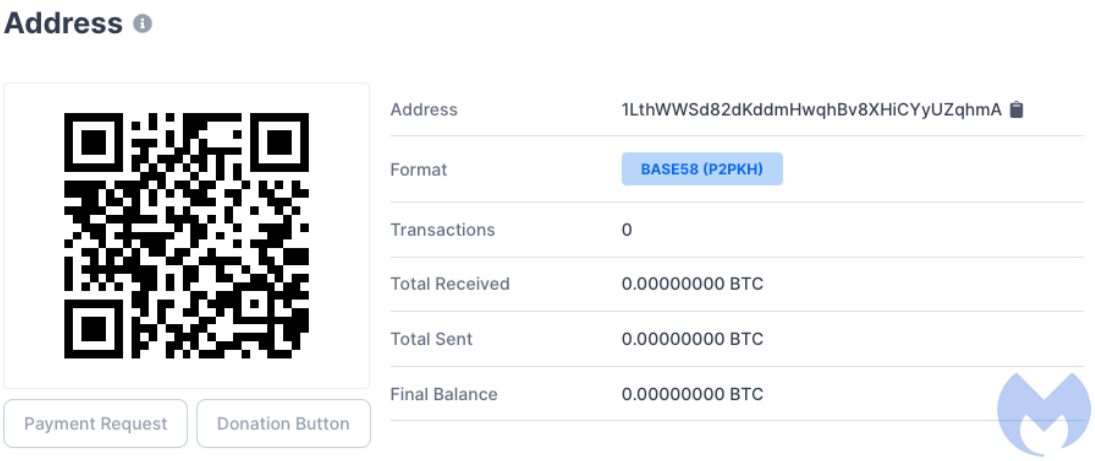
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

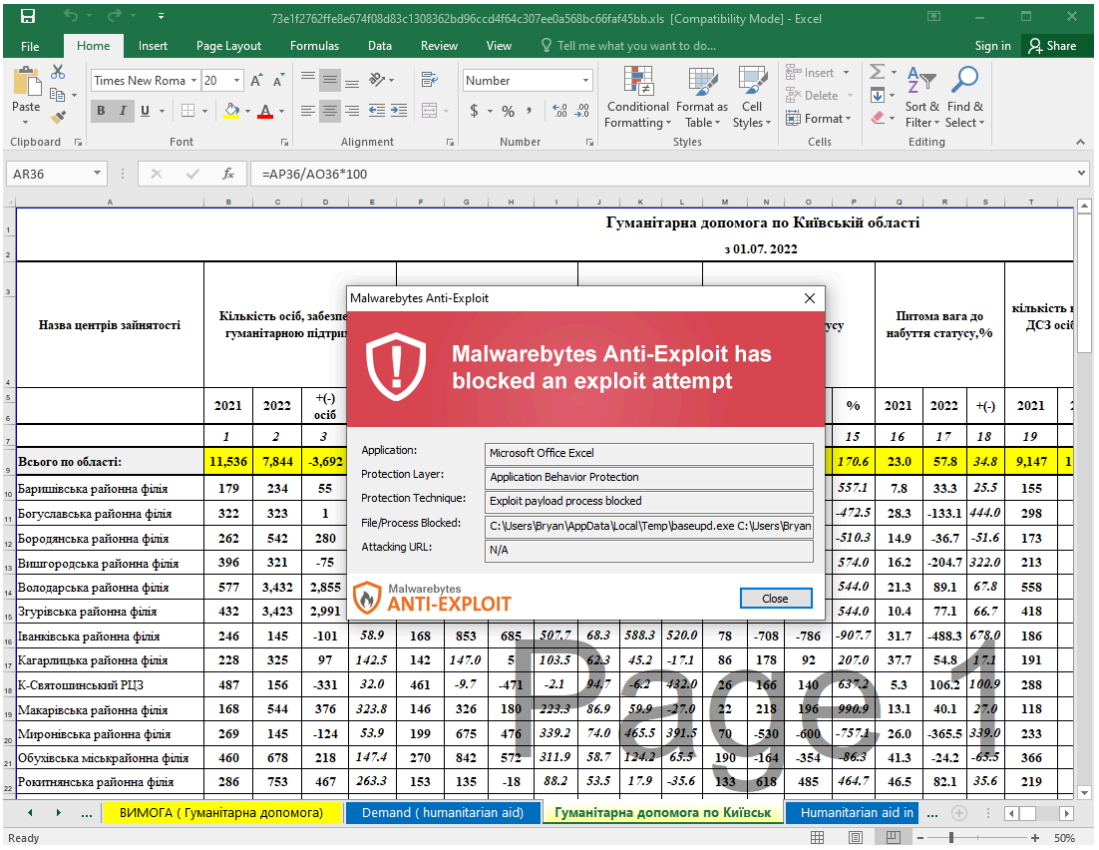
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

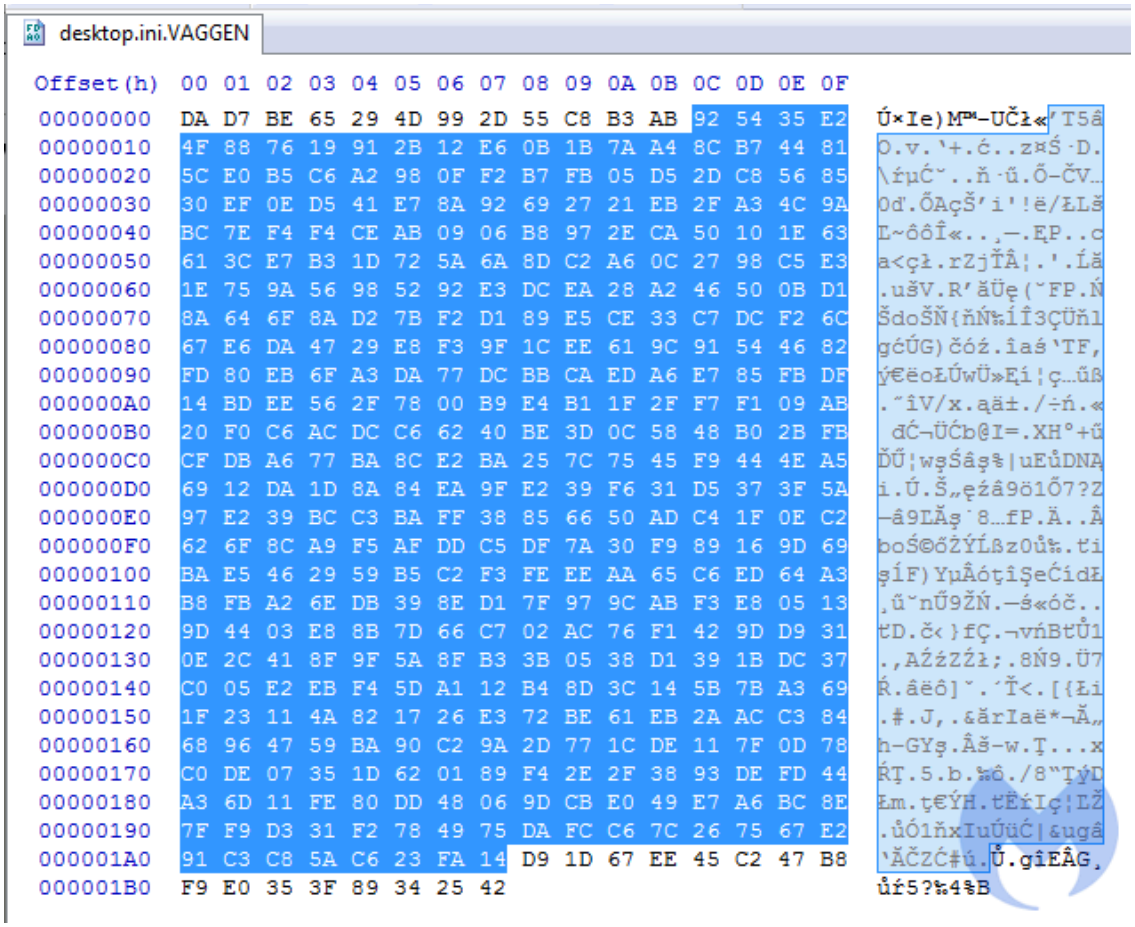
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

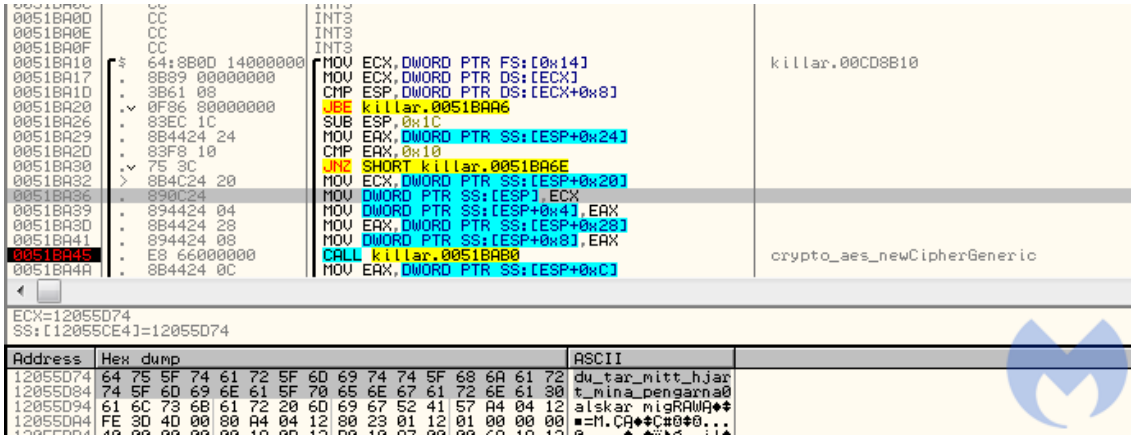


The content of the output file (with .VAGGEN extension) contains:

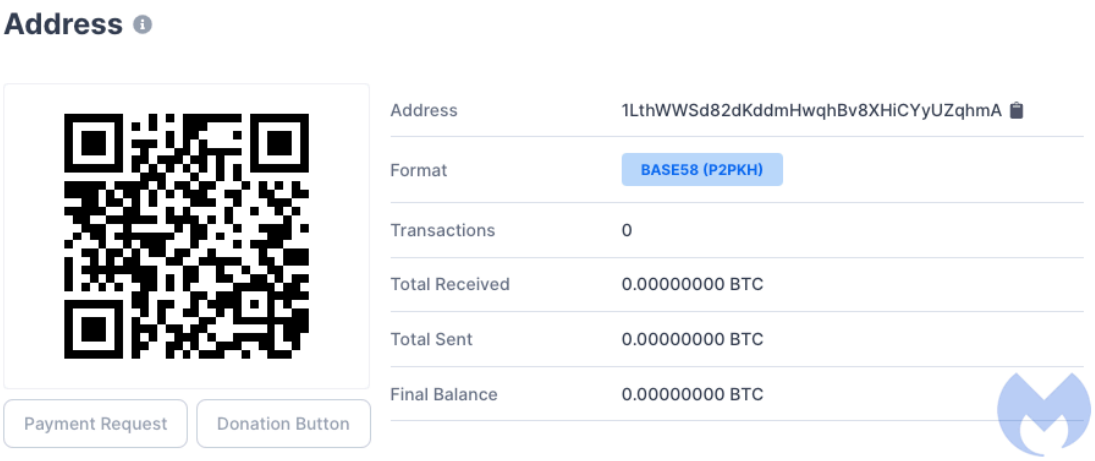
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

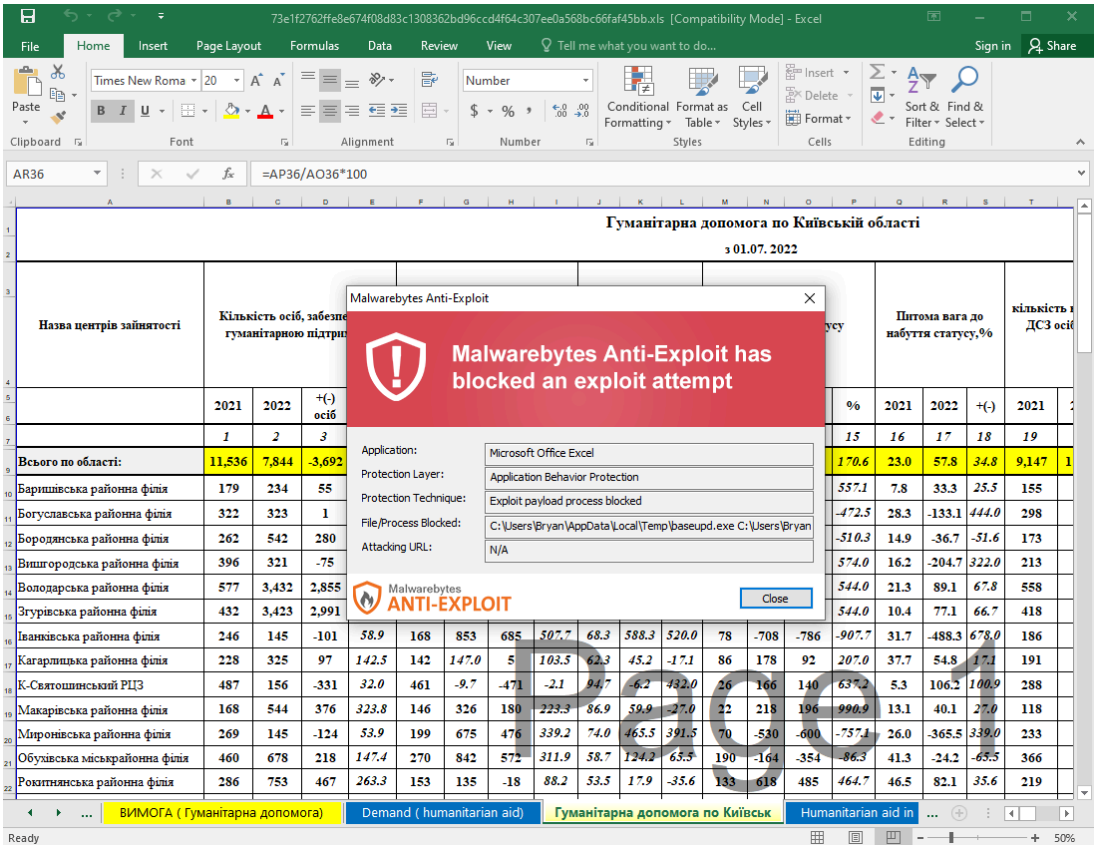
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

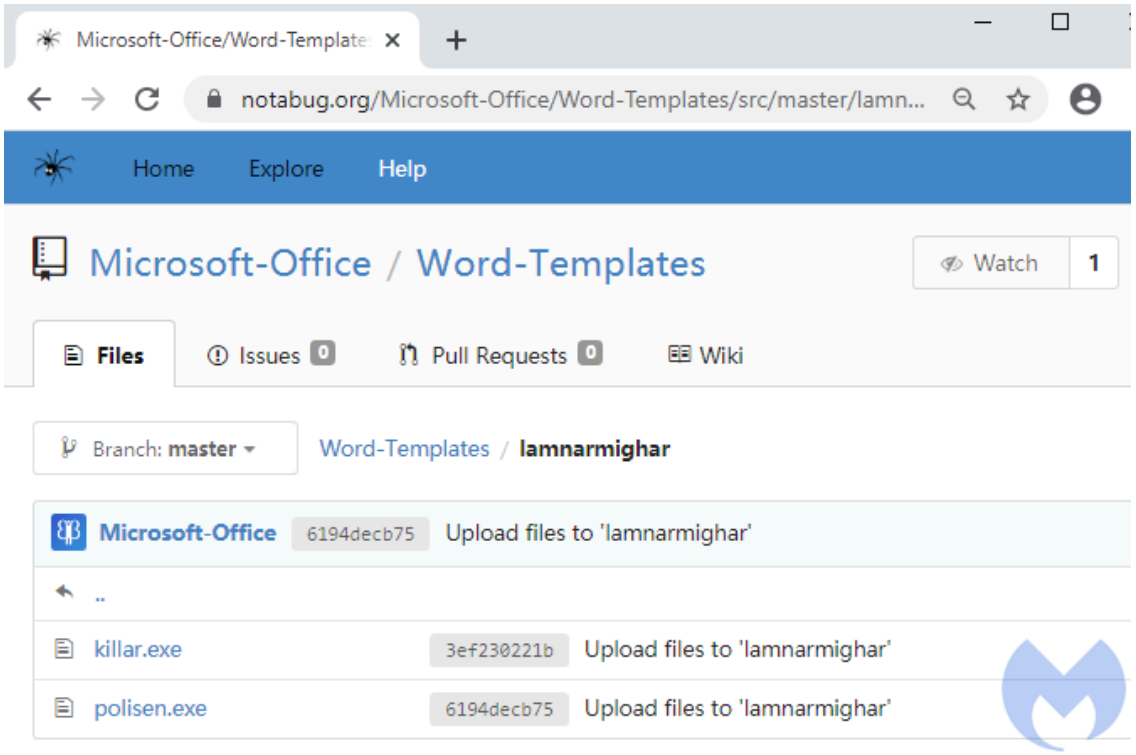
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



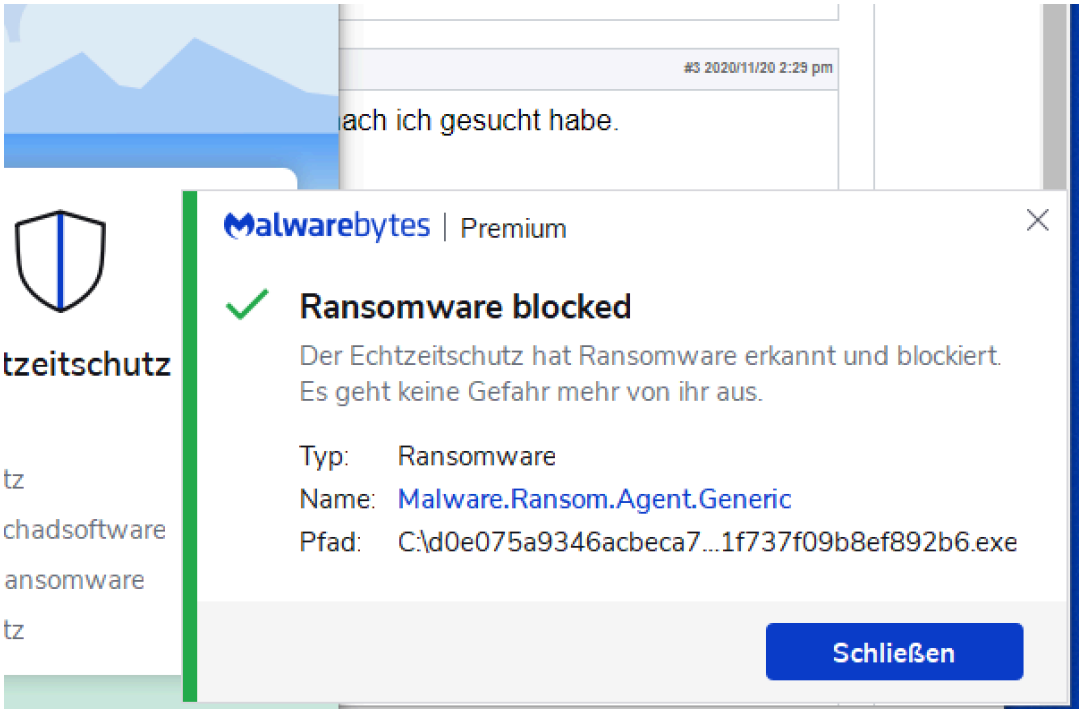
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

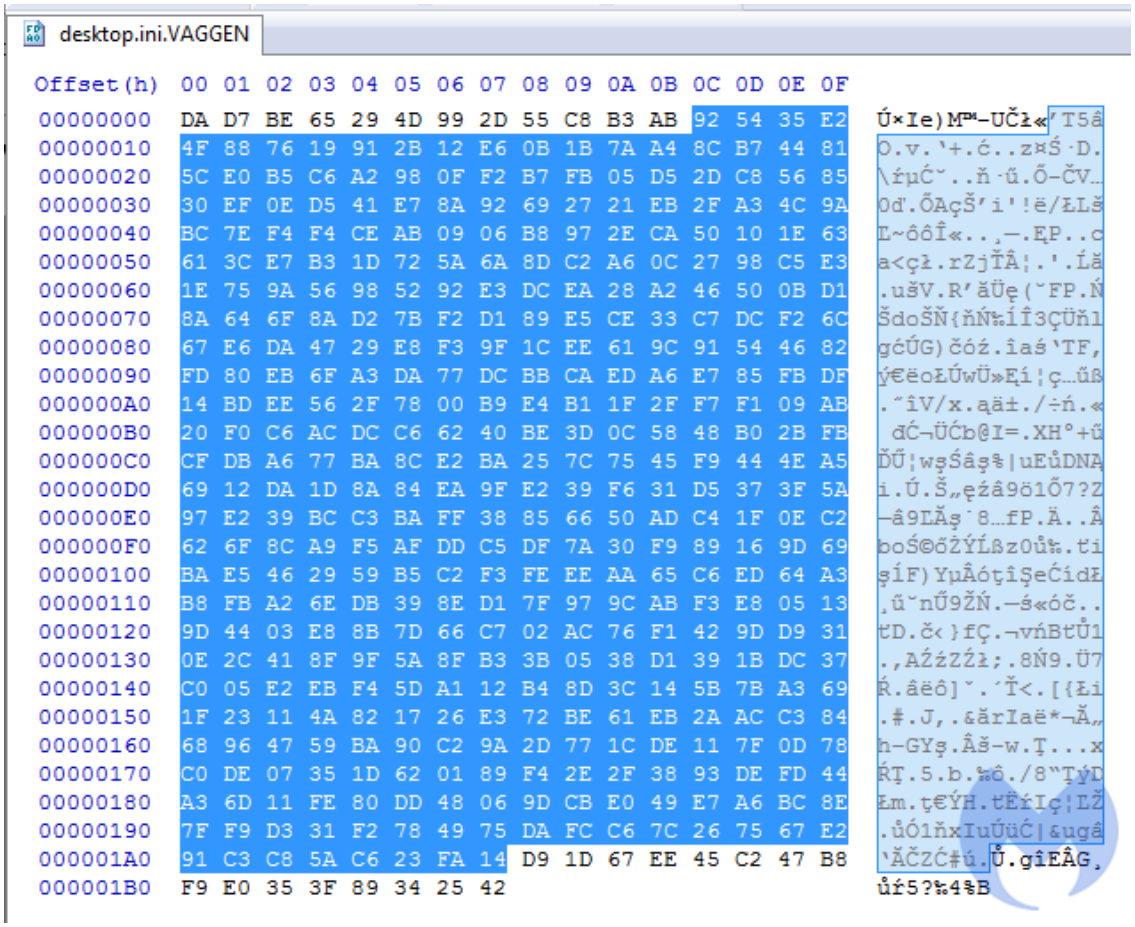
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

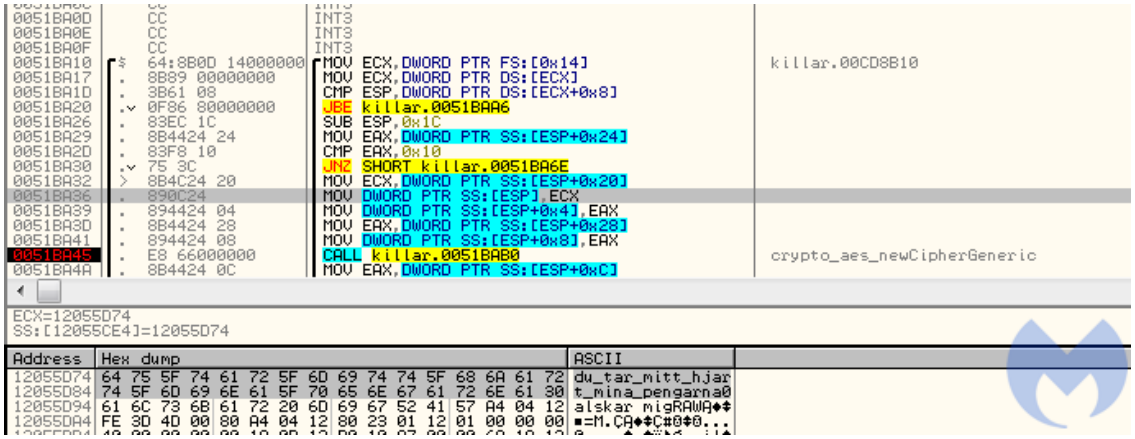


The content of the output file (with .VAGGEN extension) contains:

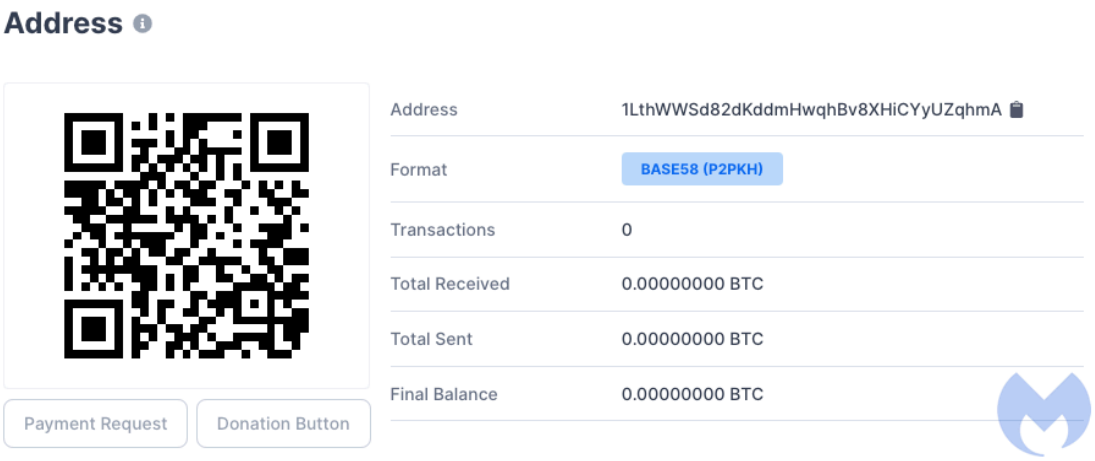
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

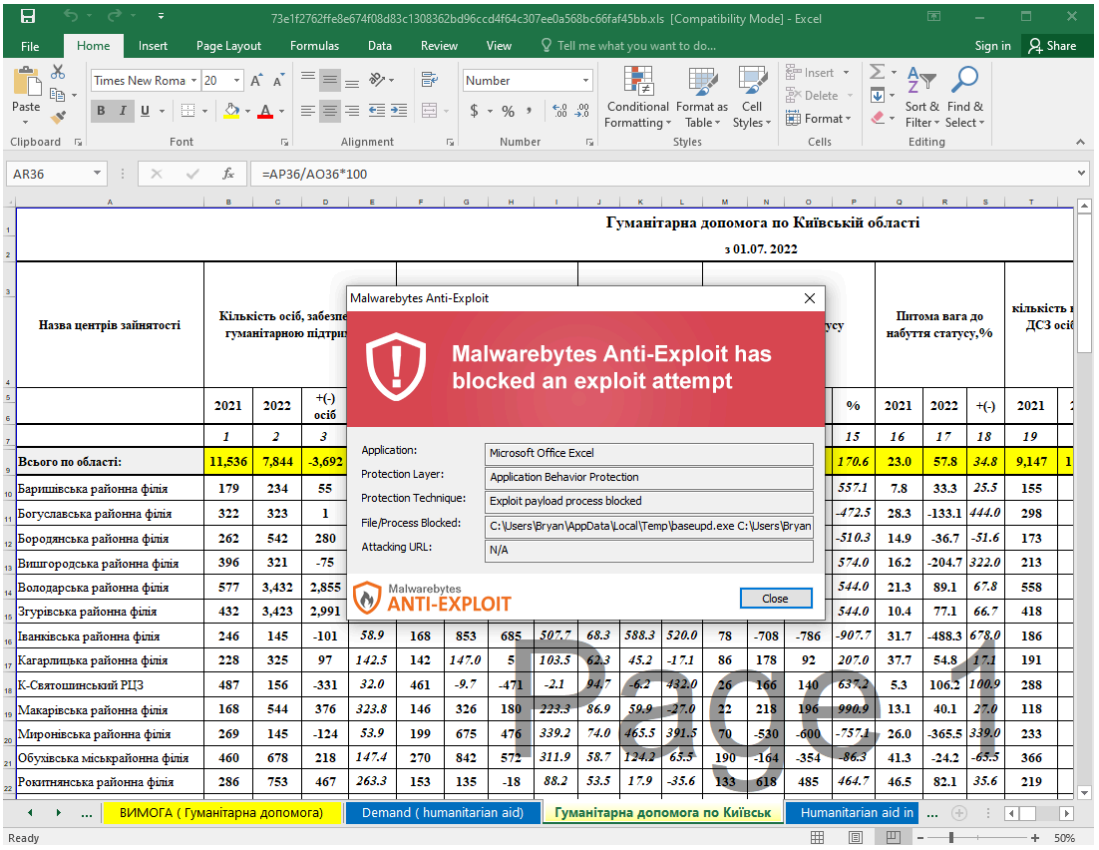
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

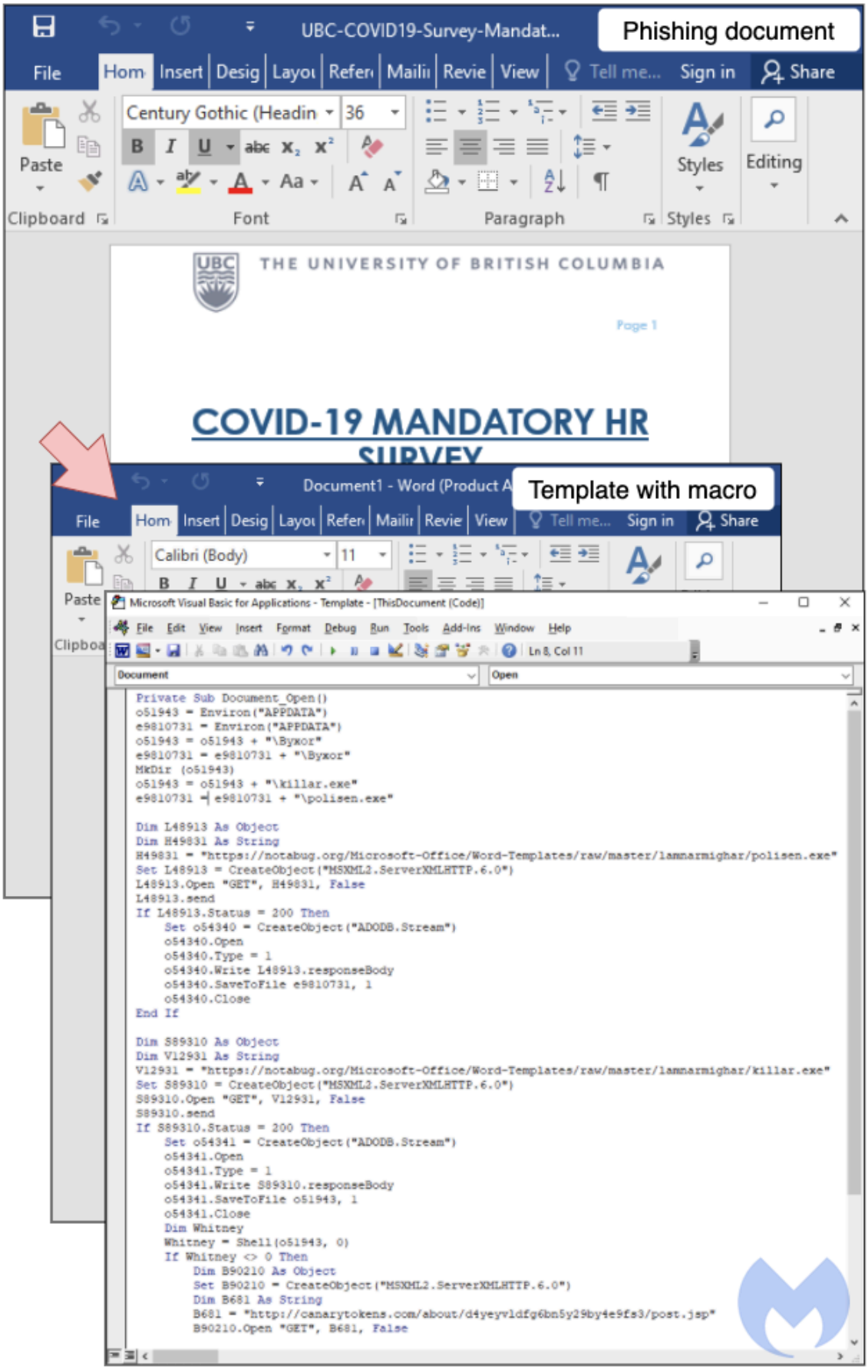
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

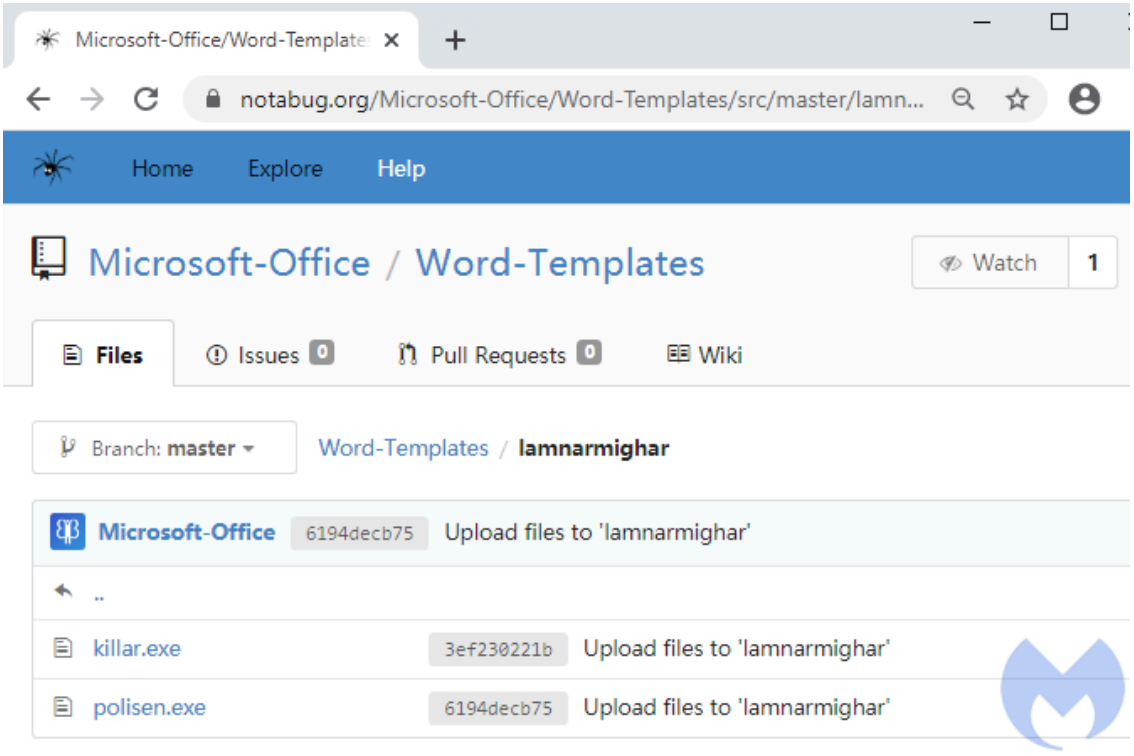
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



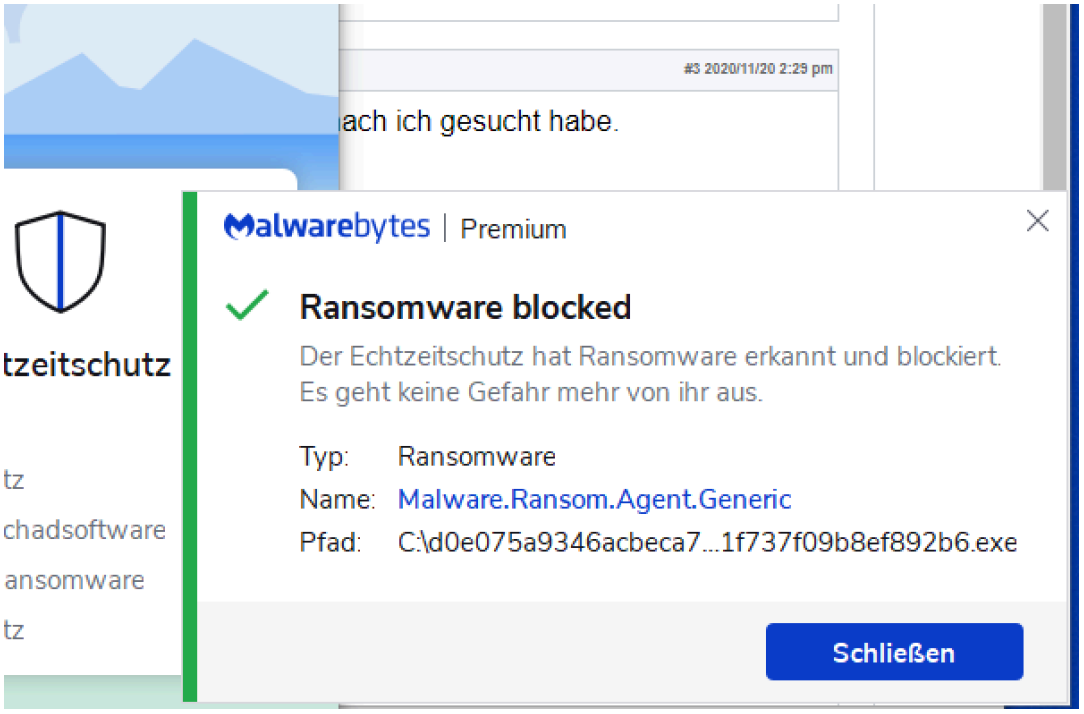
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

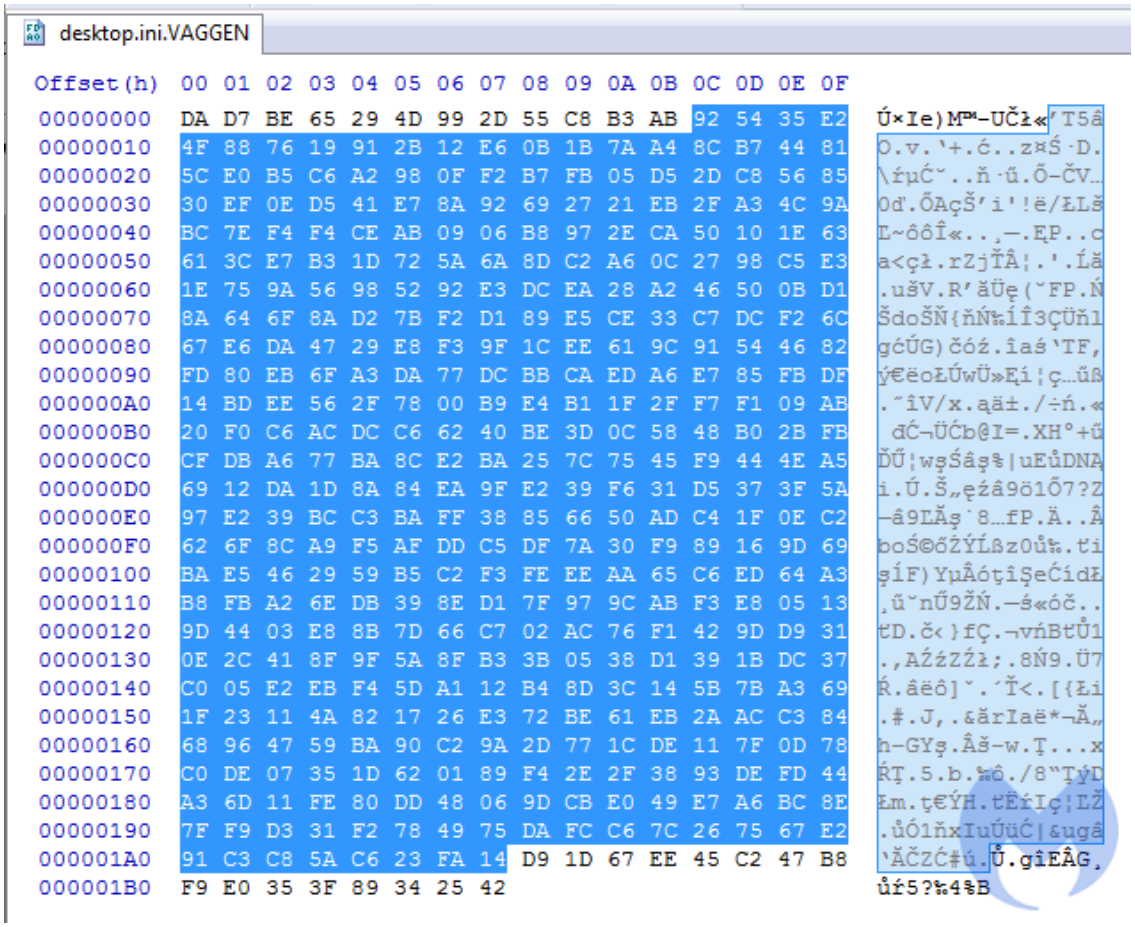
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

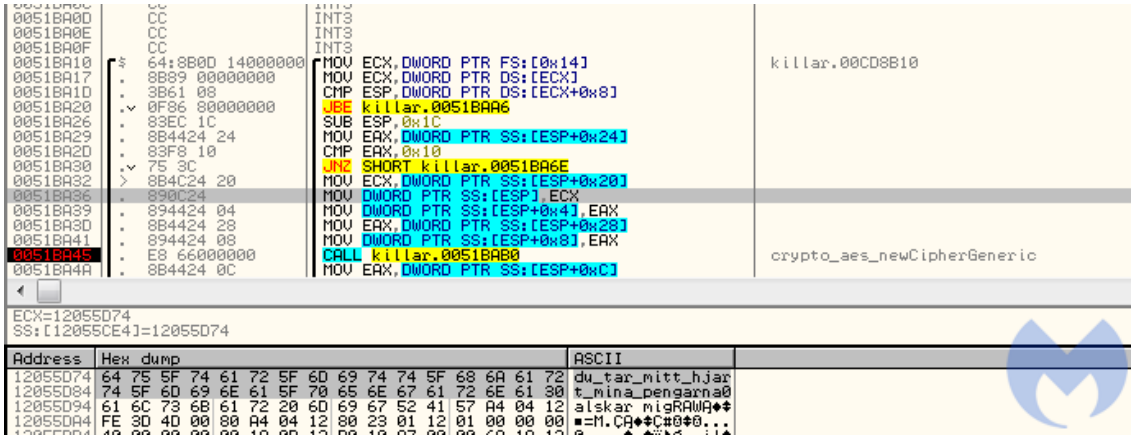


The content of the output file (with .VAGGEN extension) contains:

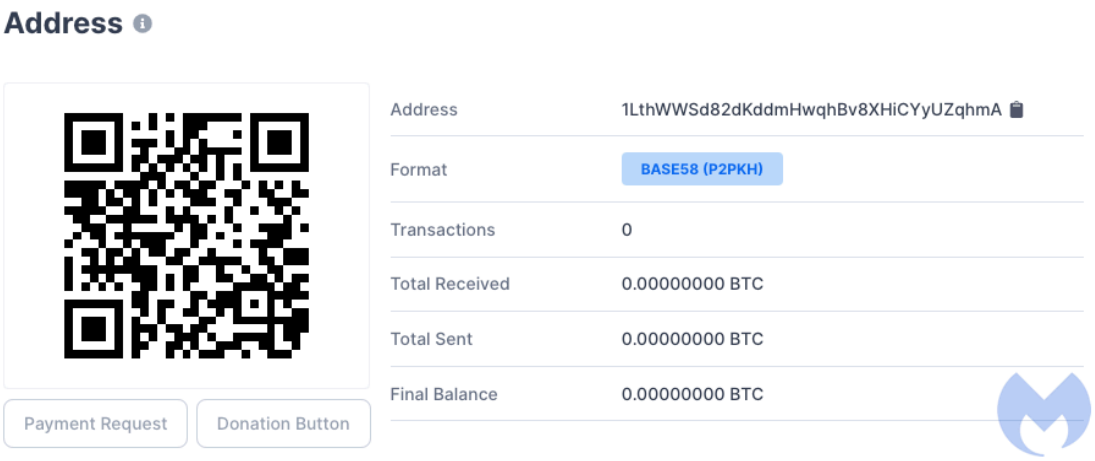
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

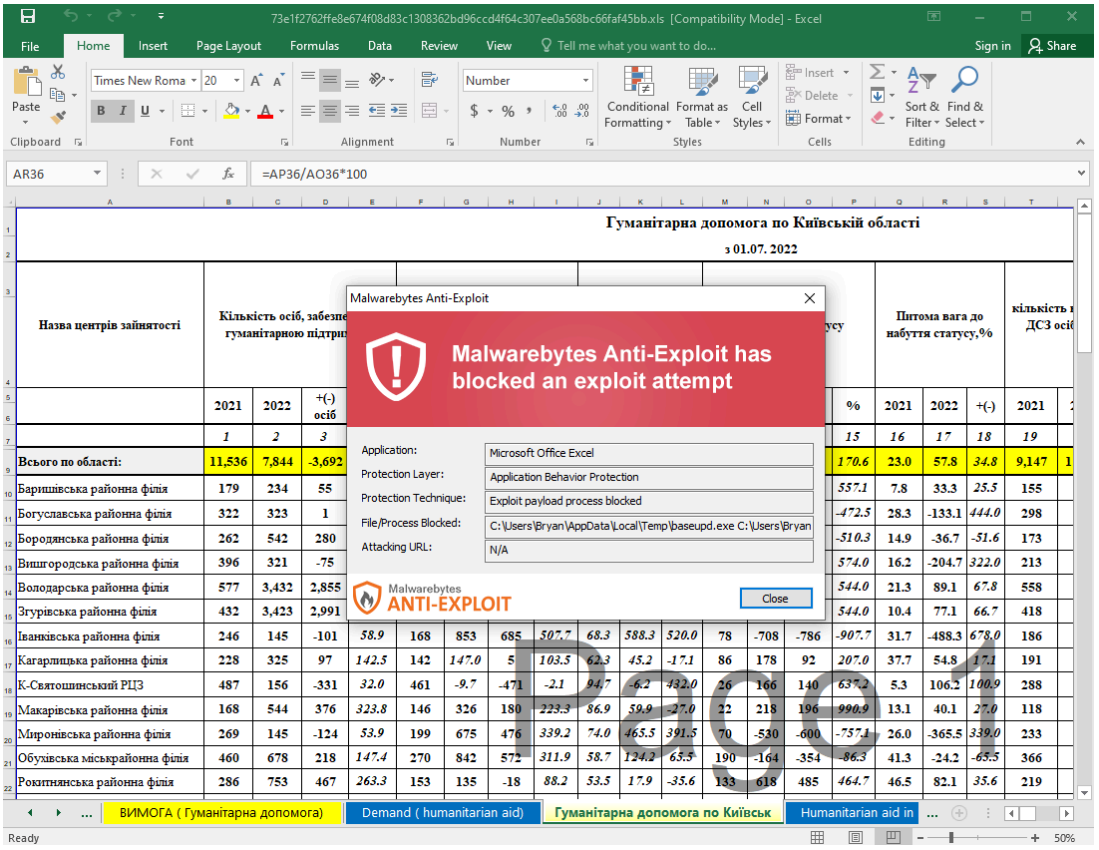
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

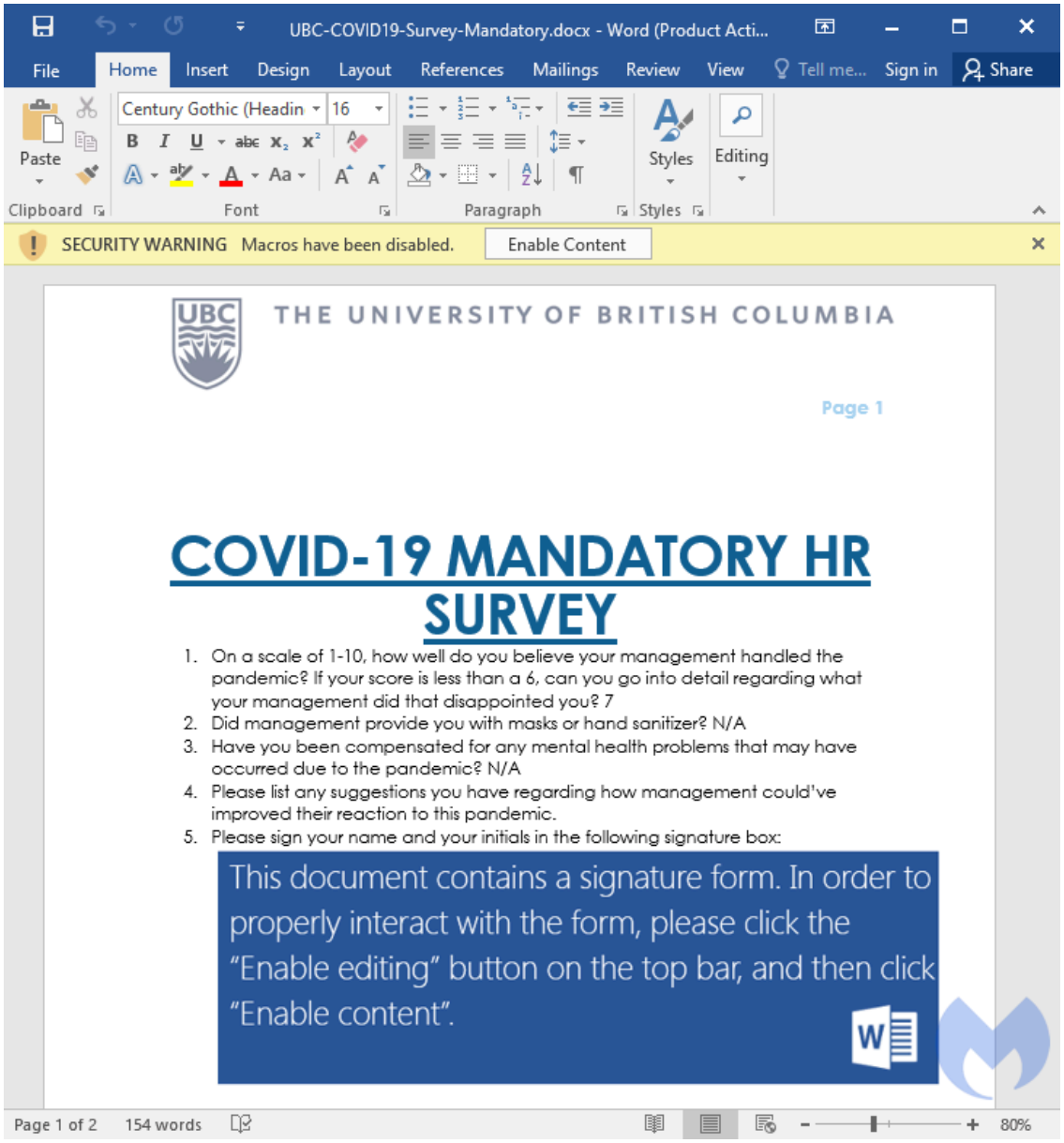
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

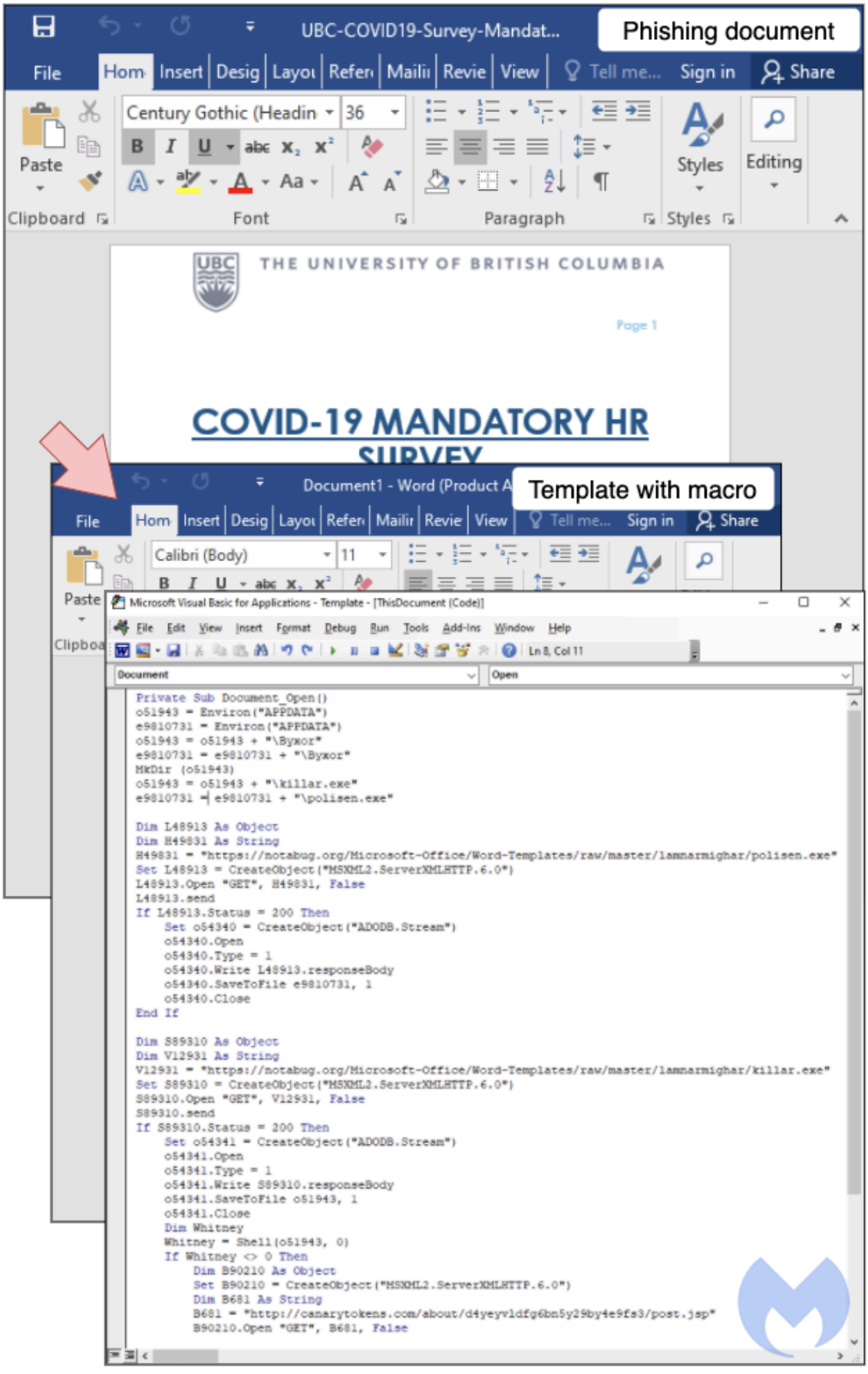
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

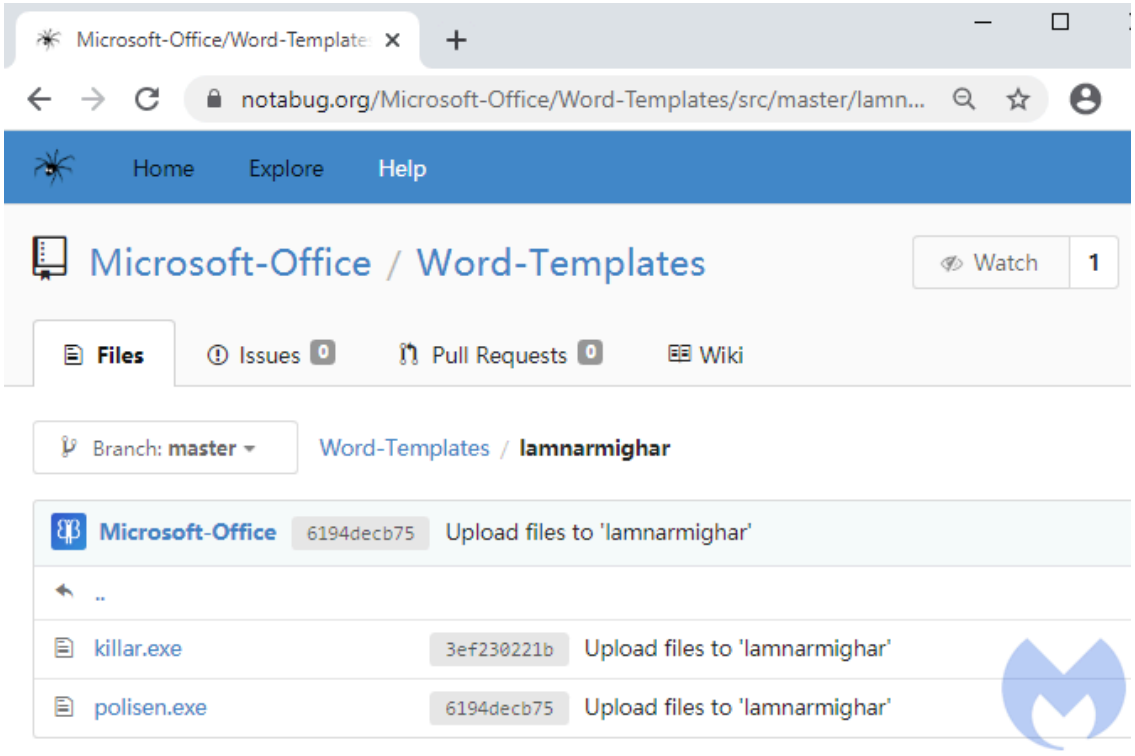
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



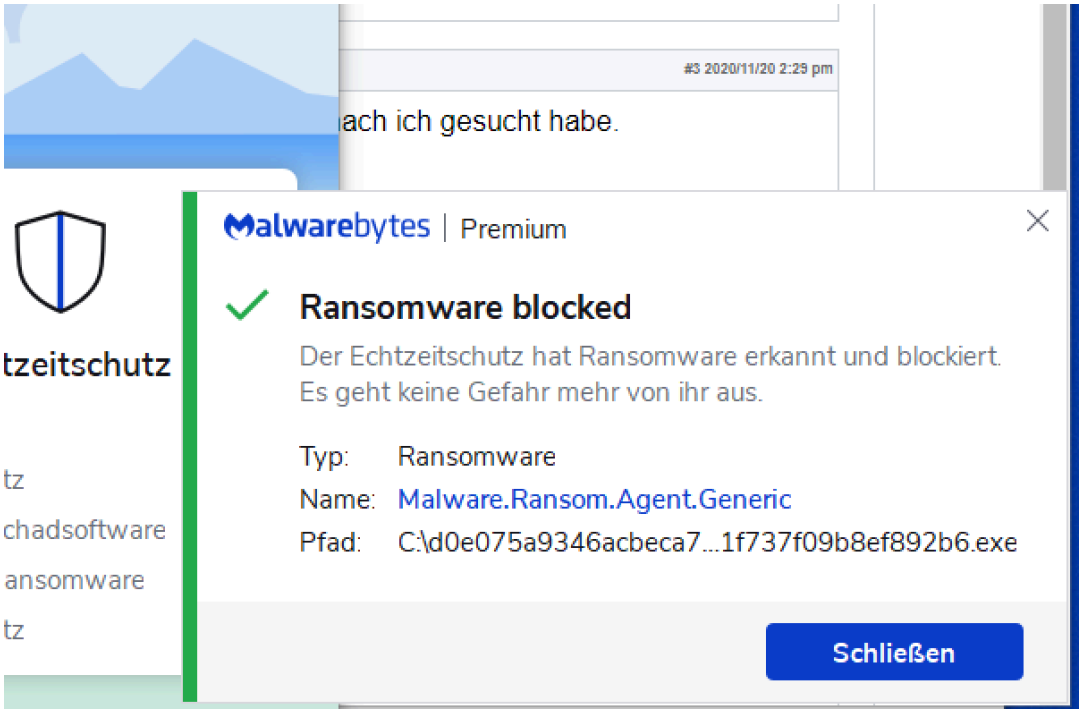
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

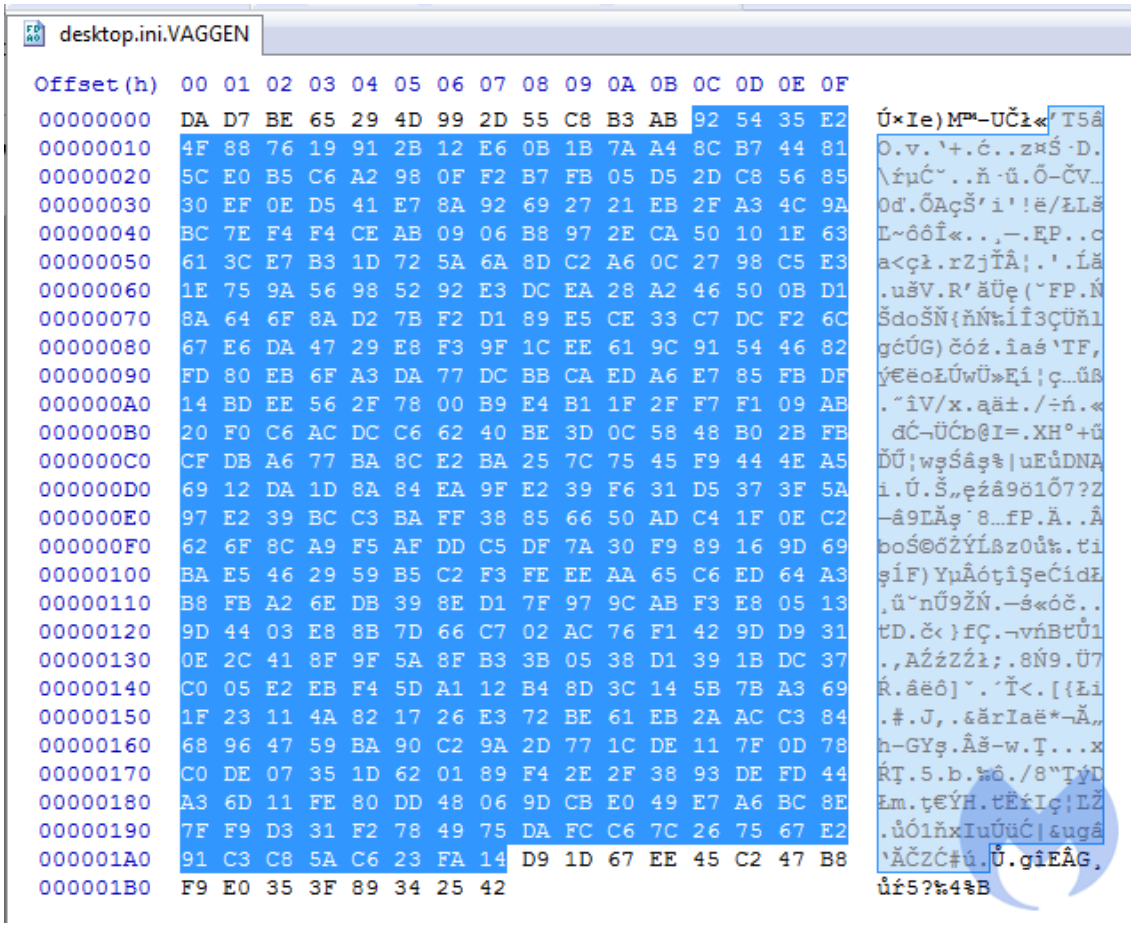
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

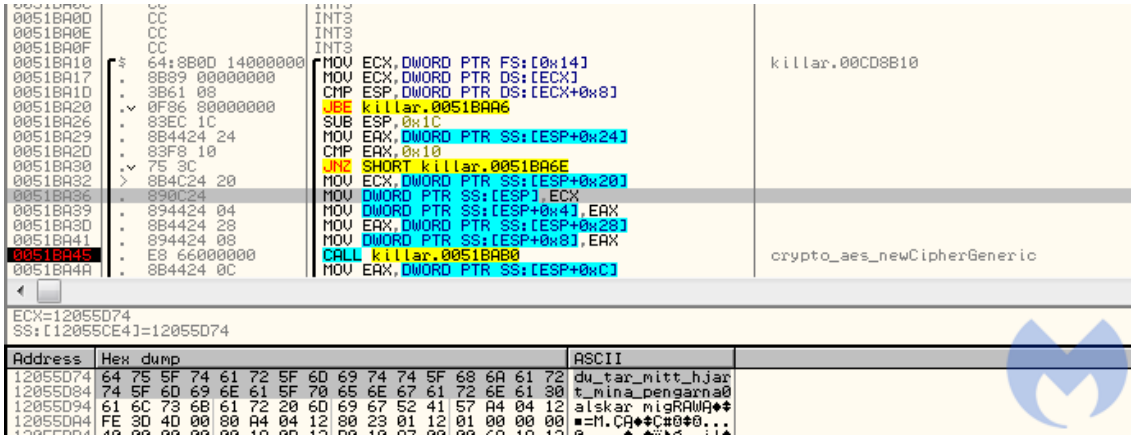


The content of the output file (with .VAGGEN extension) contains:

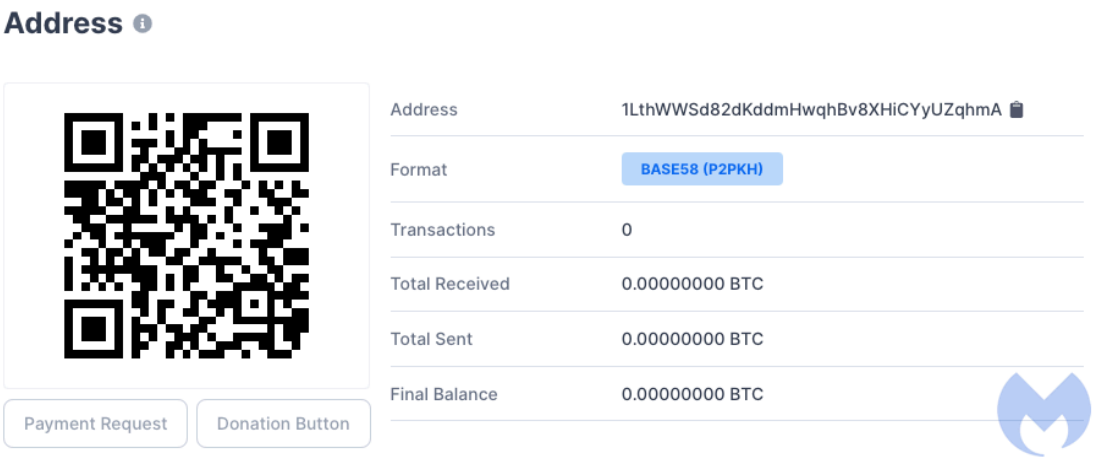
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

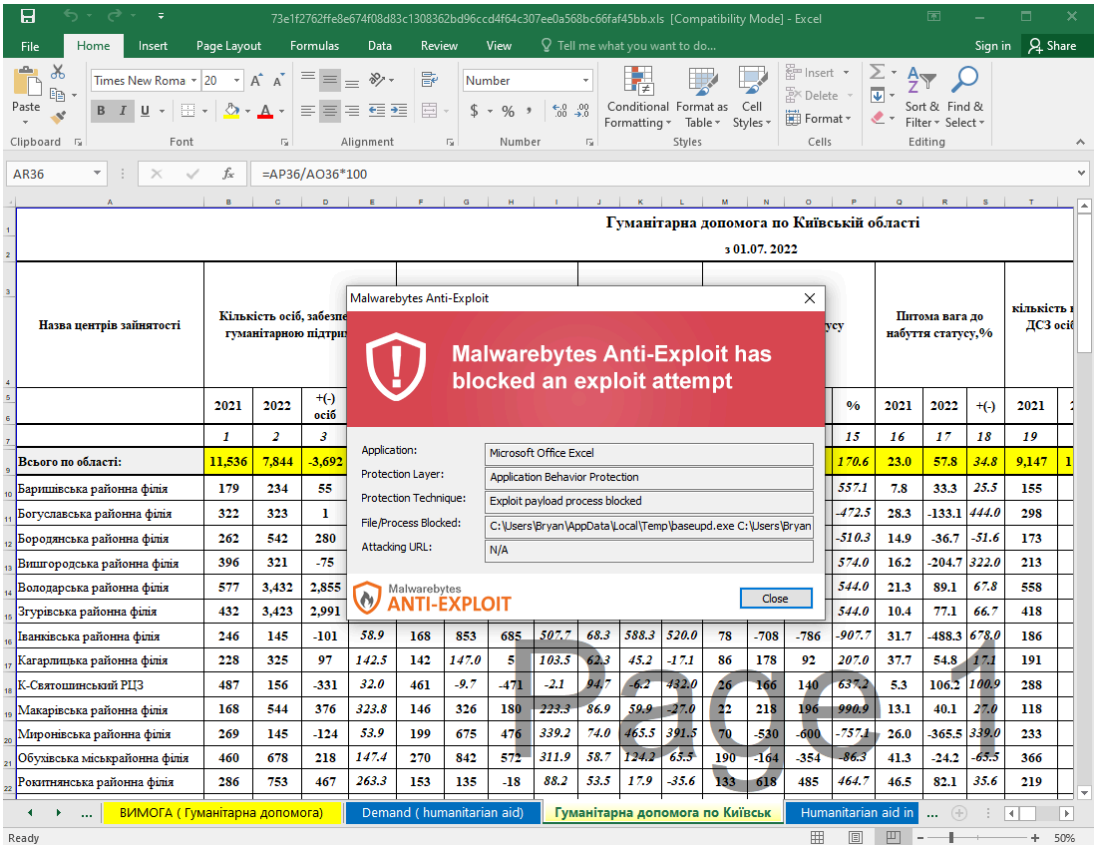
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

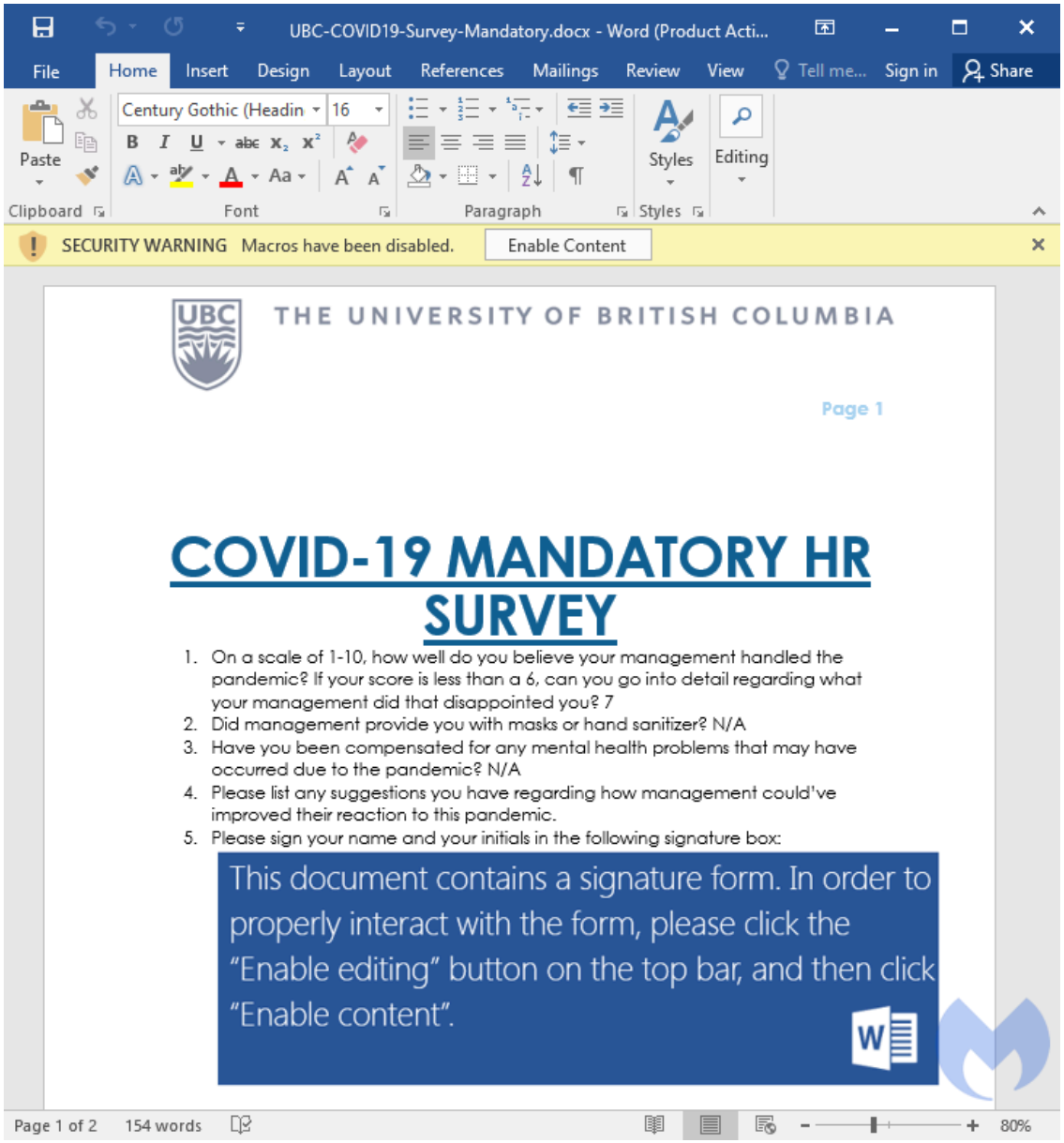
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

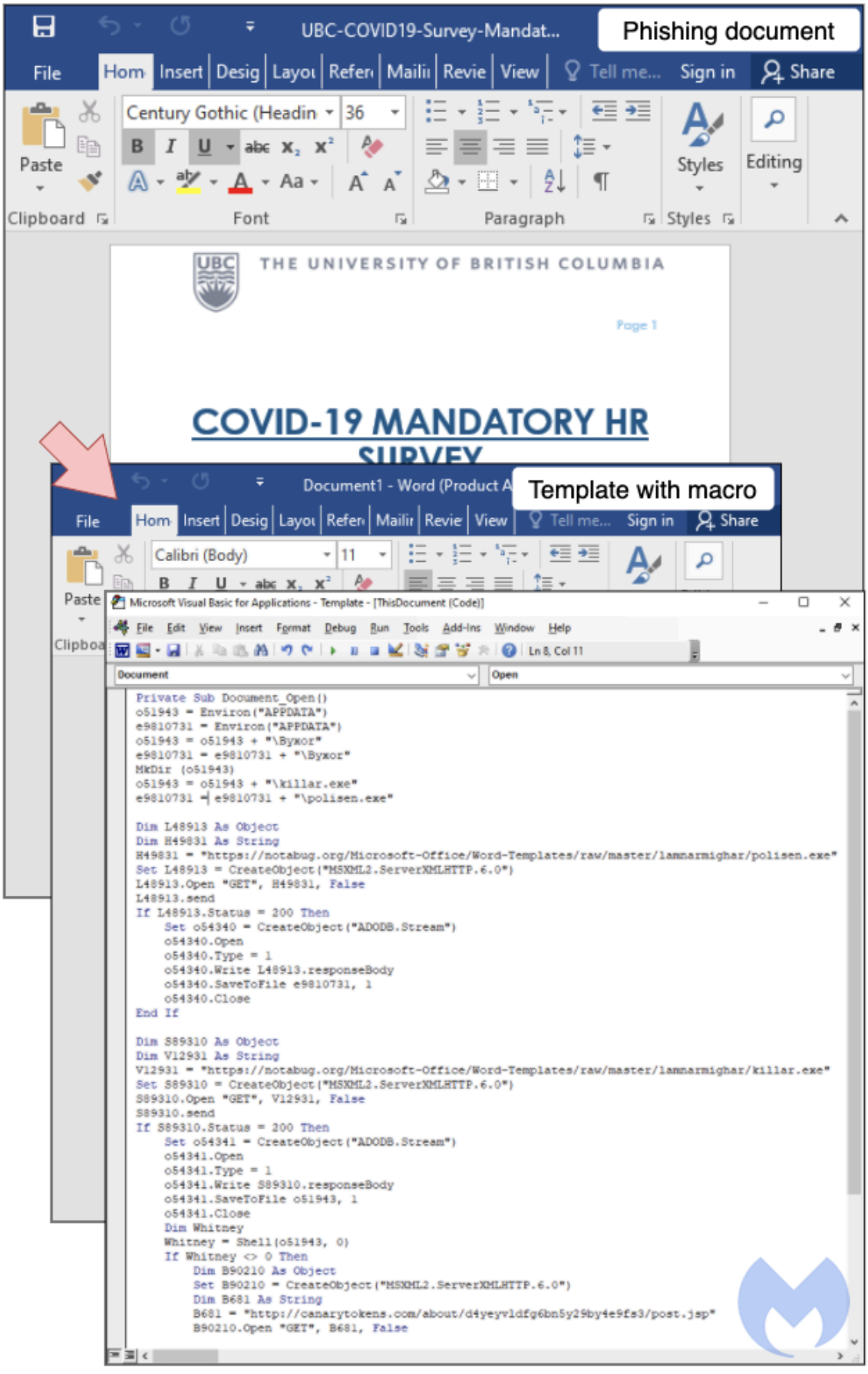
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

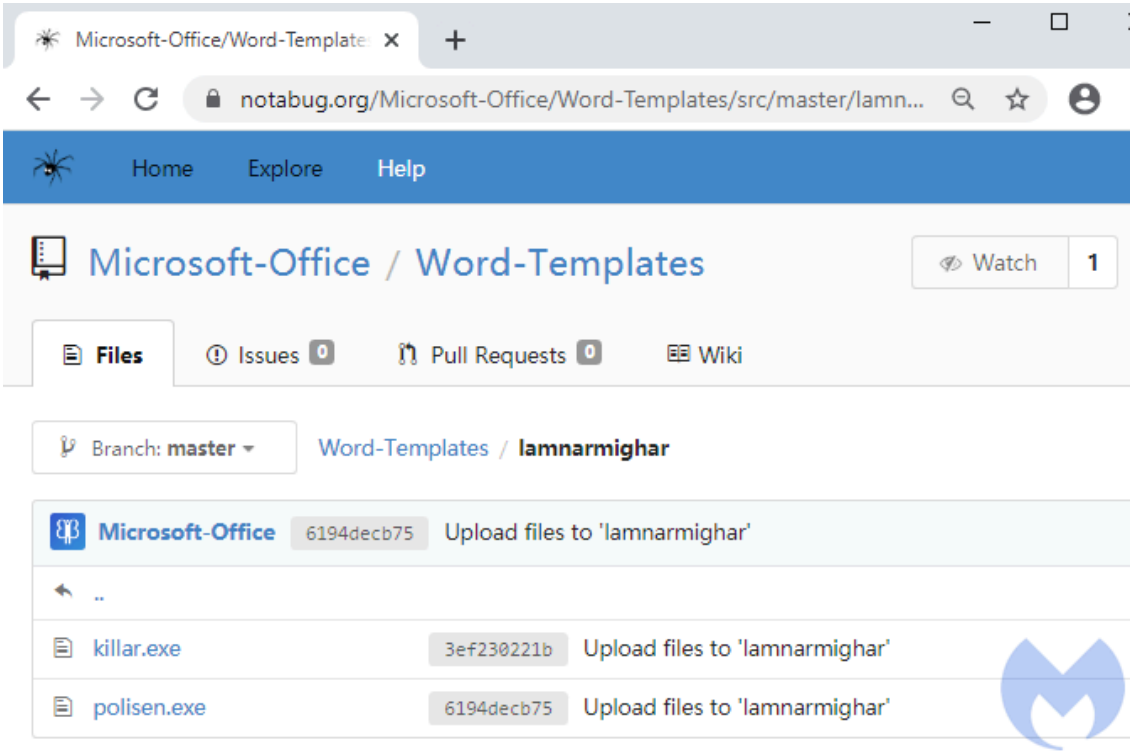
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



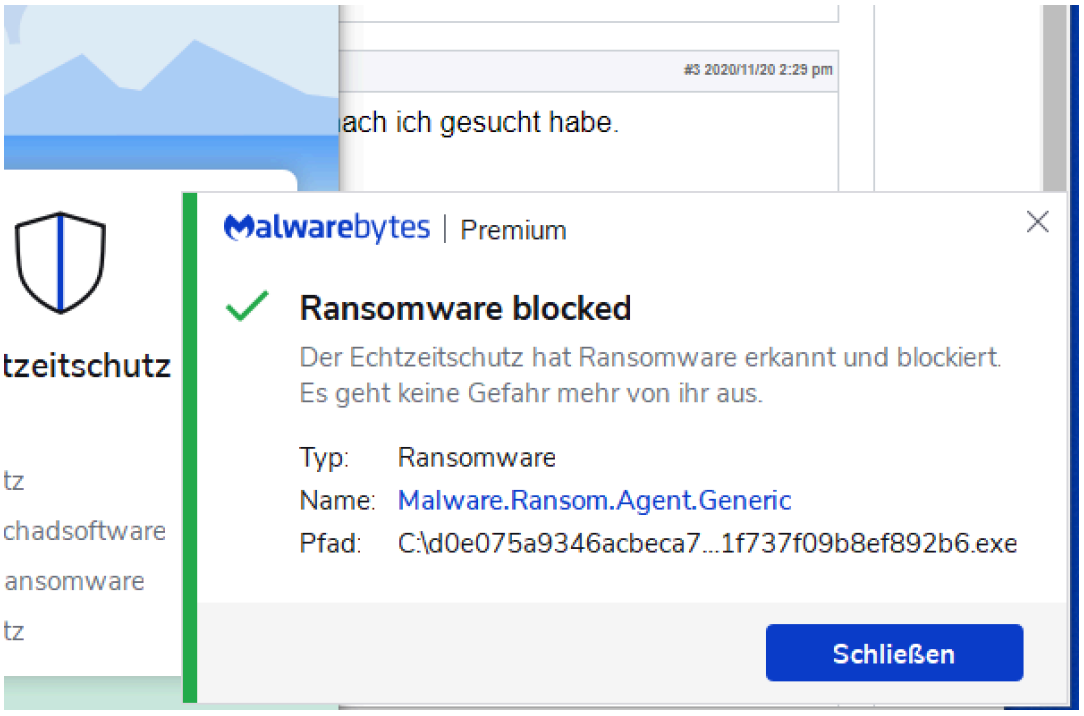
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

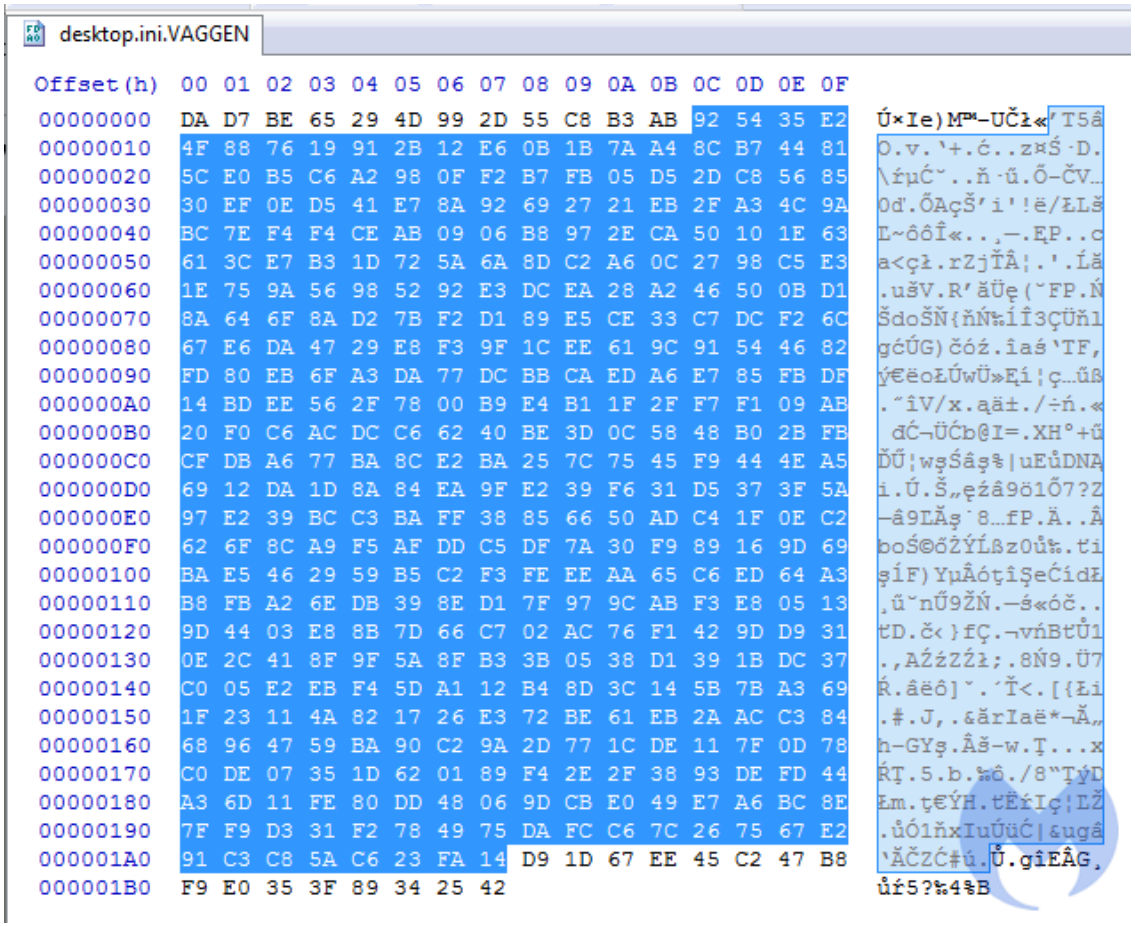
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

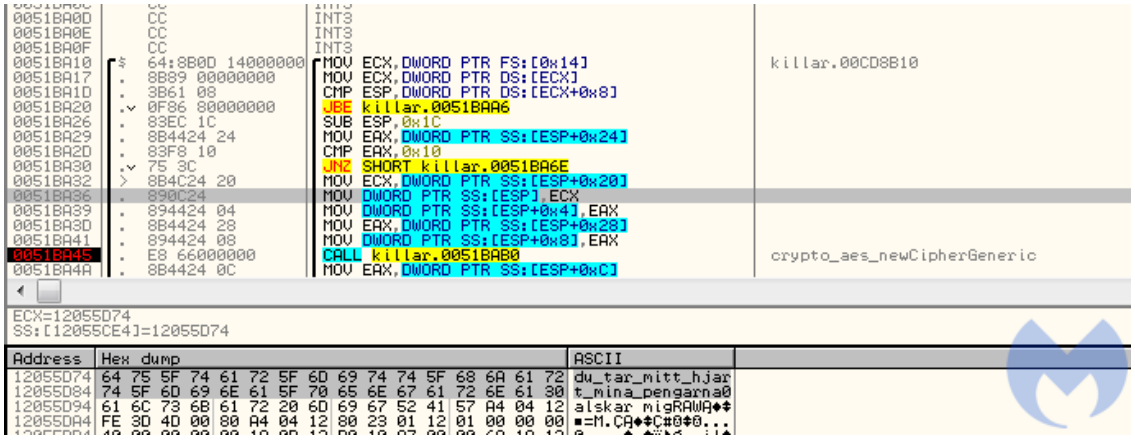


The content of the output file (with .VAGGEN extension) contains:

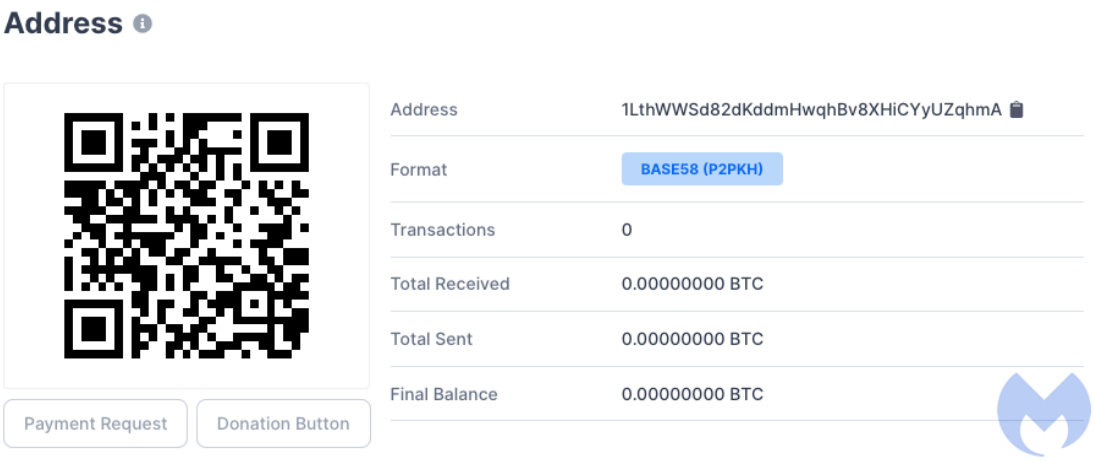
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

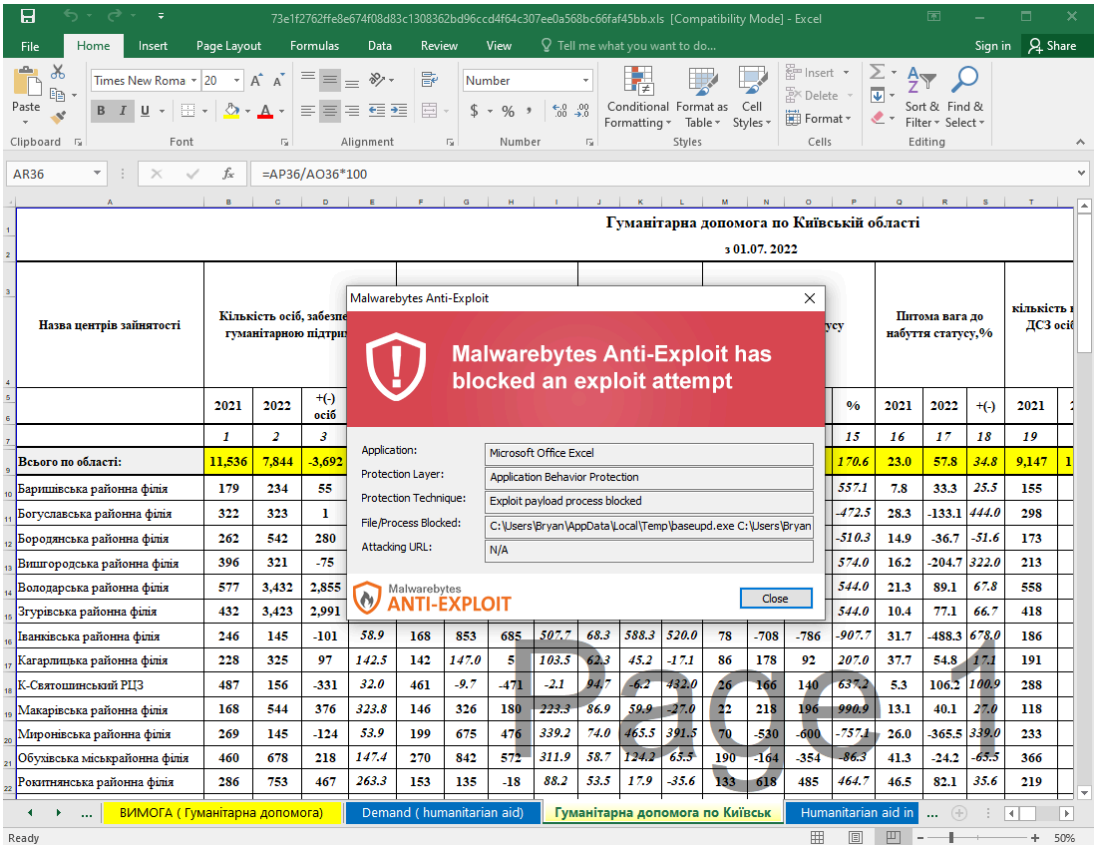
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

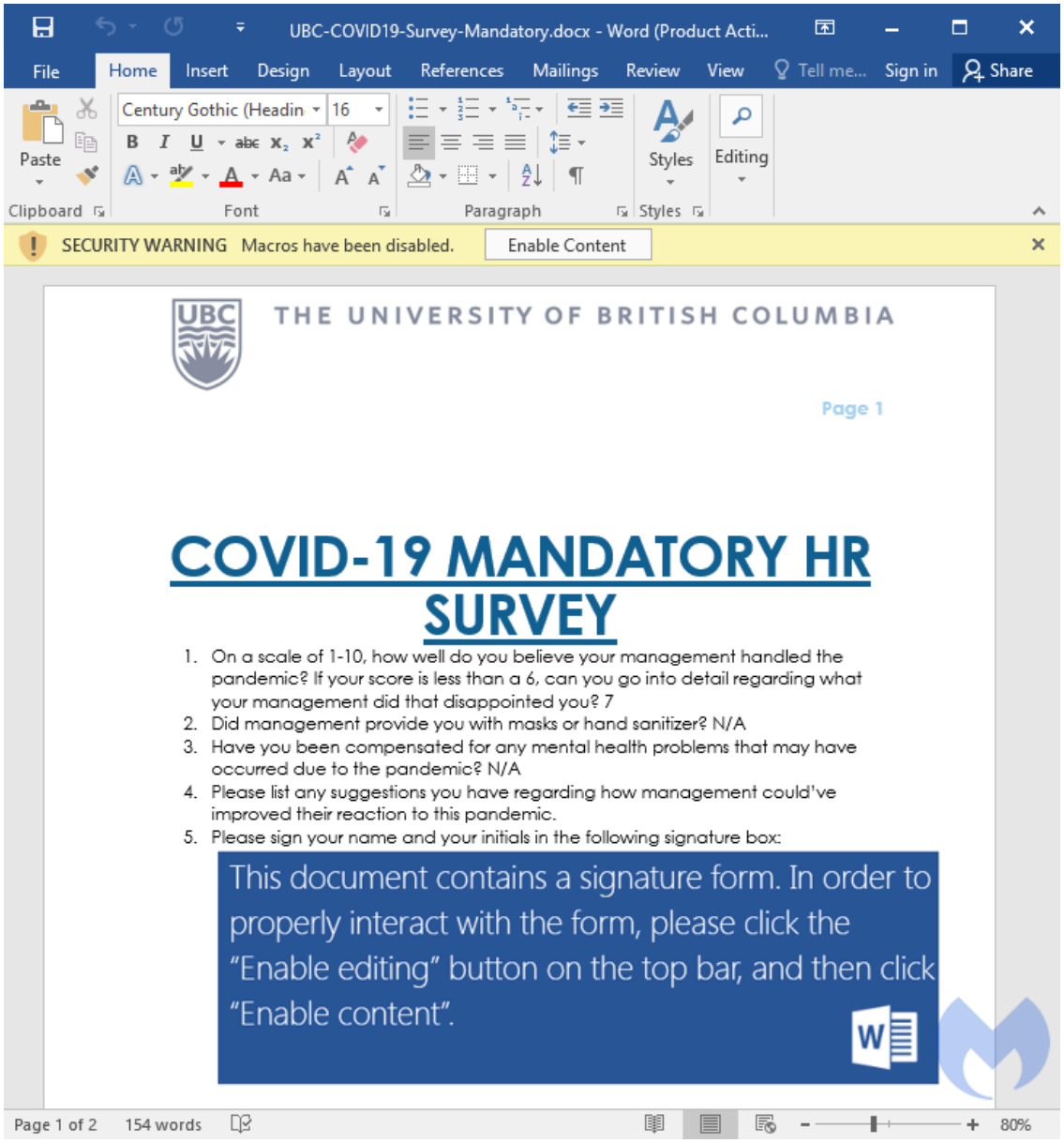
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

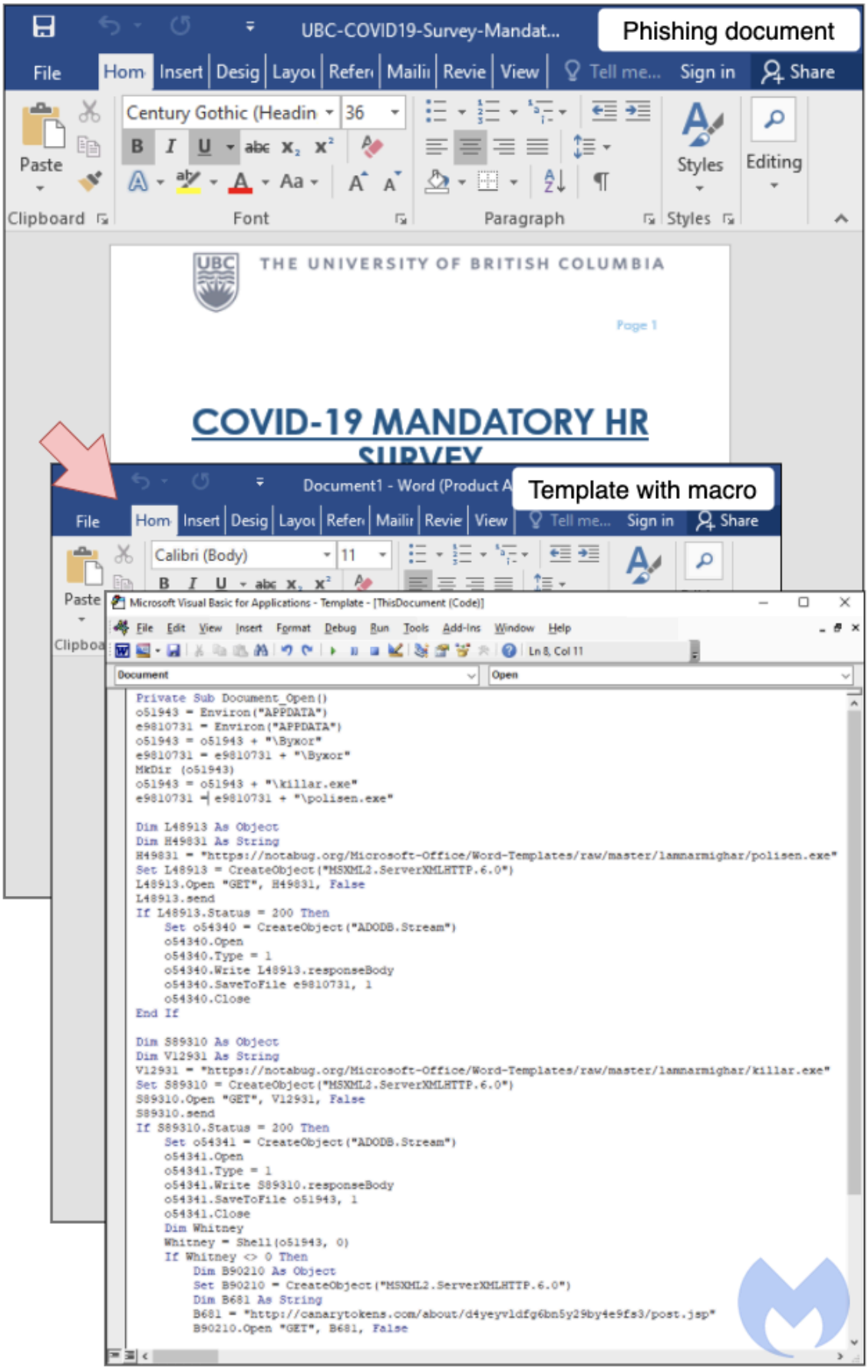
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

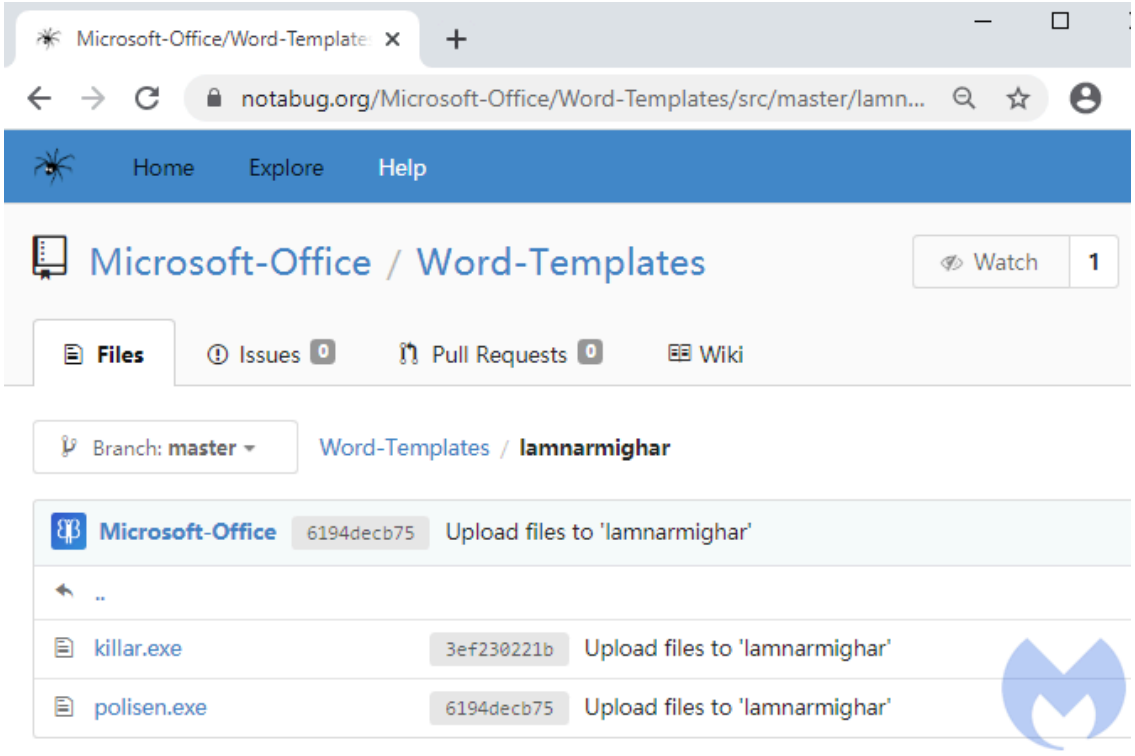
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



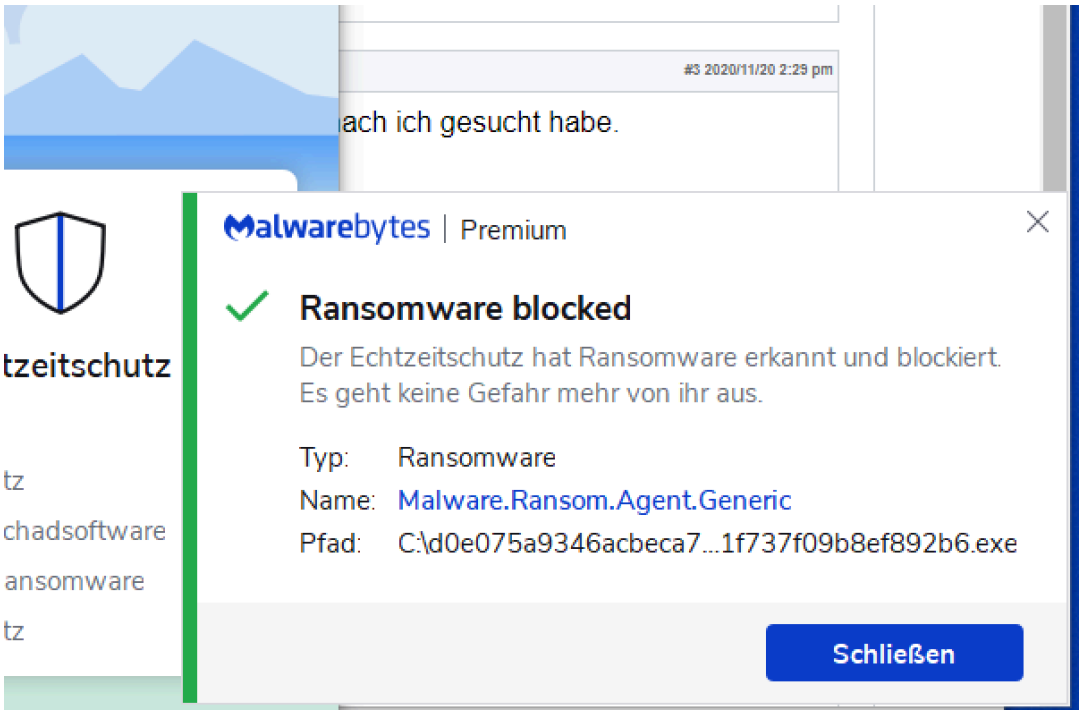
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

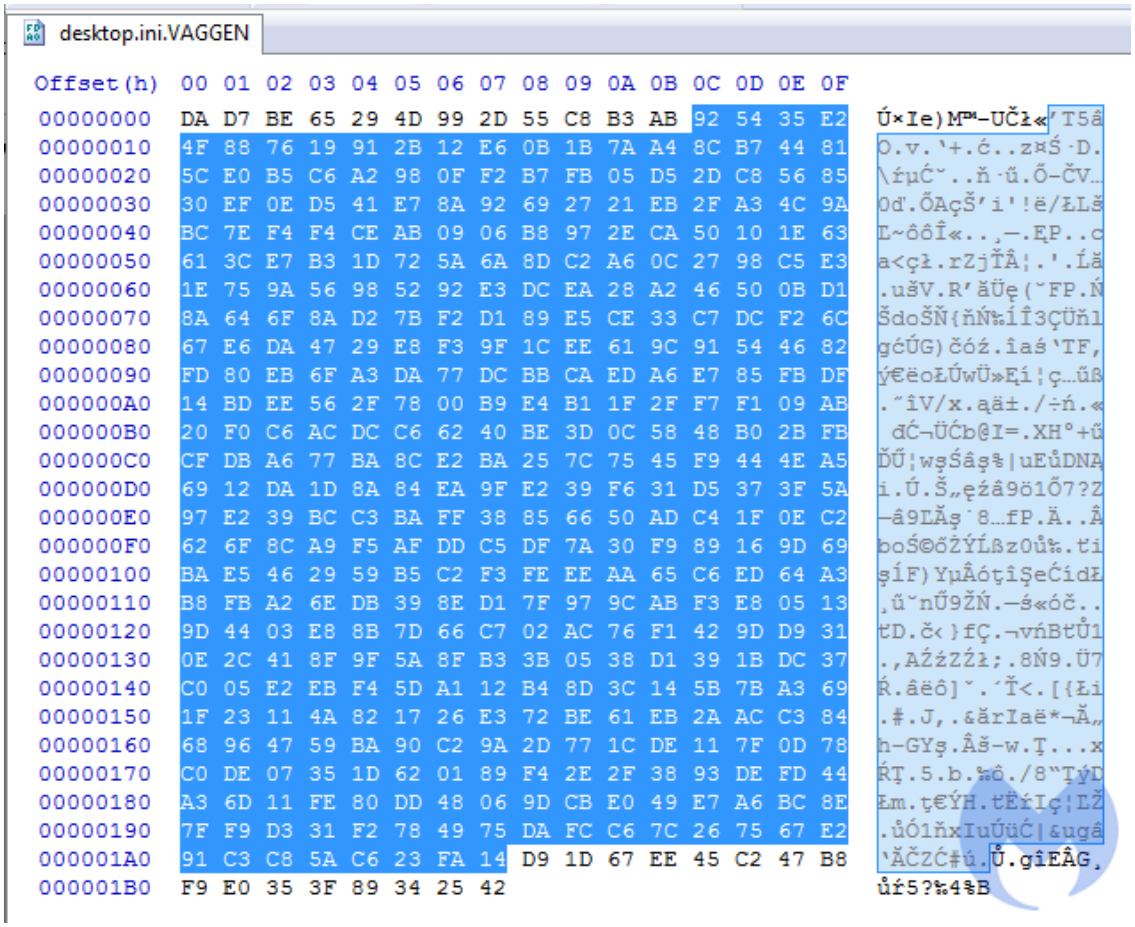
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

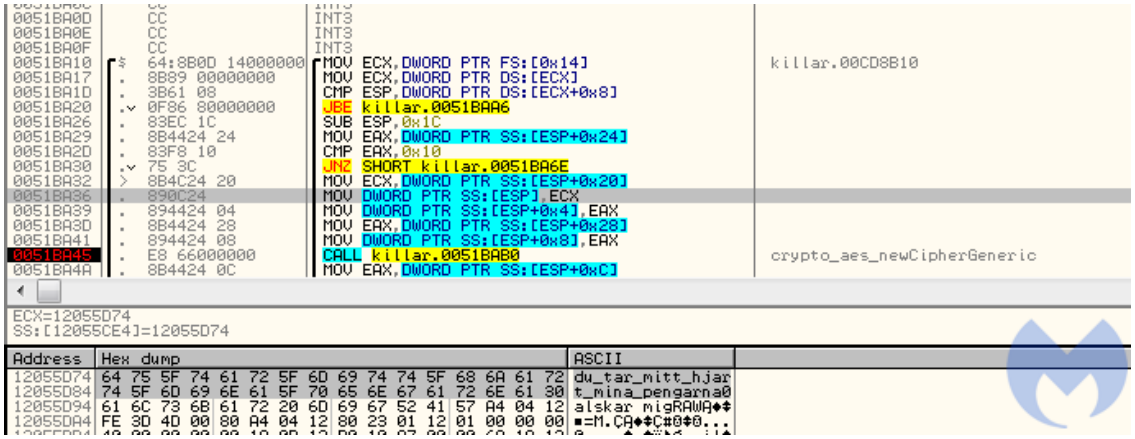


The content of the output file (with .VAGGEN extension) contains:

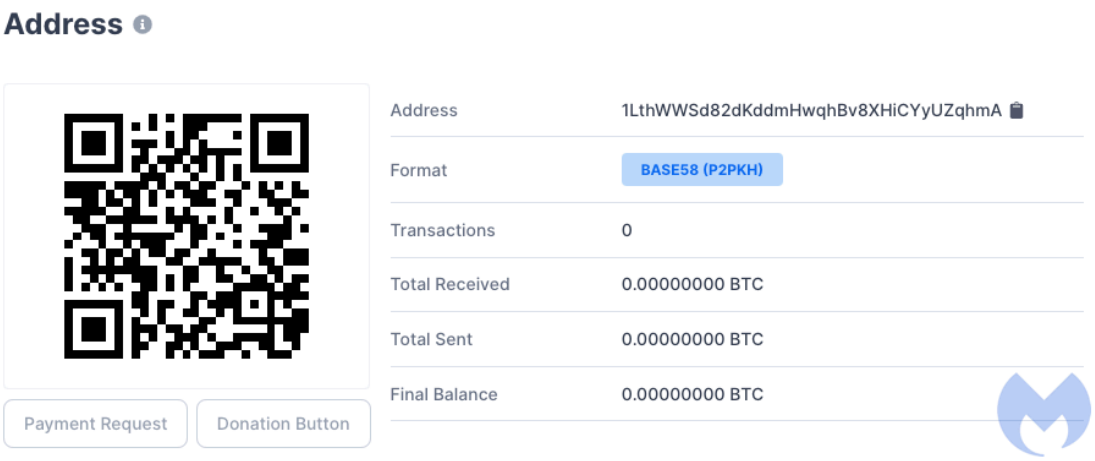
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

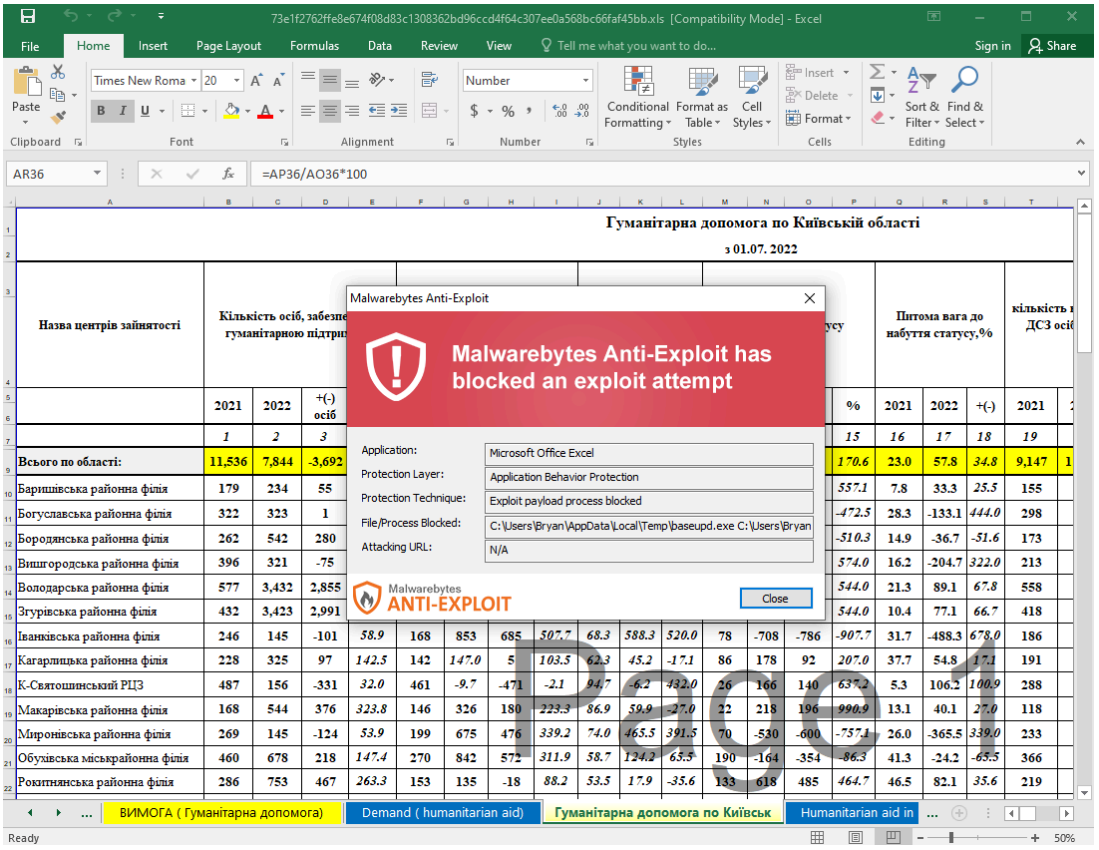
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

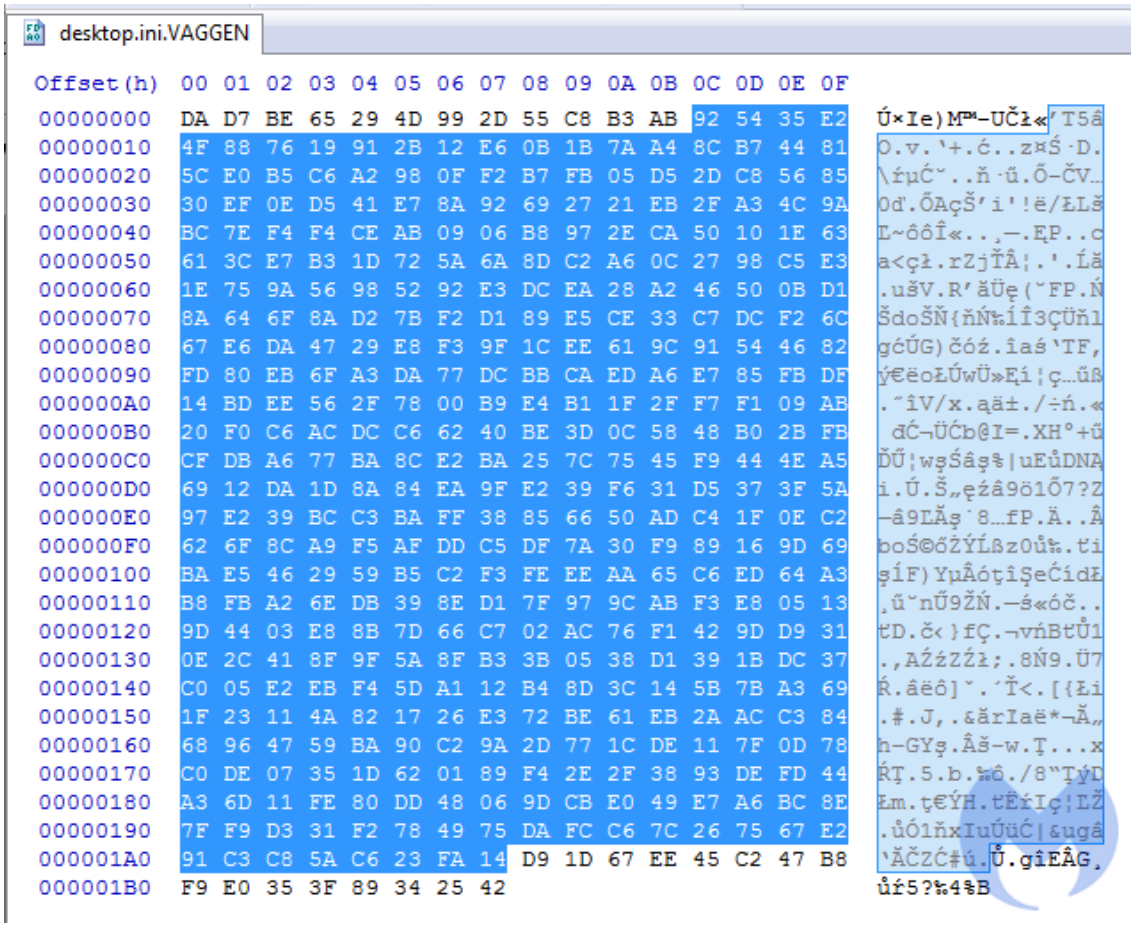
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



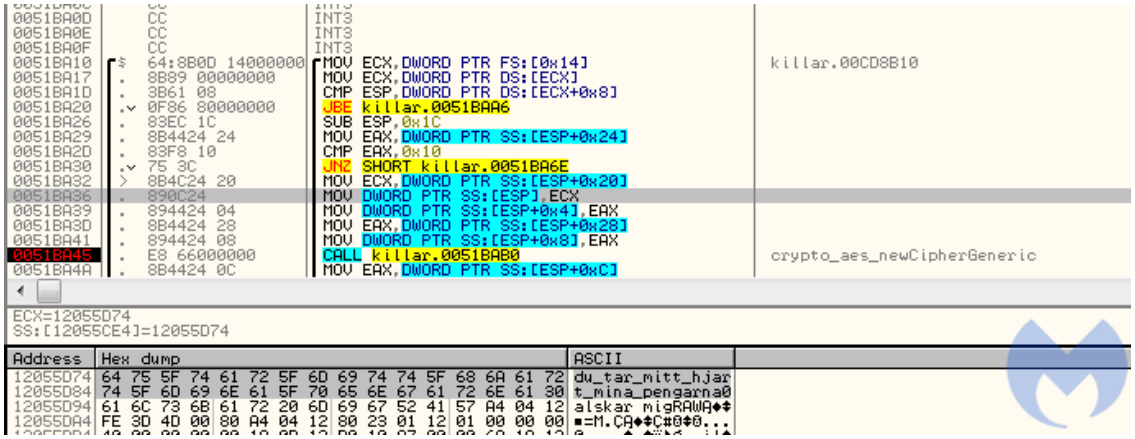
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

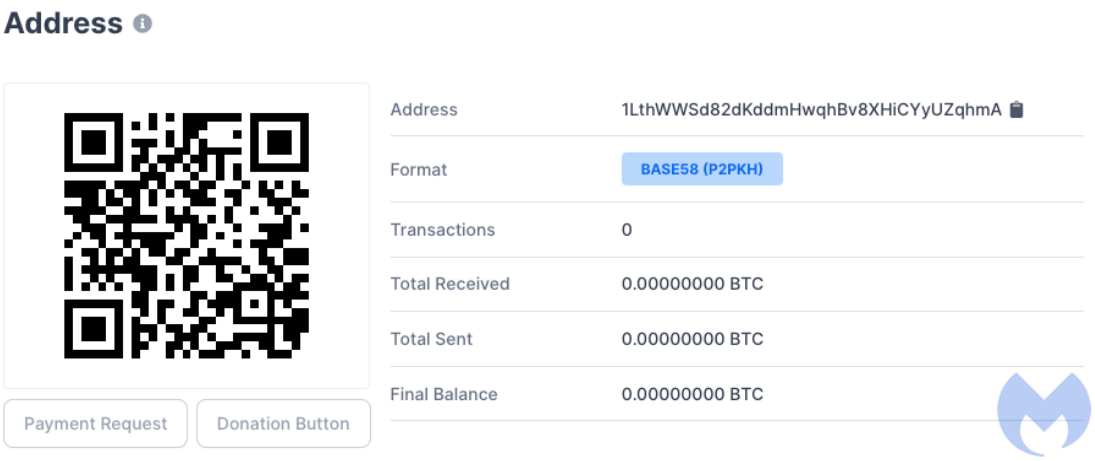
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

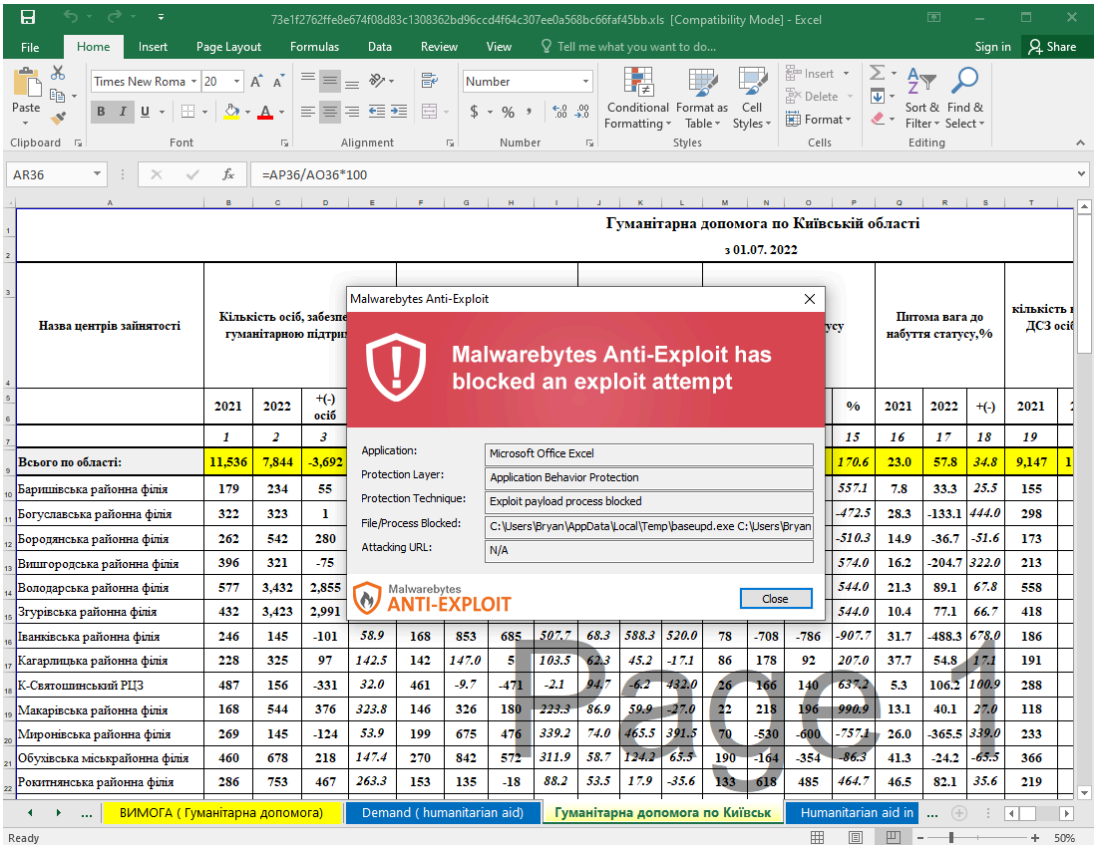
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

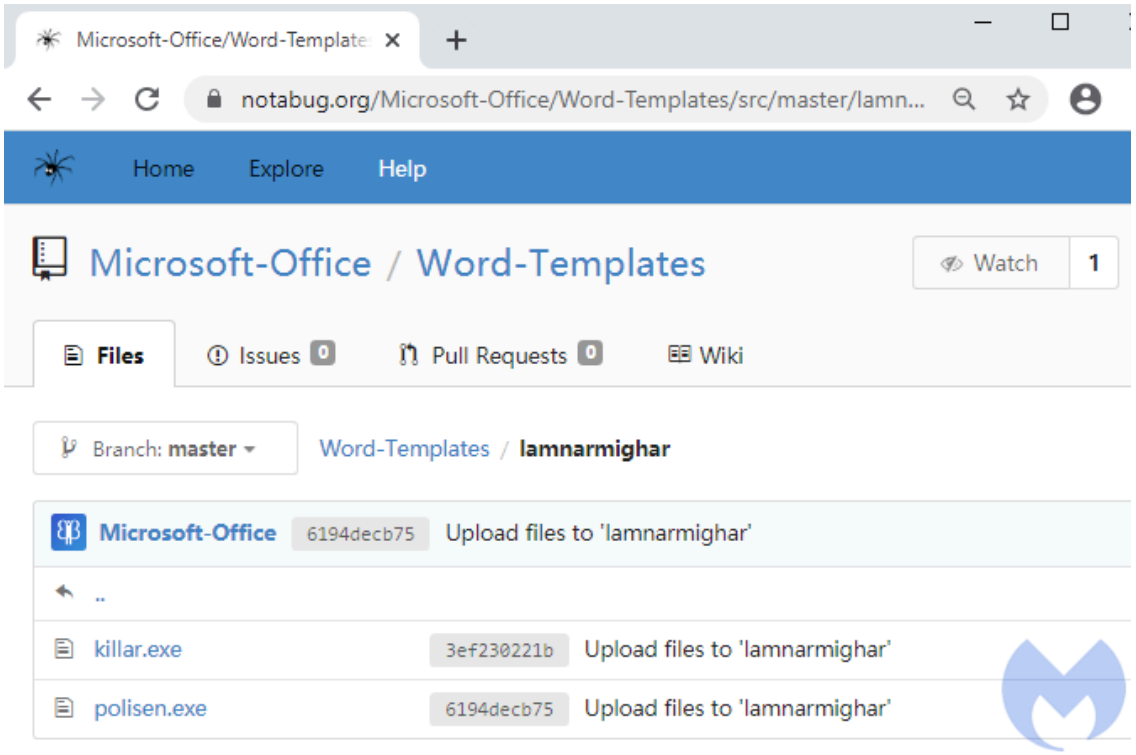
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



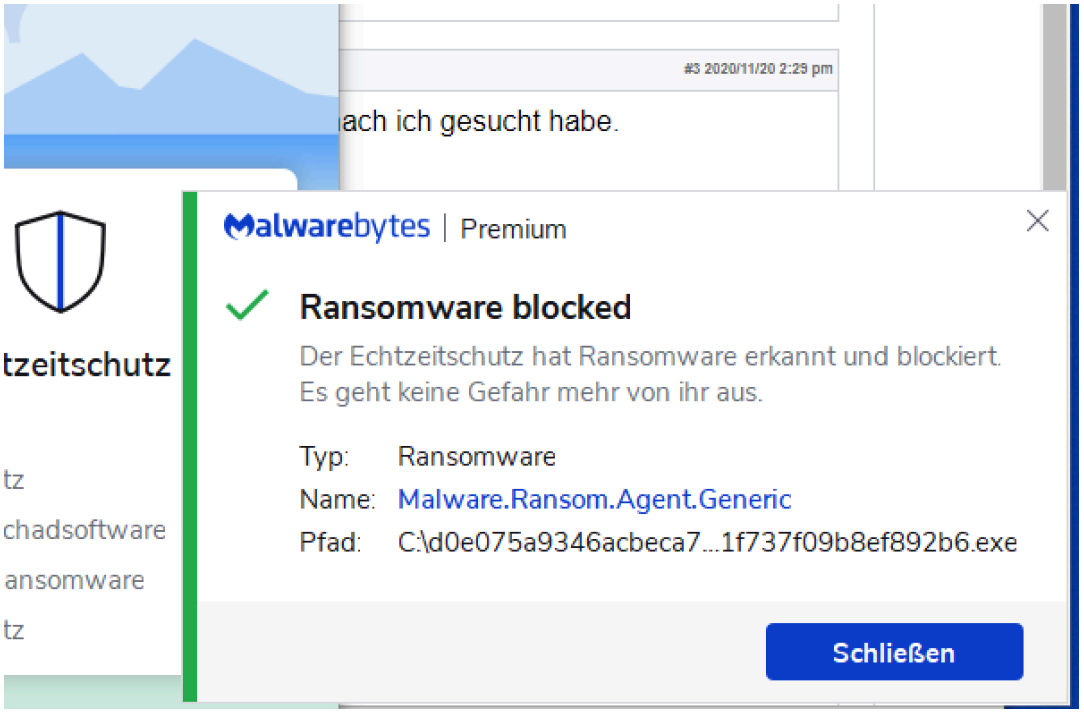
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

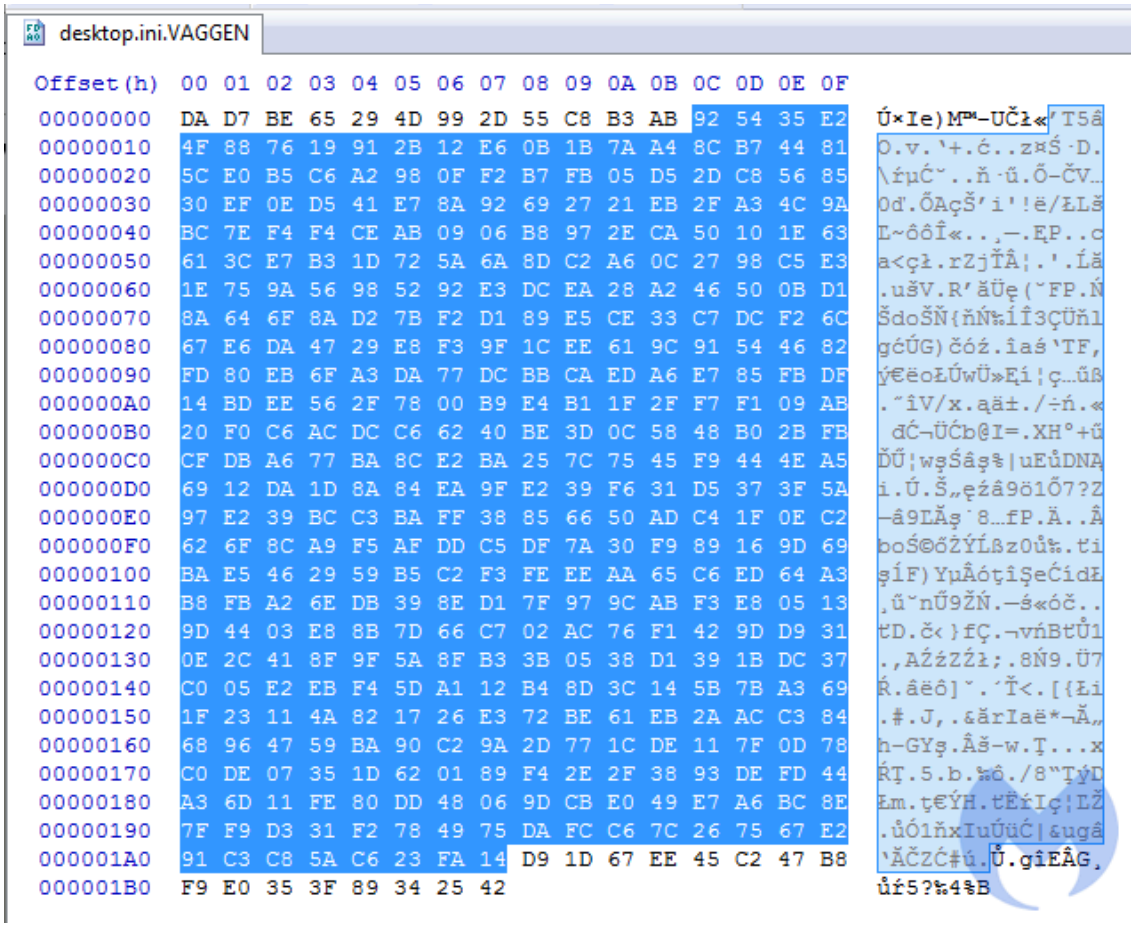
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

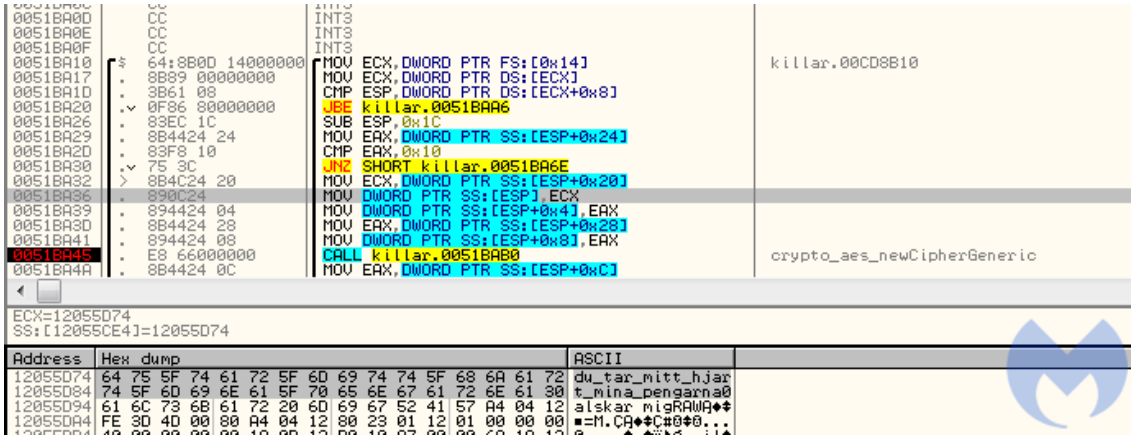


The content of the output file (with .VAGGEN extension) contains:

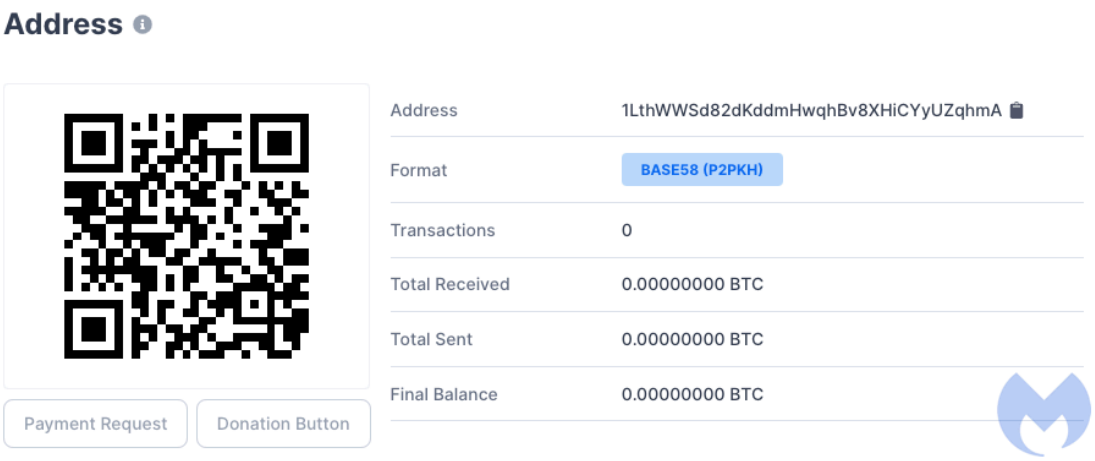
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

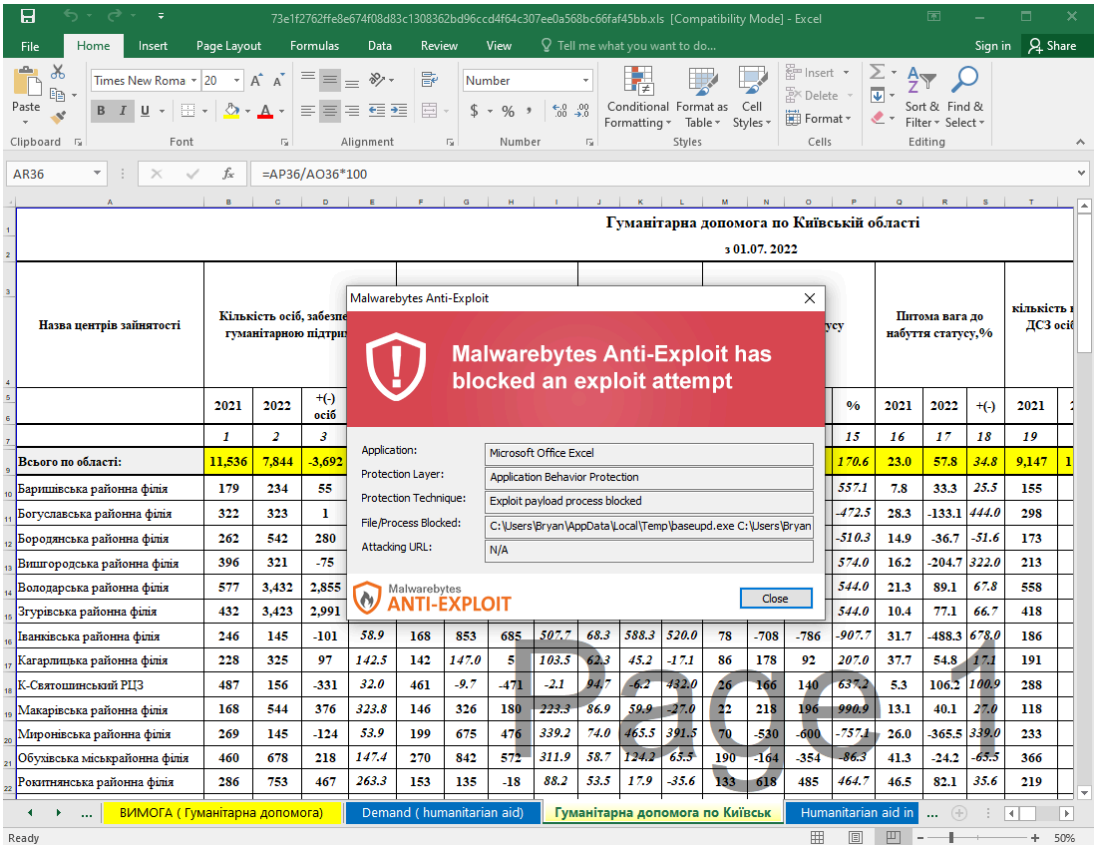
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

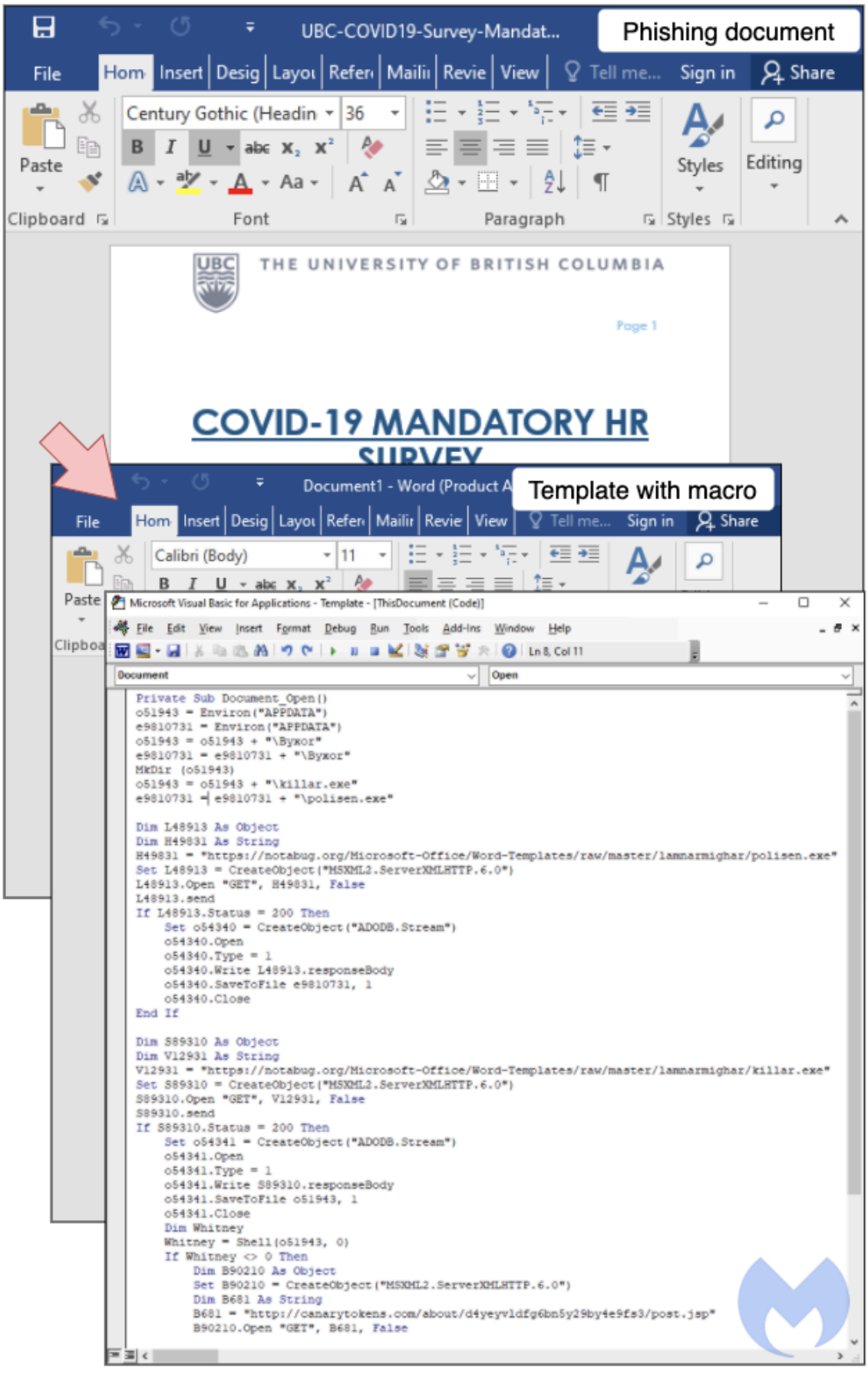
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

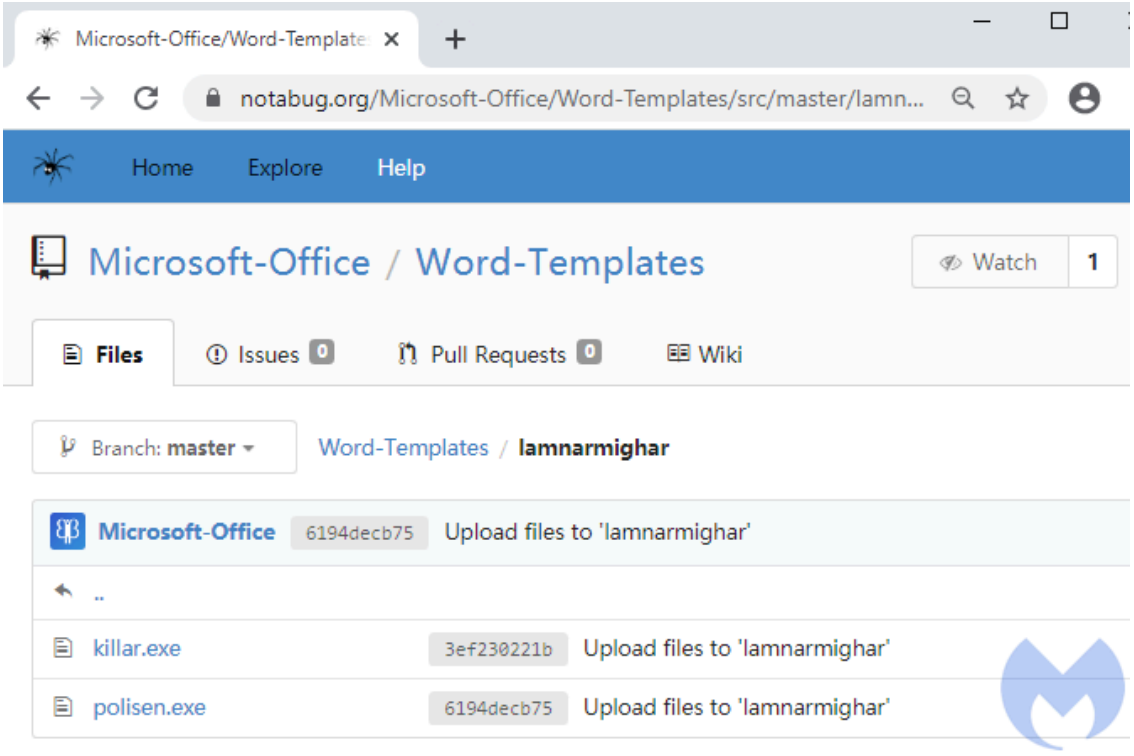
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



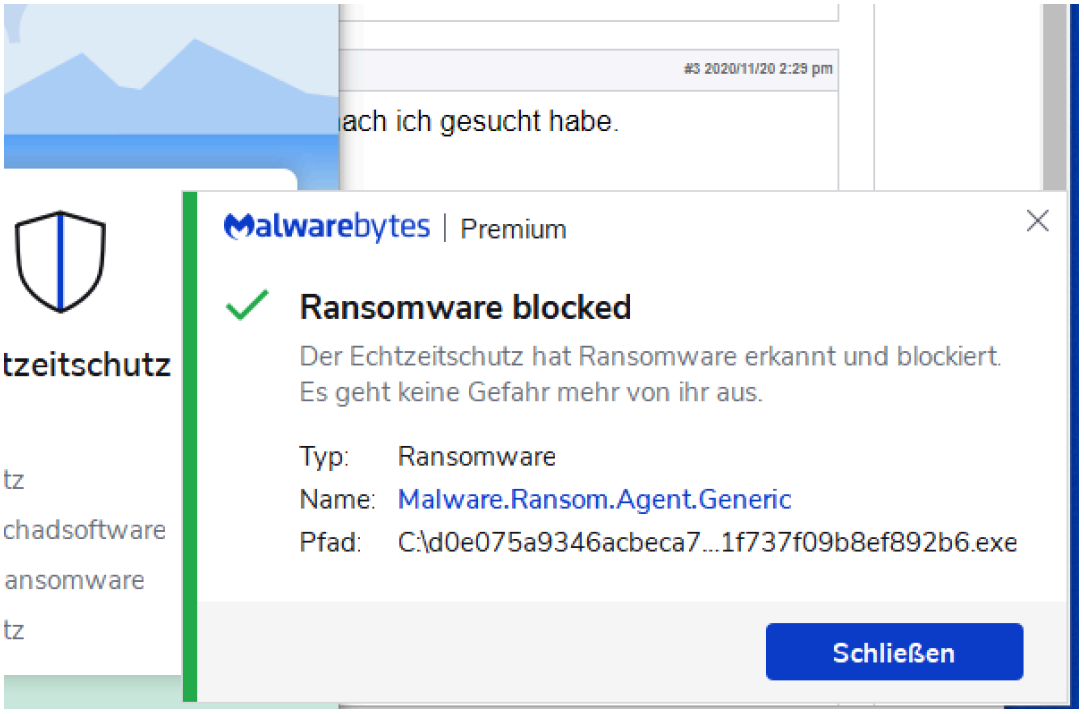
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

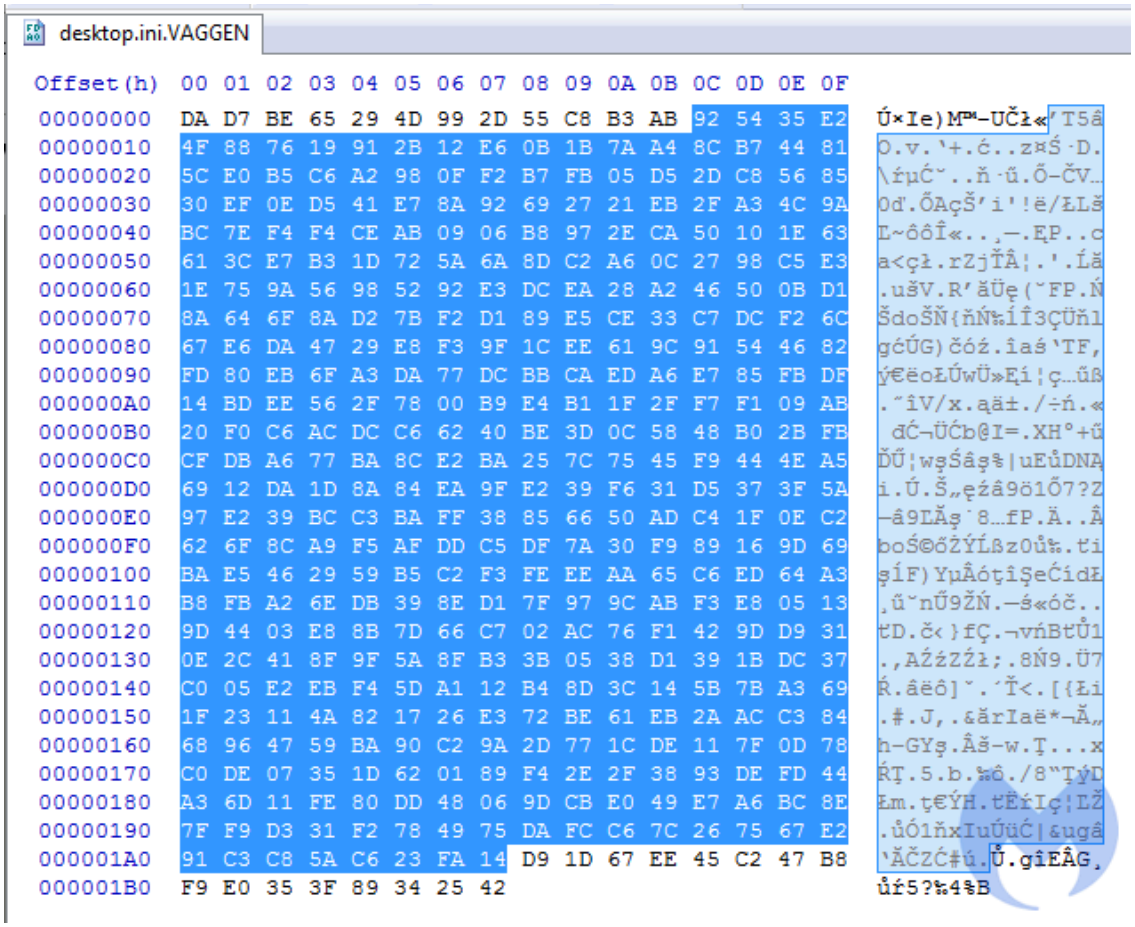
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

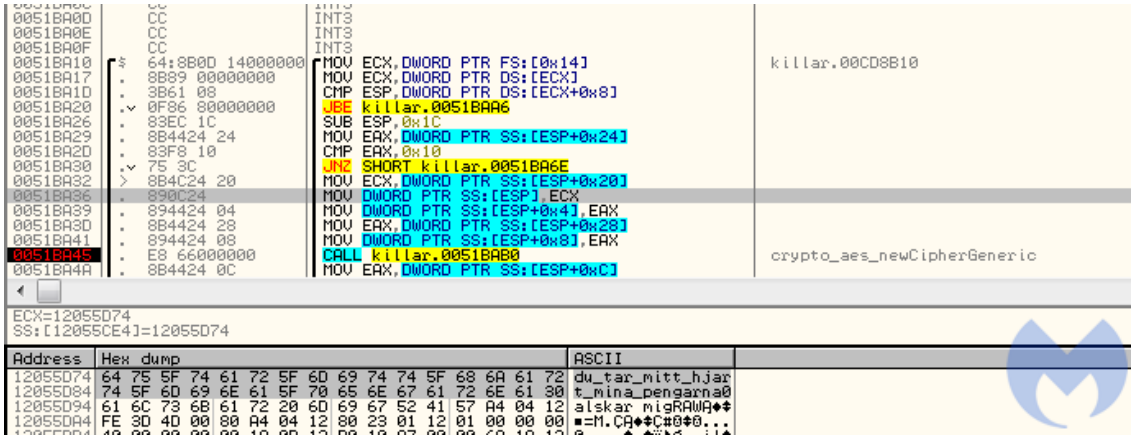


The content of the output file (with .VAGGEN extension) contains:

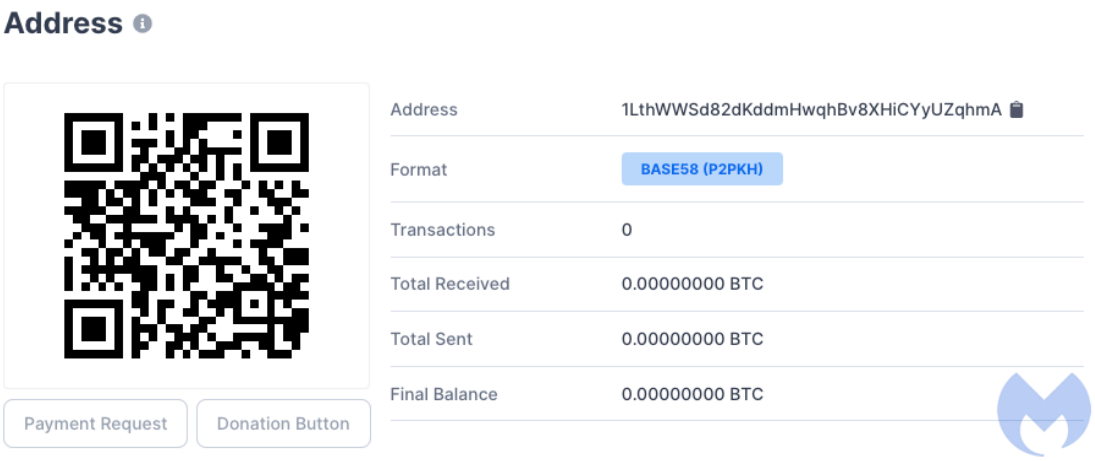
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

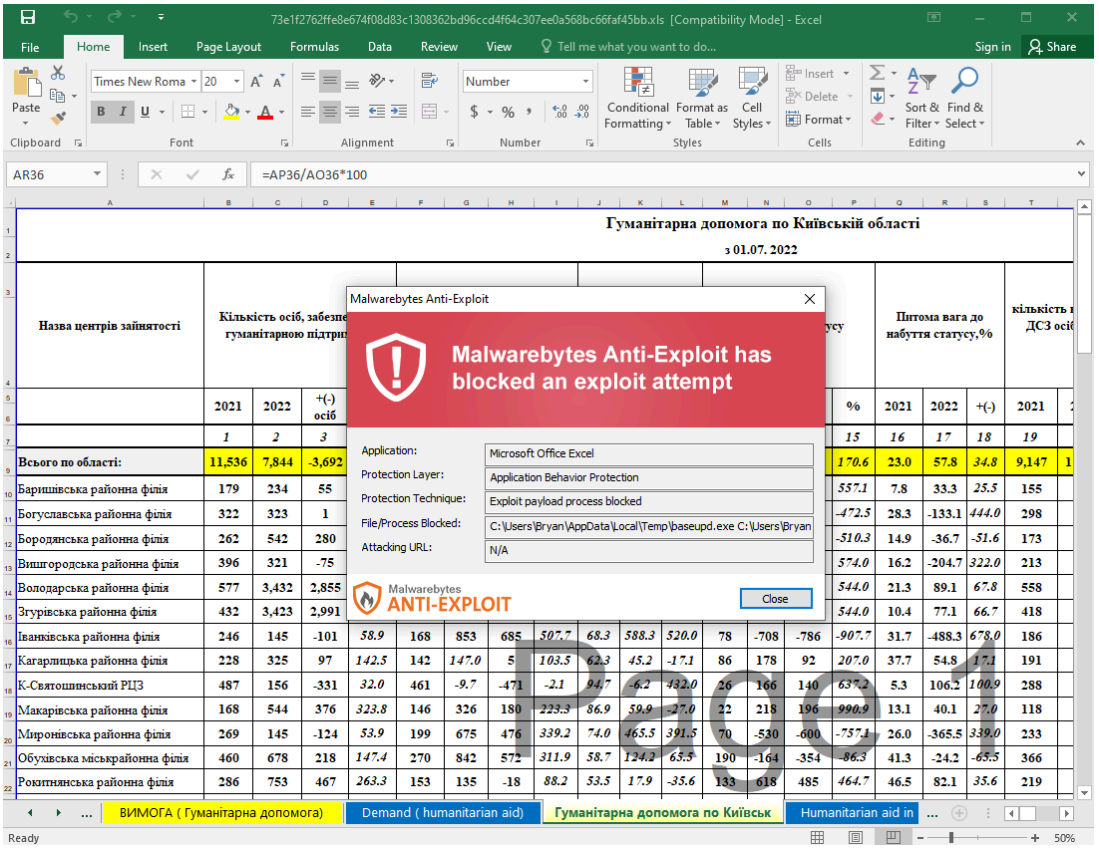
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

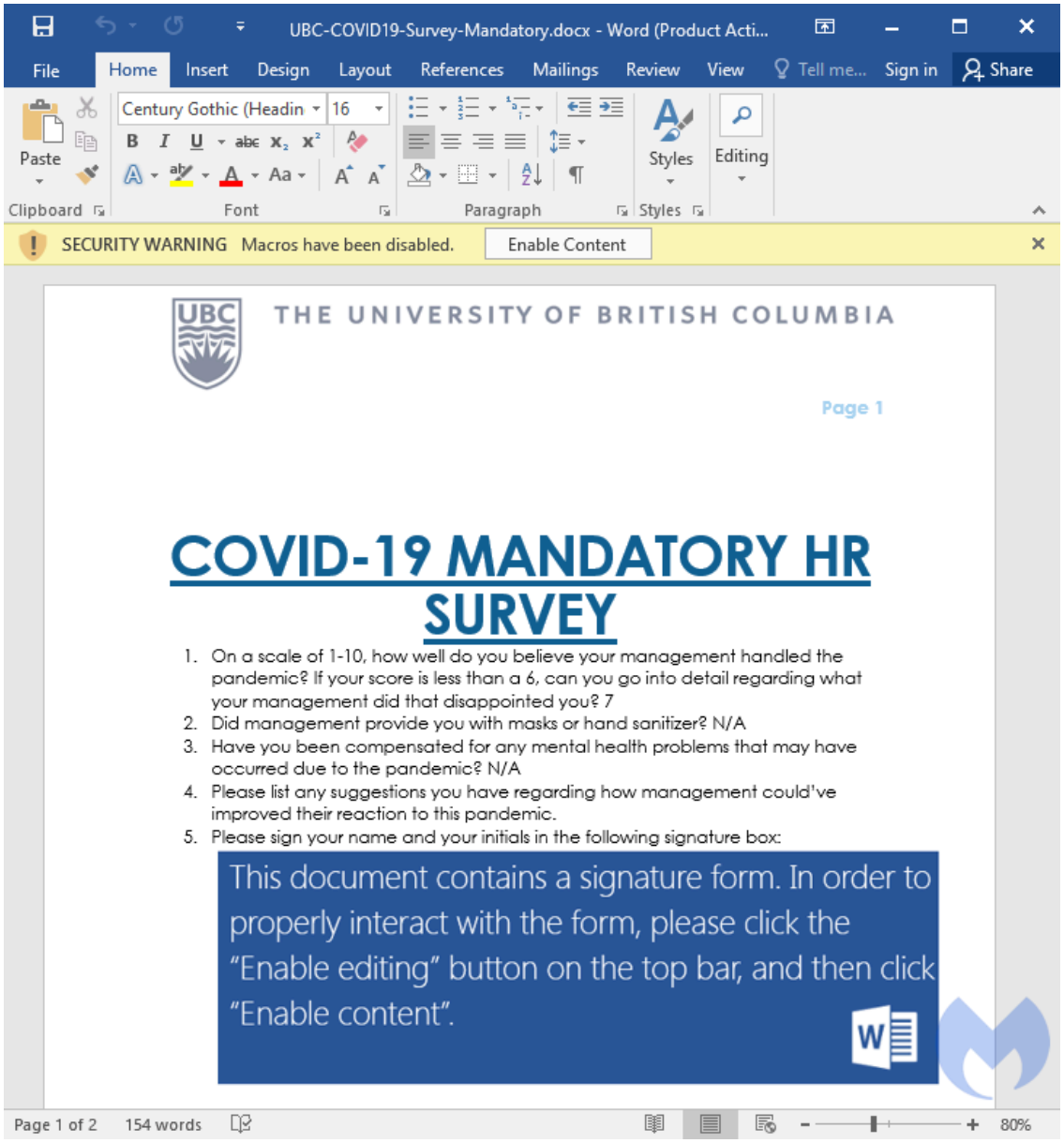
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

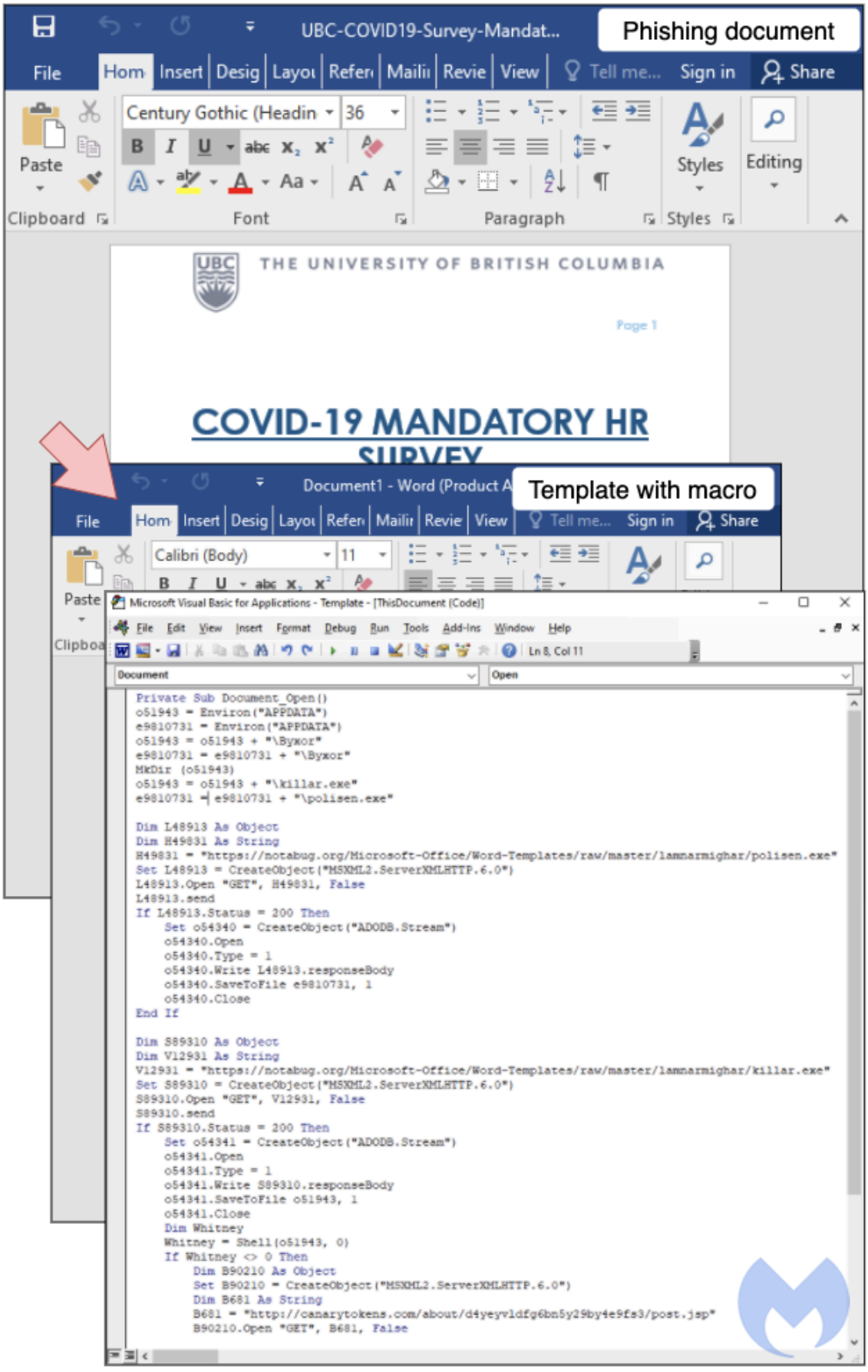
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

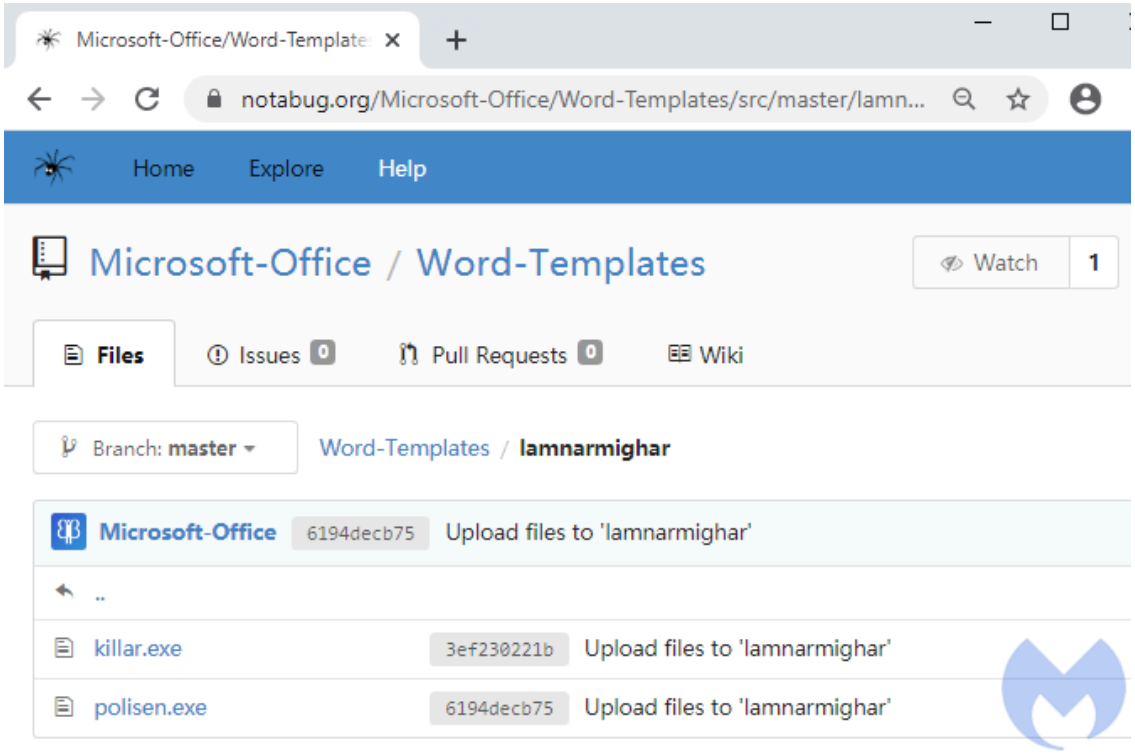
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



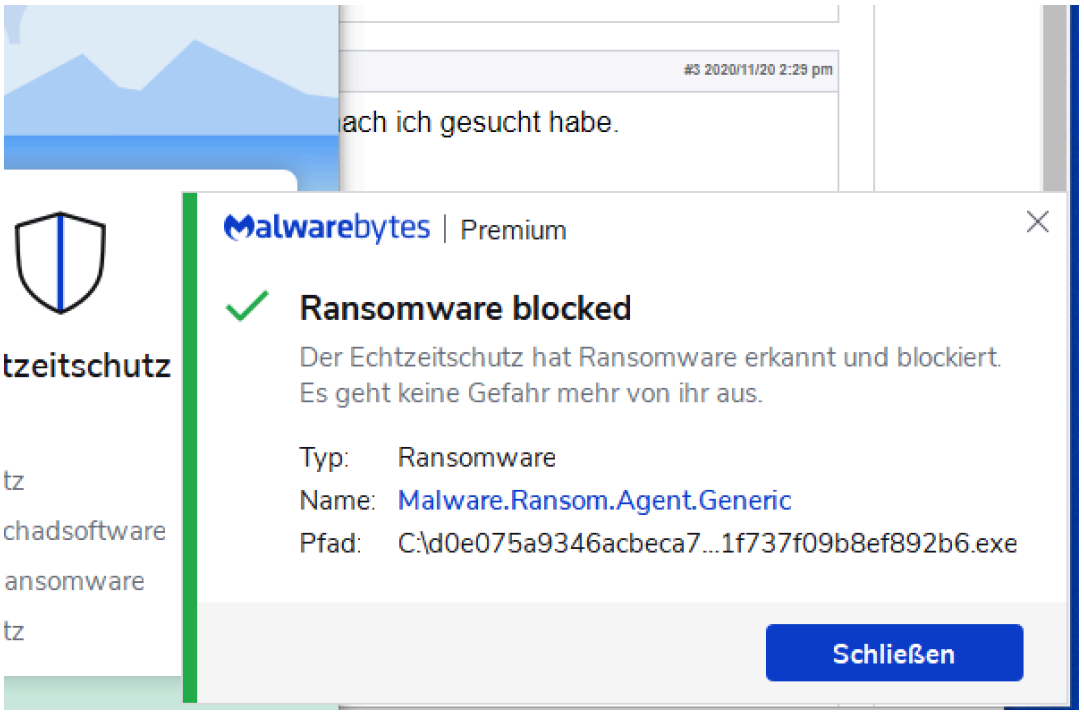
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

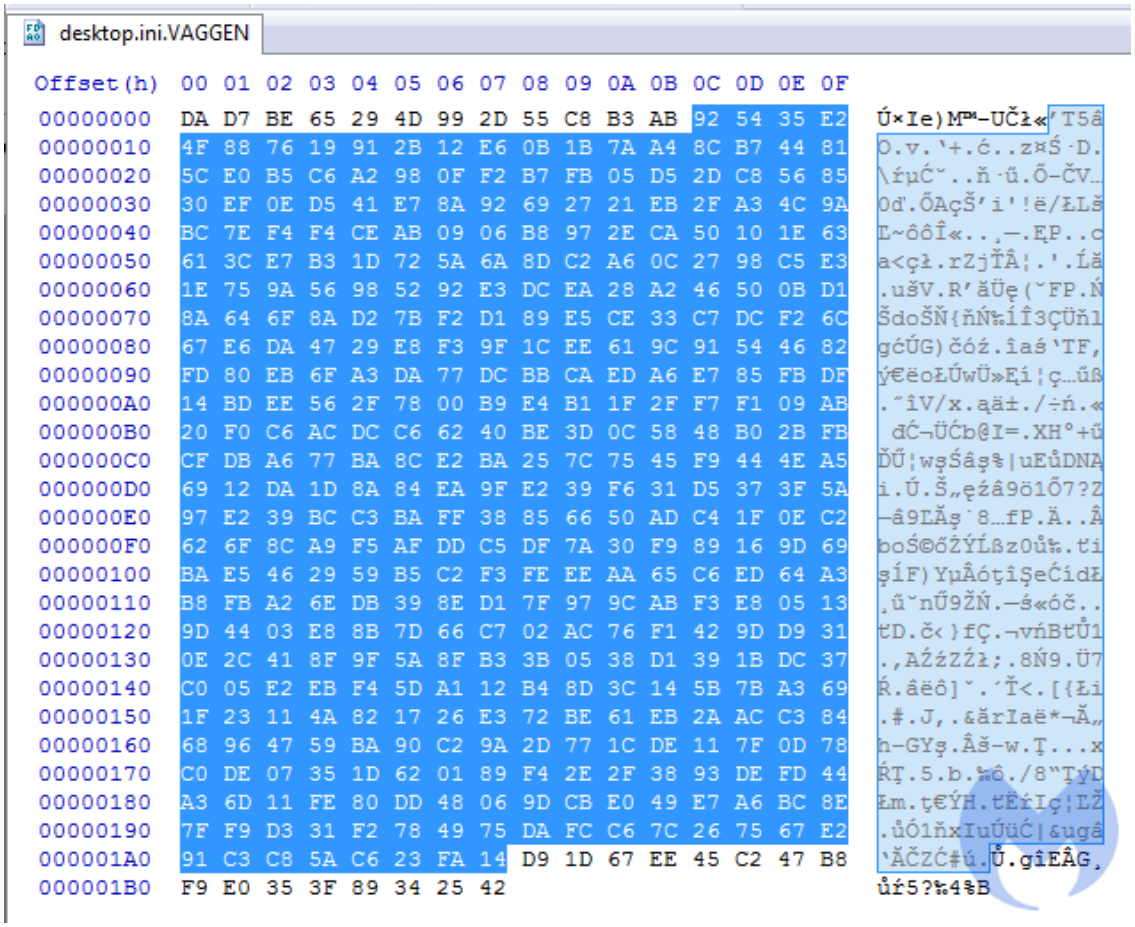
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

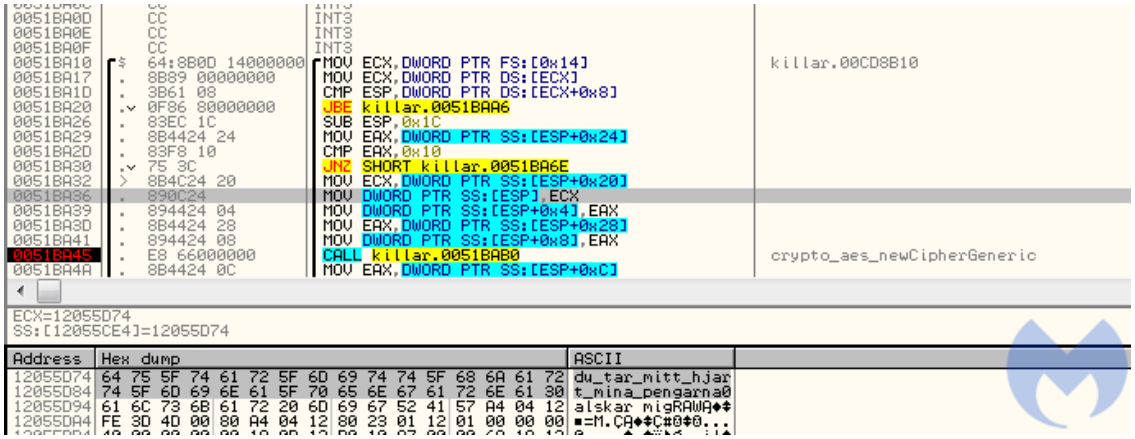


The content of the output file (with .VAGGEN extension) contains:

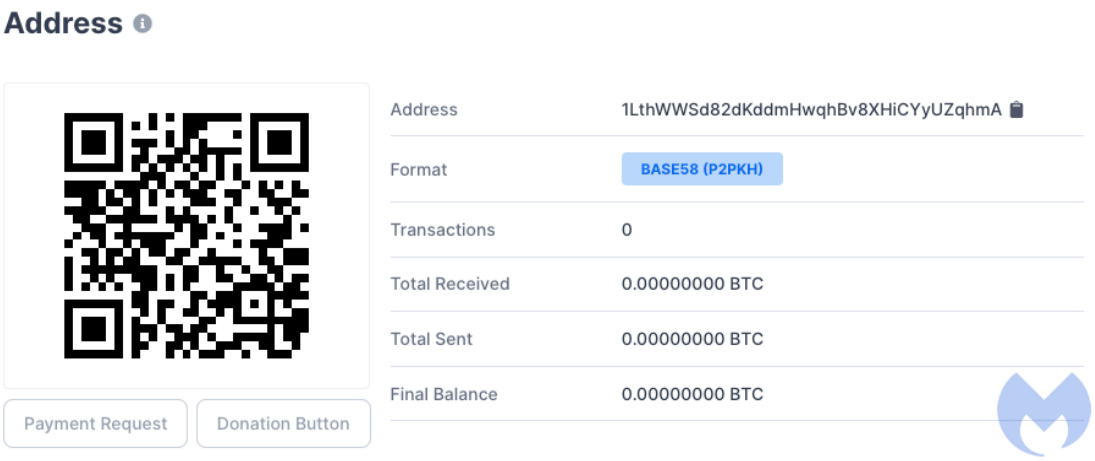
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

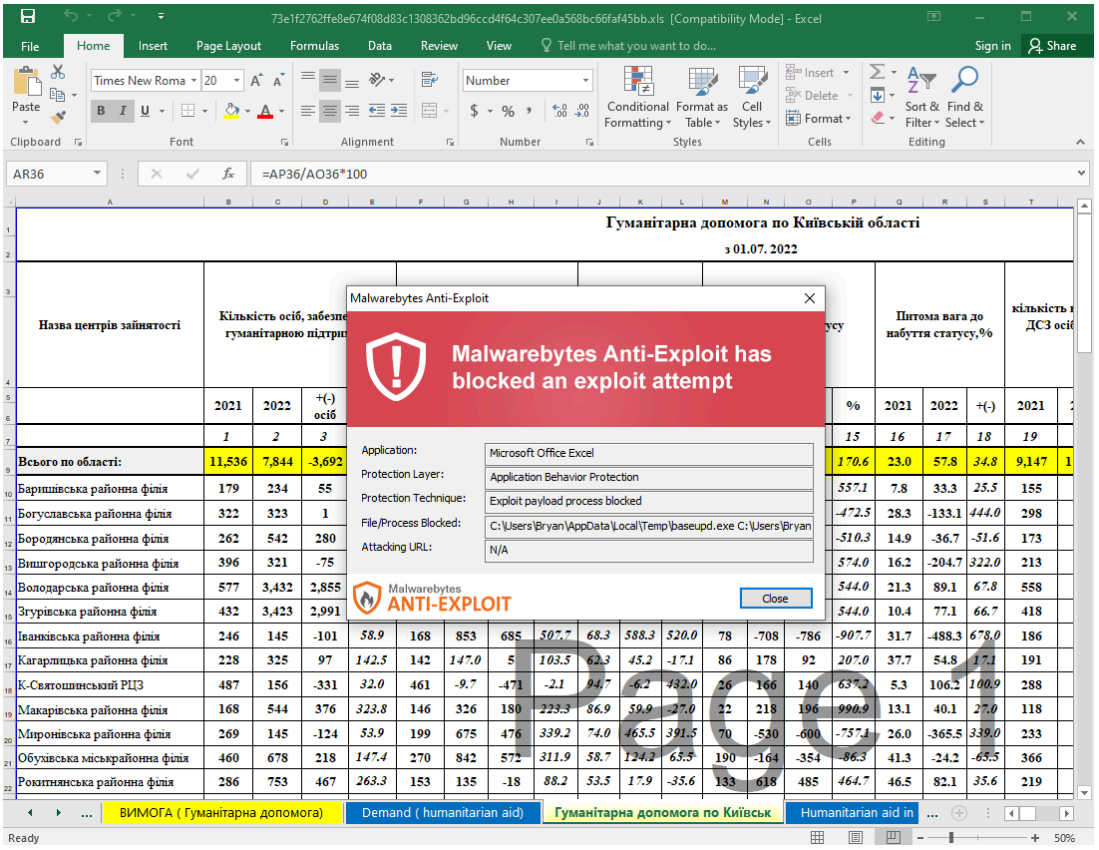
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

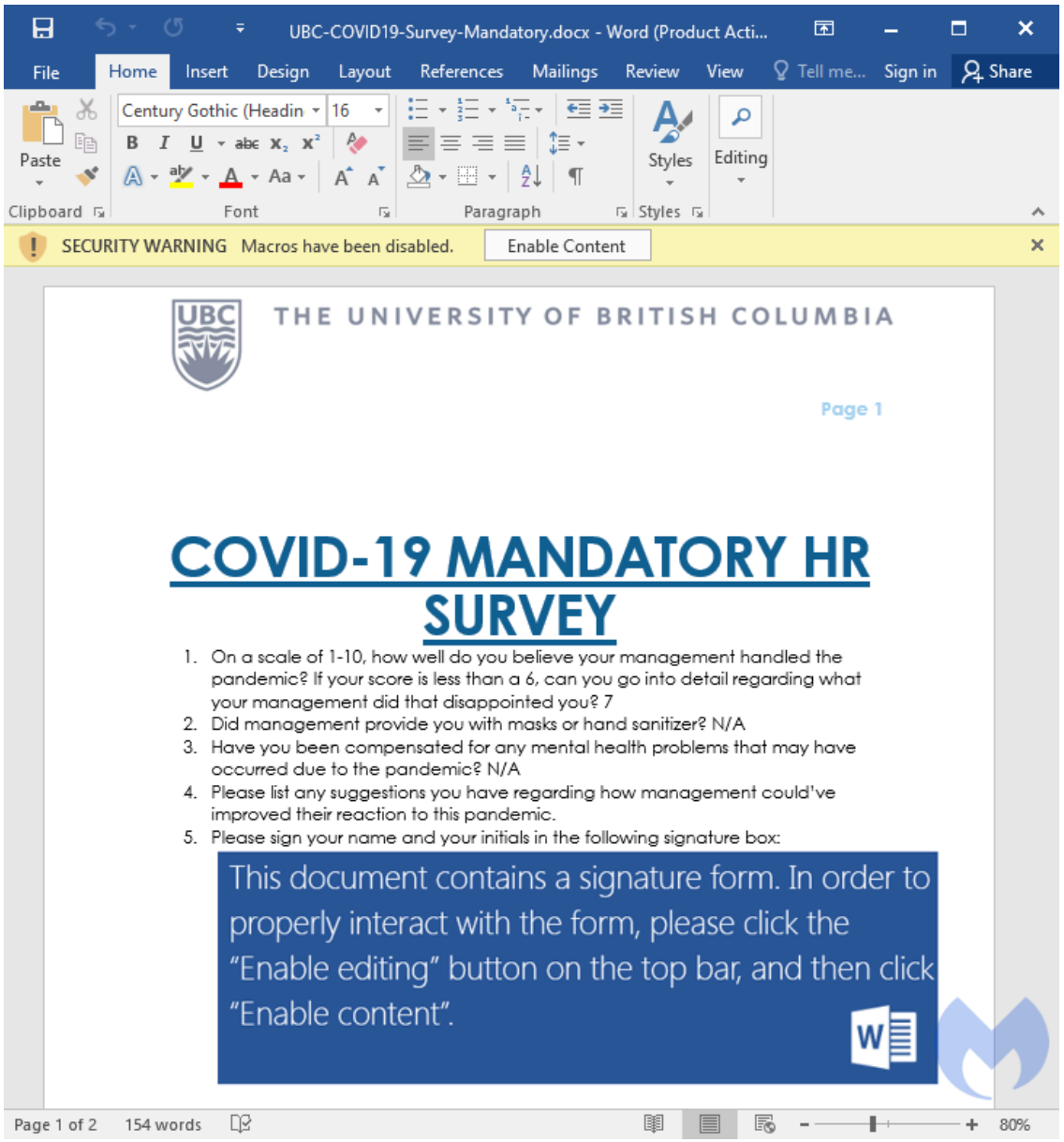
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

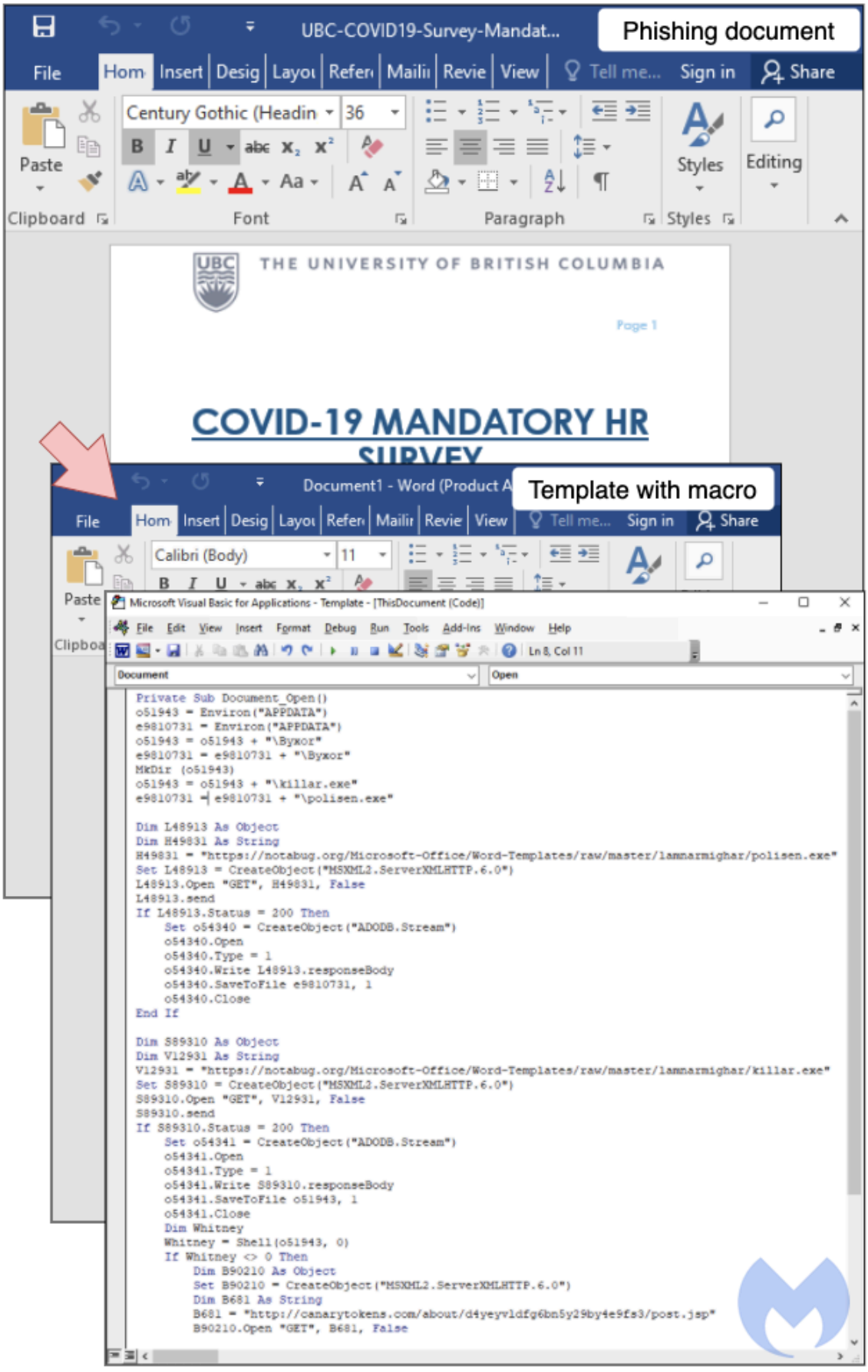
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

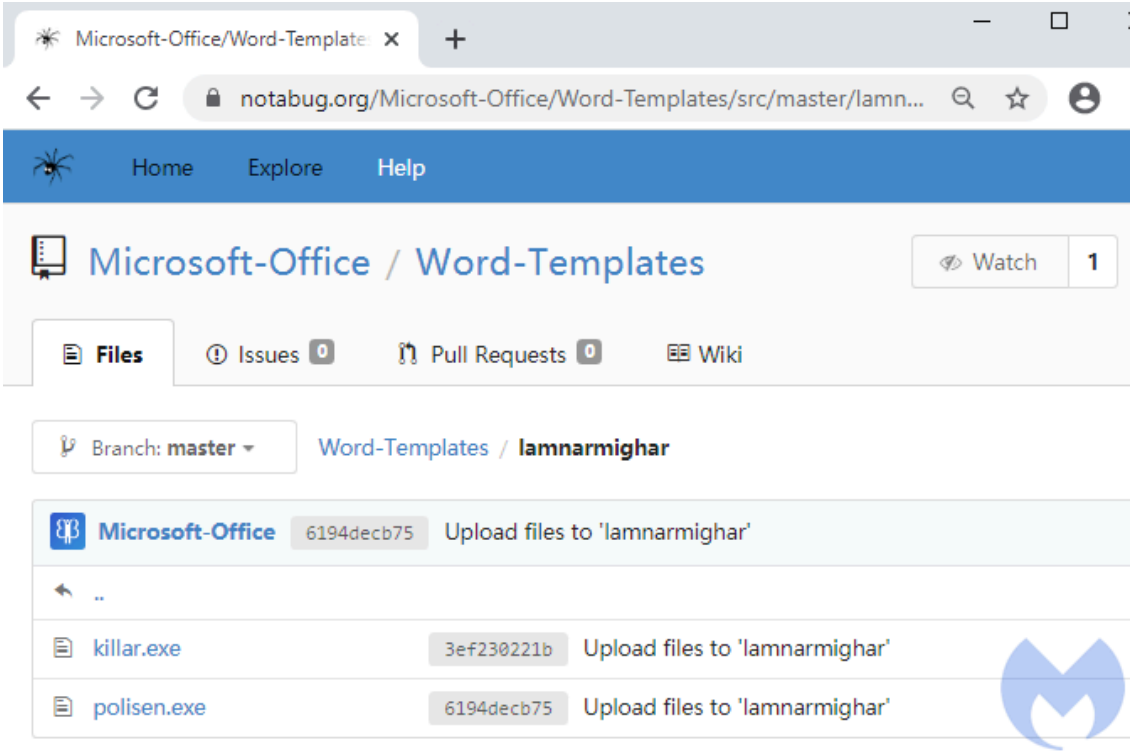
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



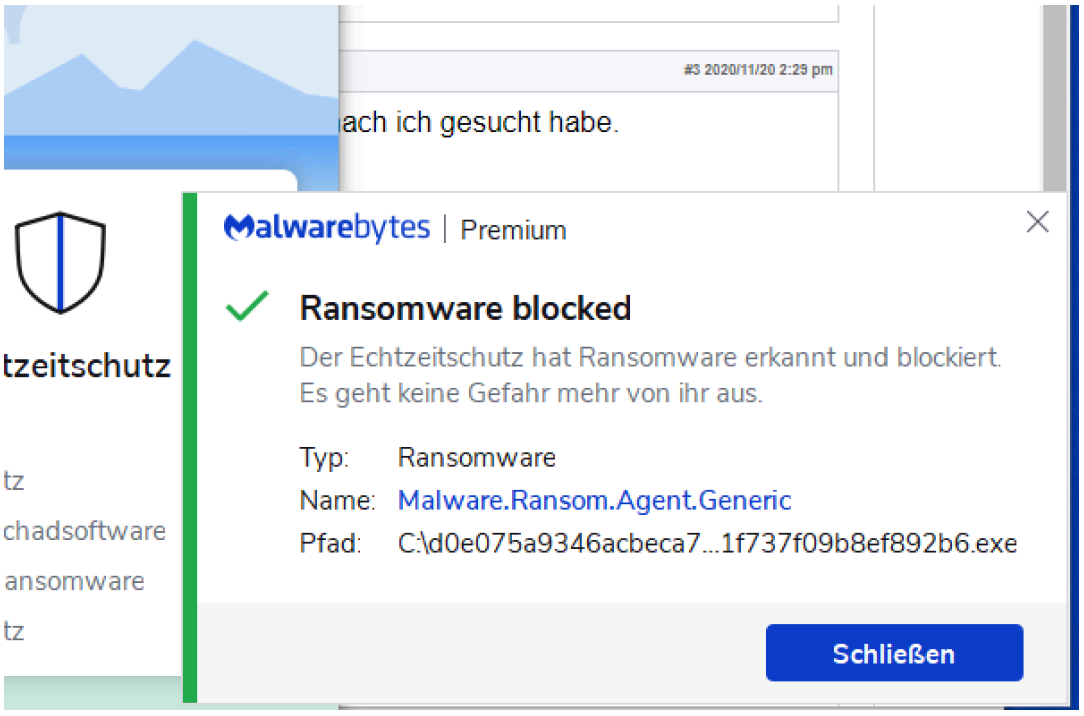
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

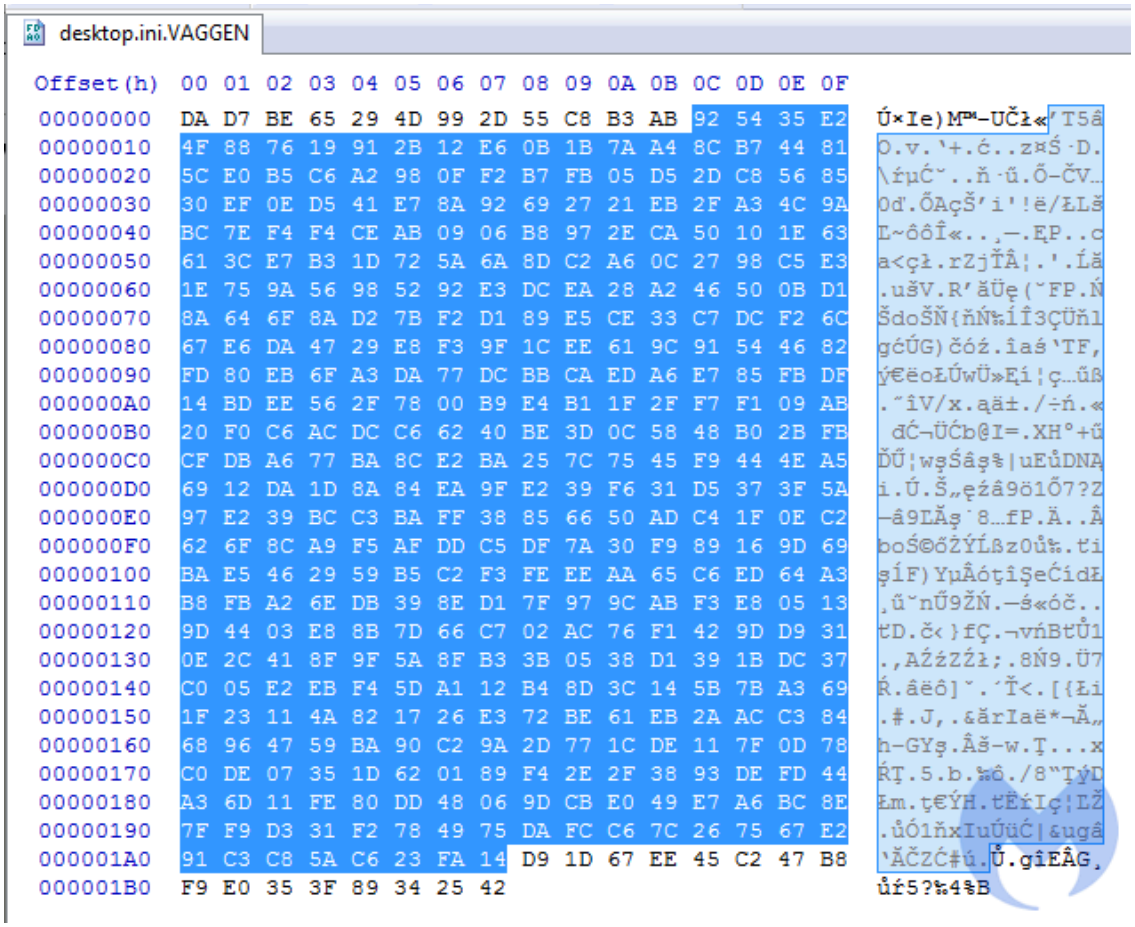
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

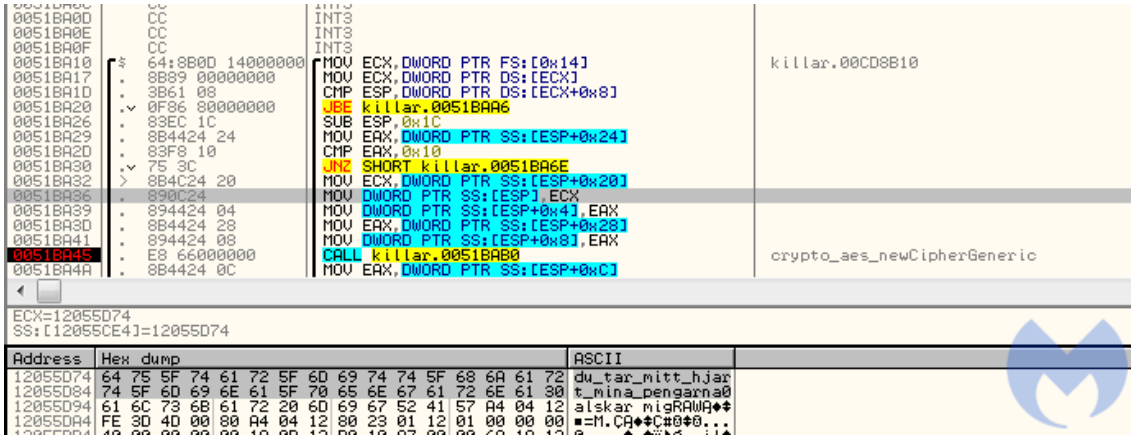


The content of the output file (with .VAGGEN extension) contains:

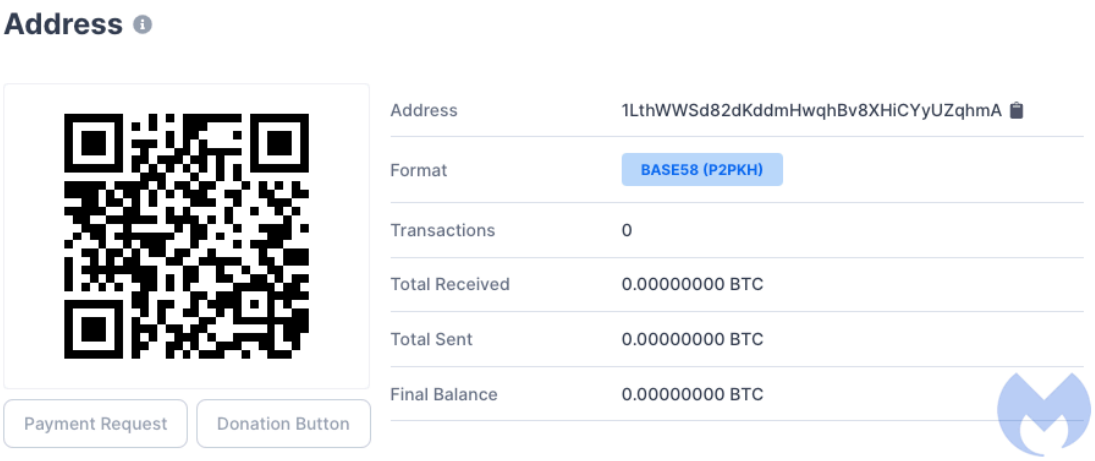
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

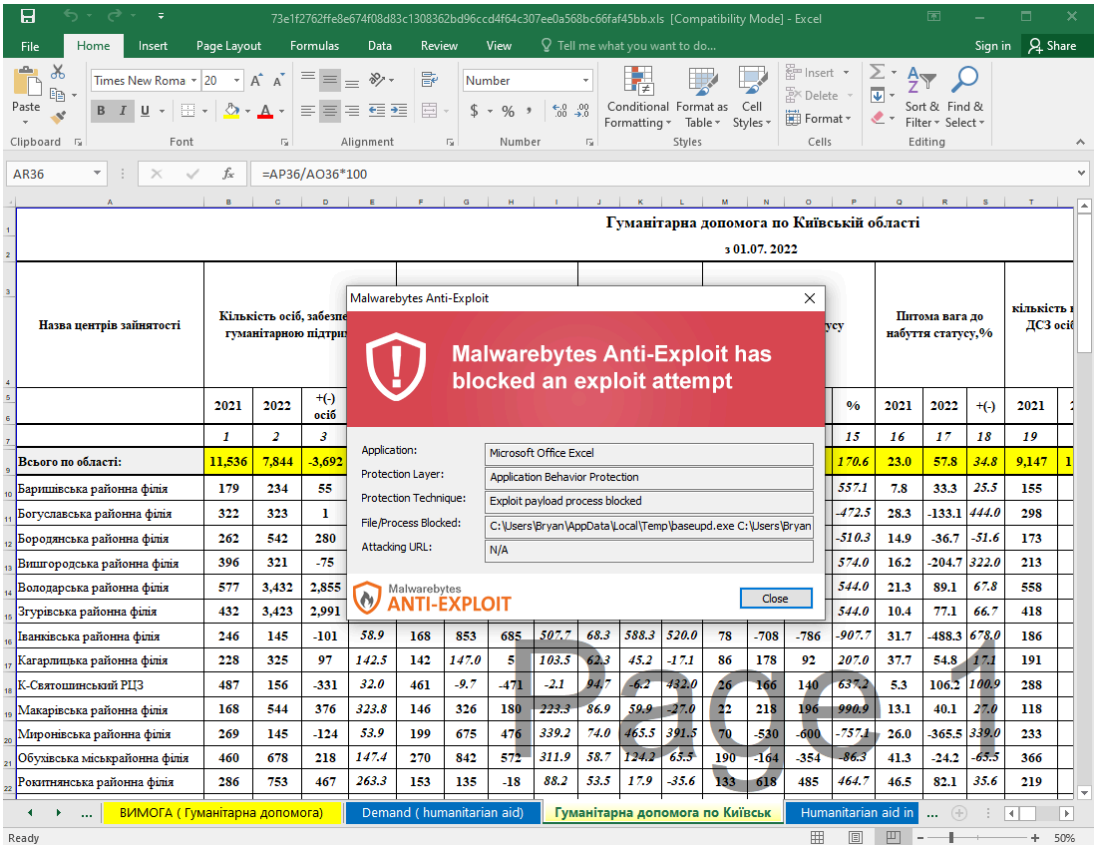
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

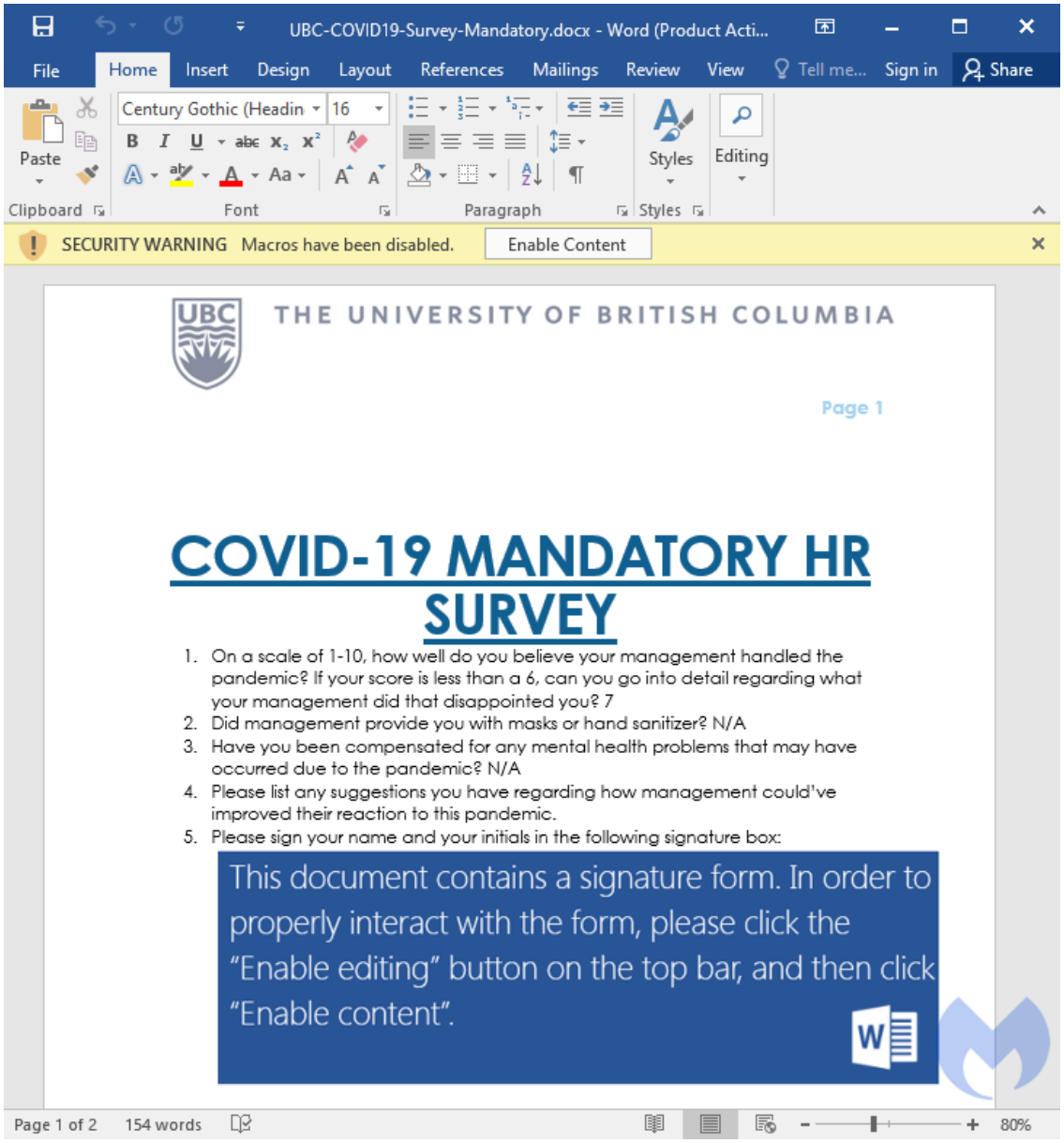
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

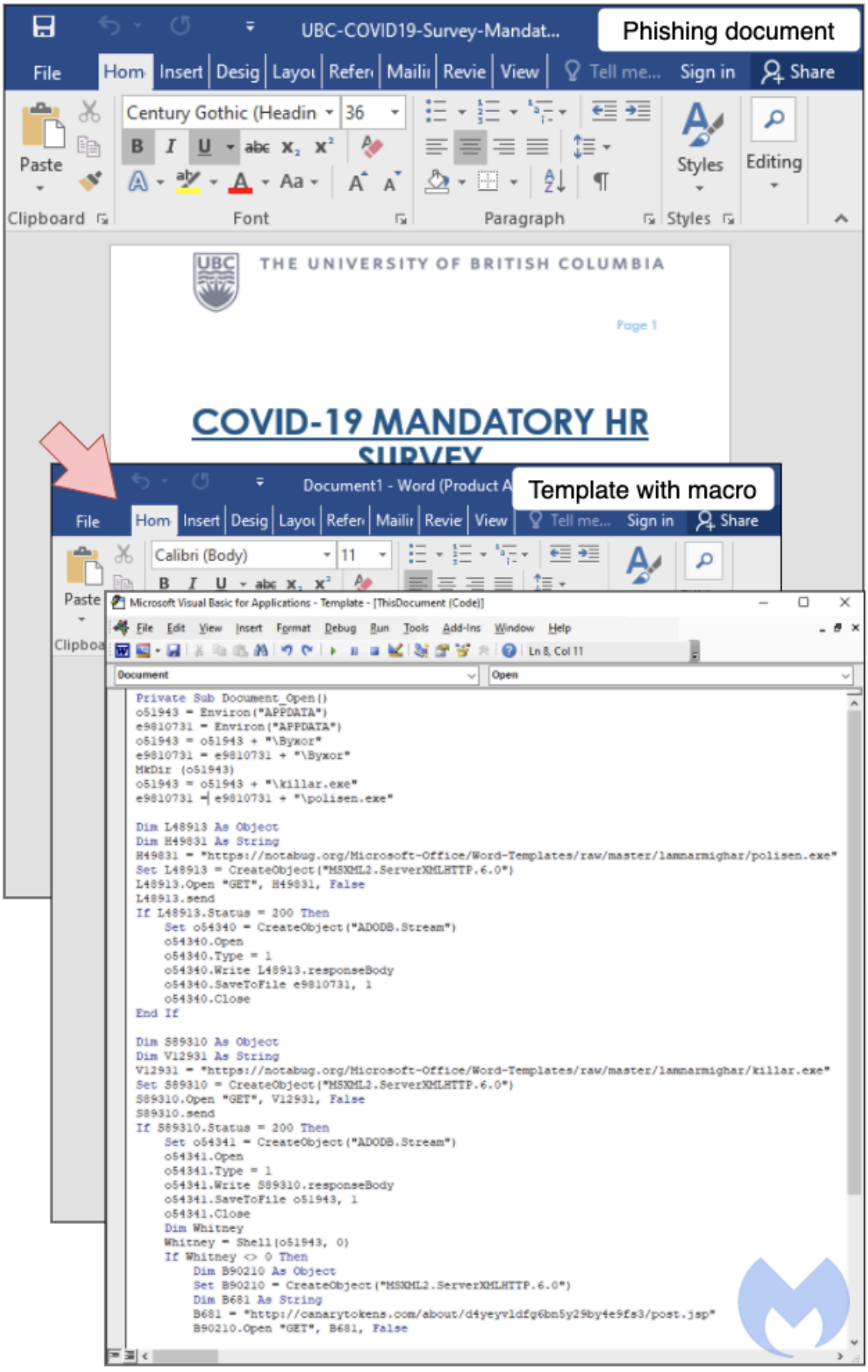
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

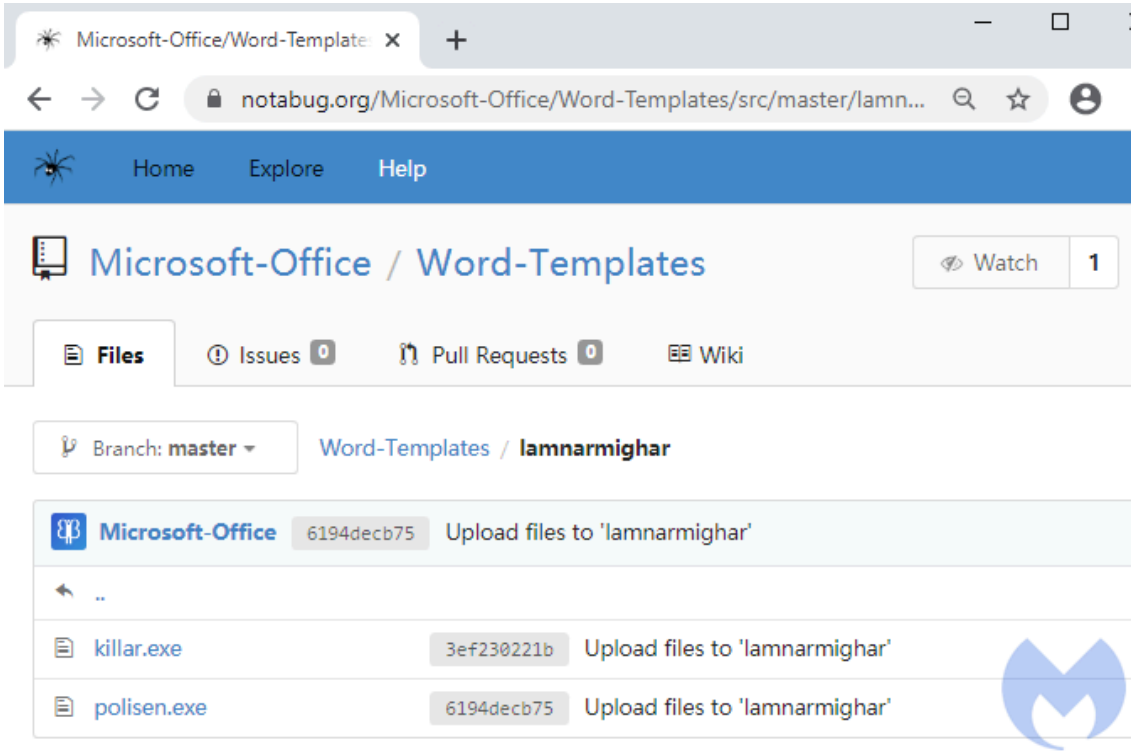
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



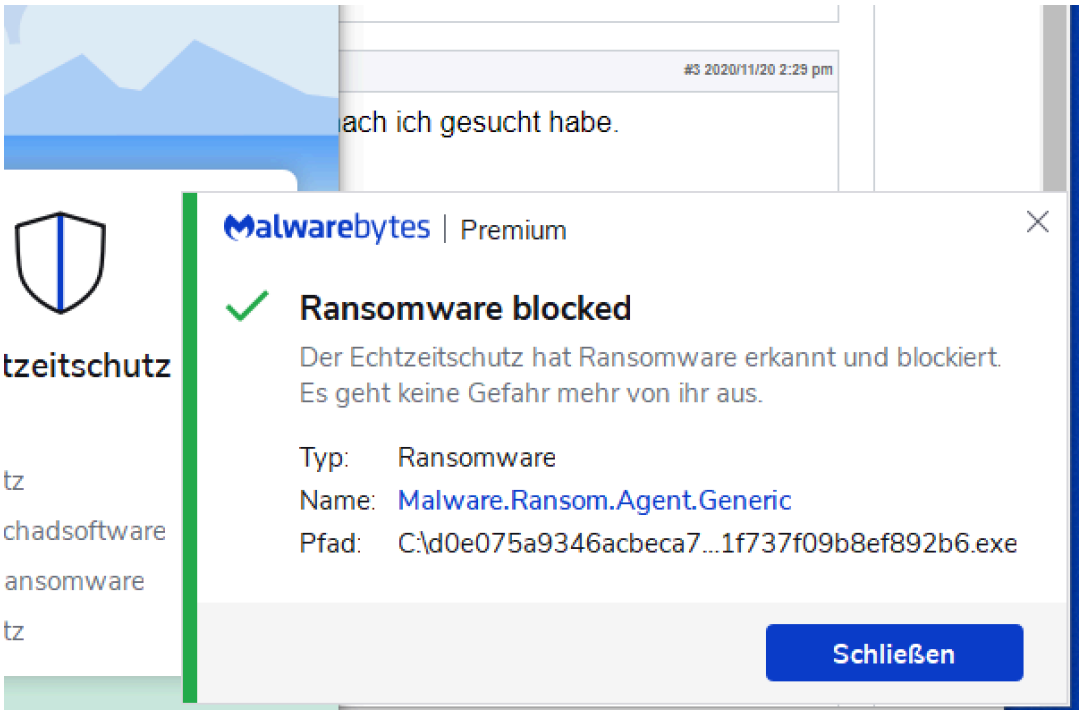
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

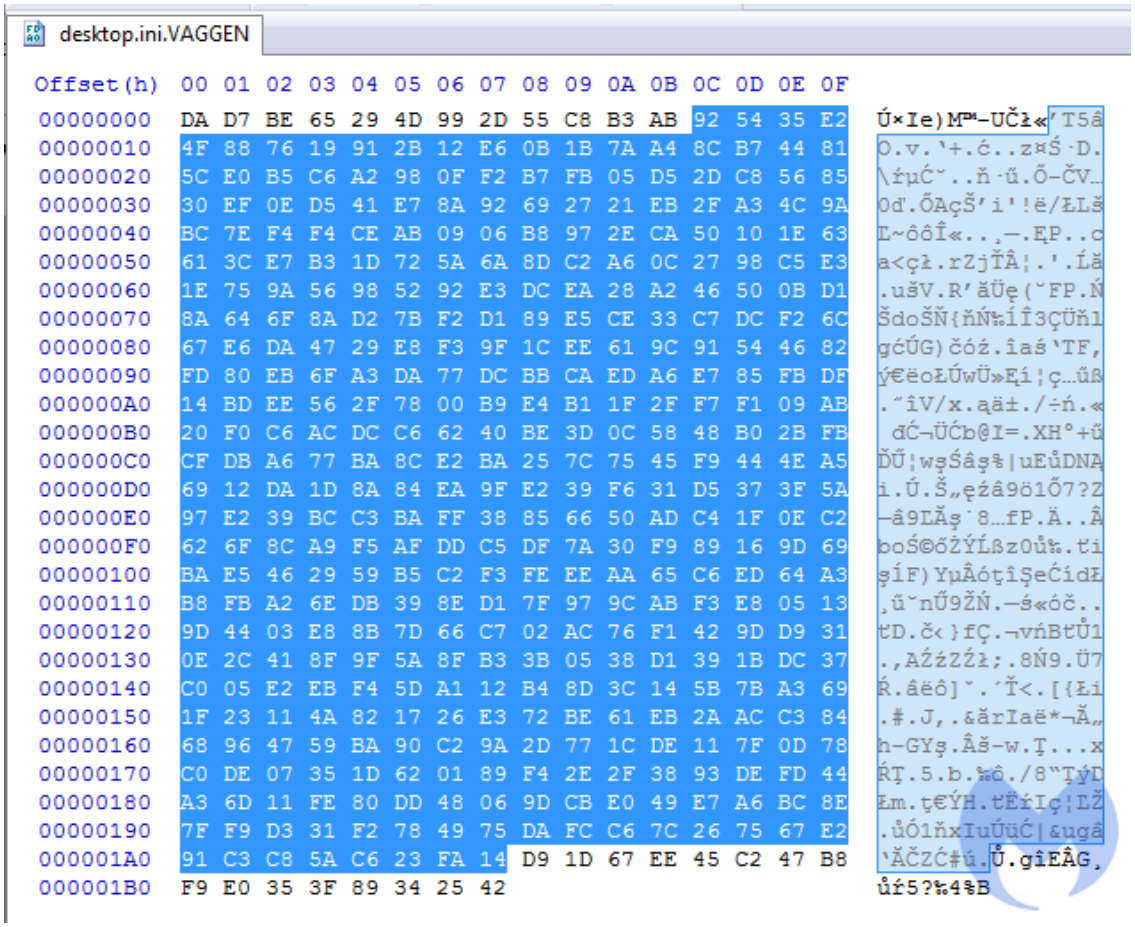
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

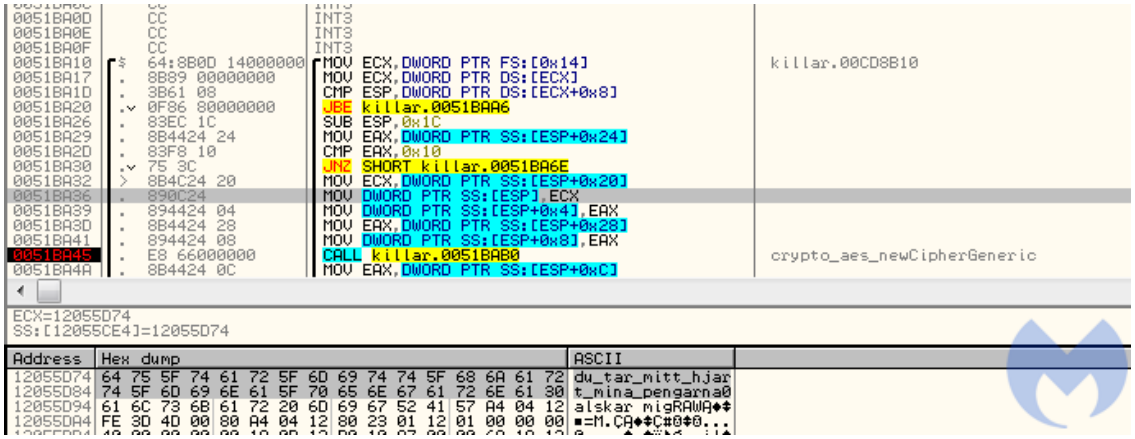


The content of the output file (with .VAGGEN extension) contains:

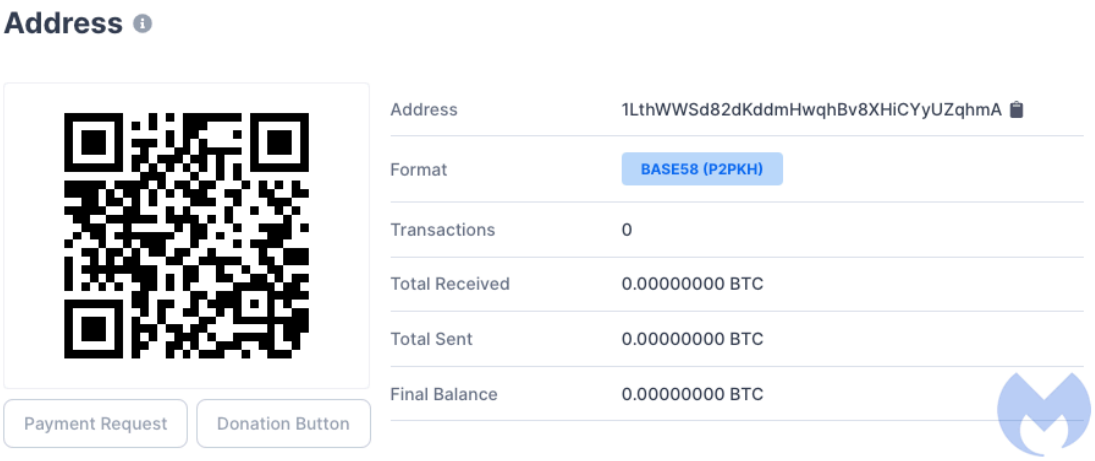
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

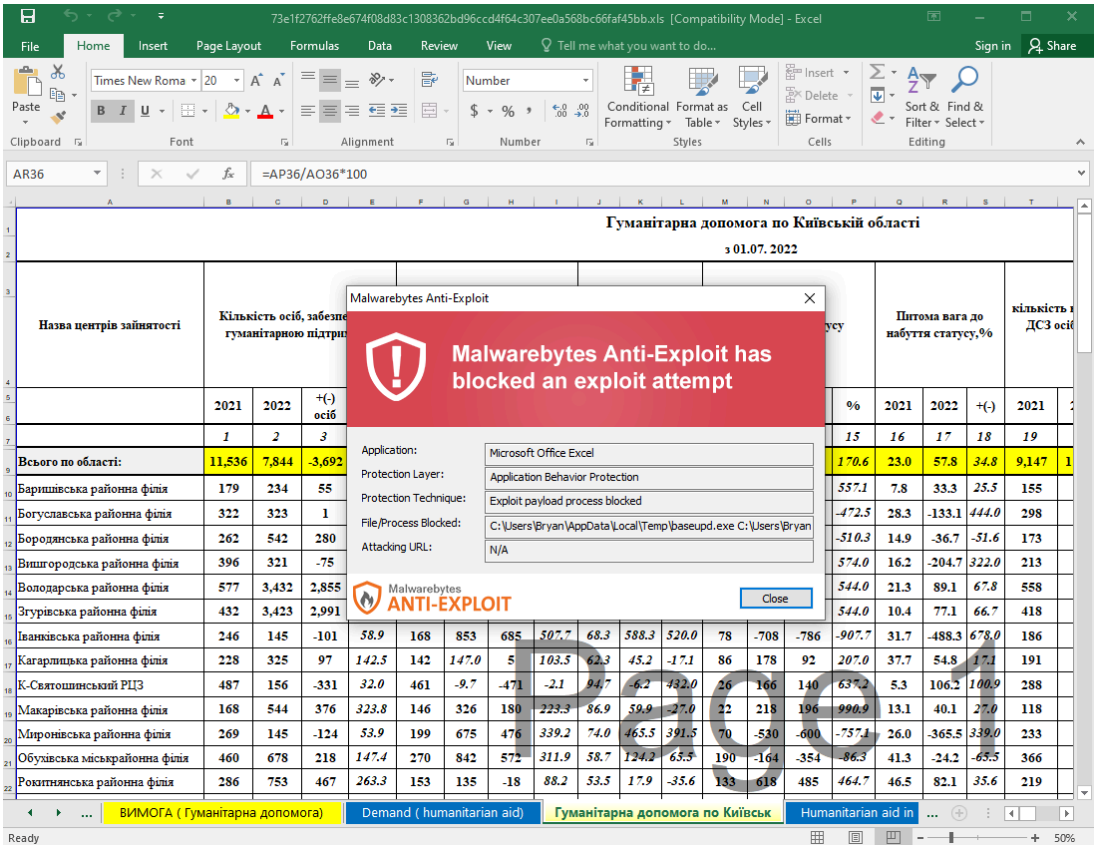
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

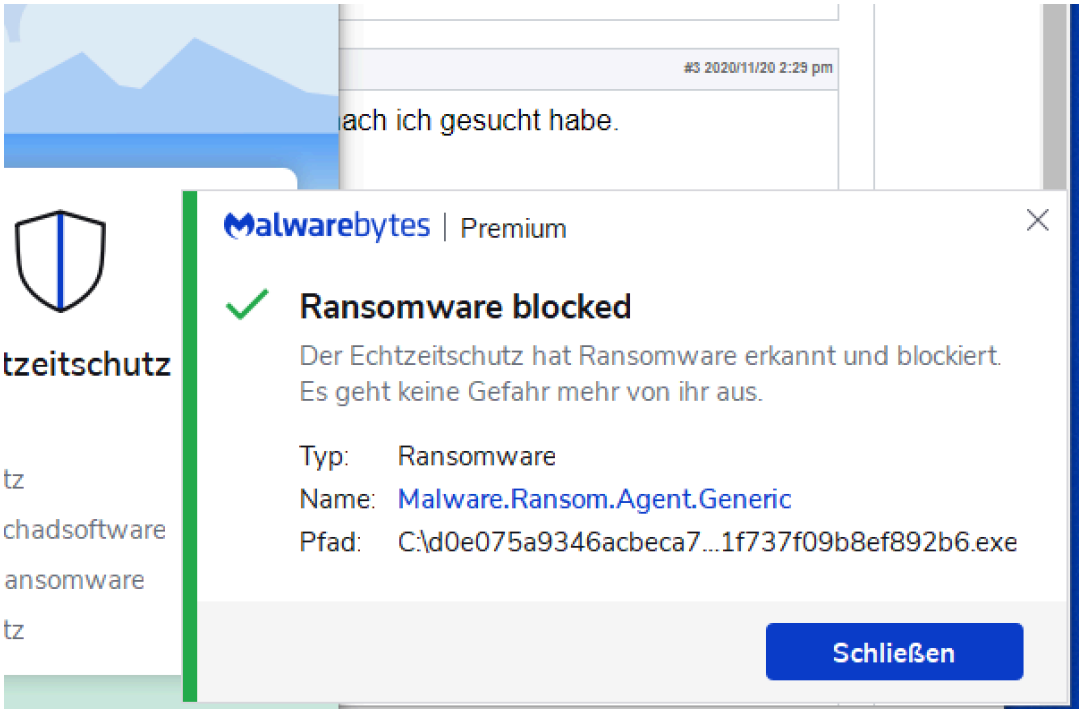
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

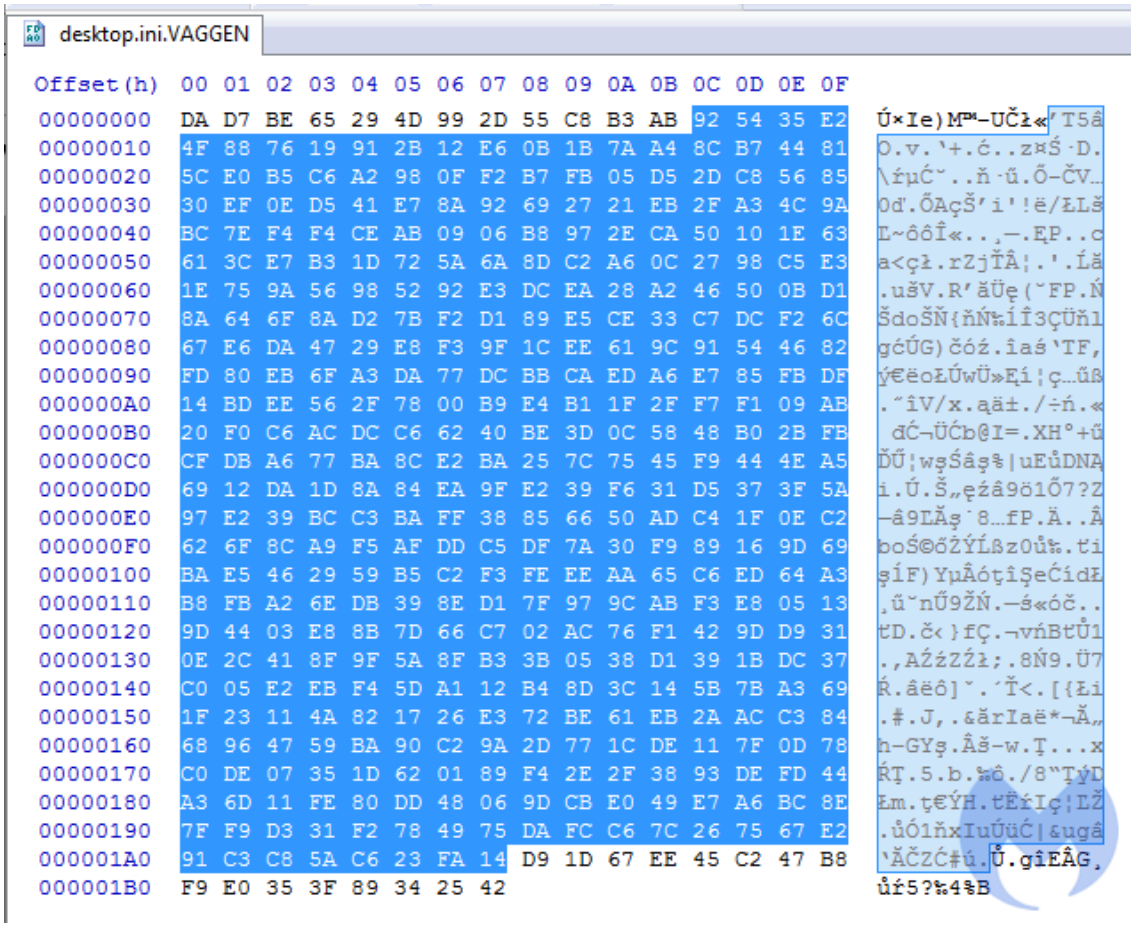
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



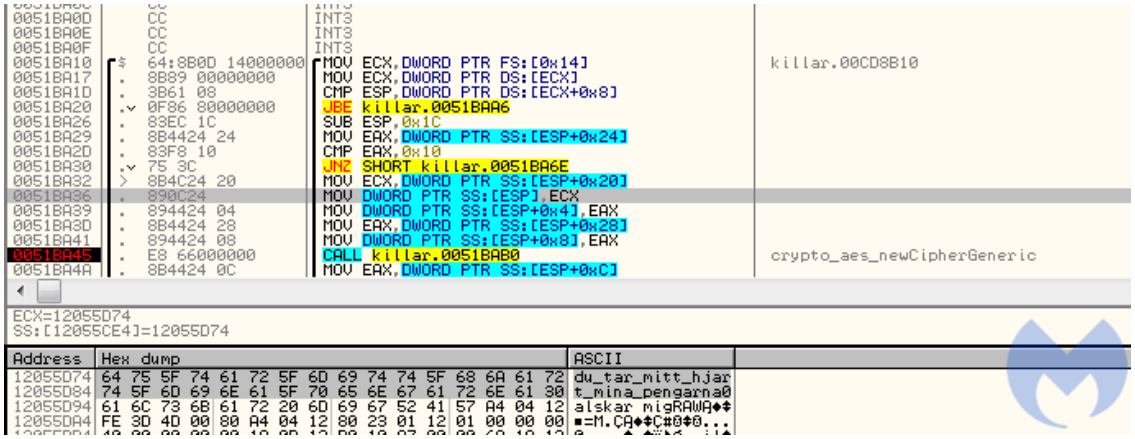
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

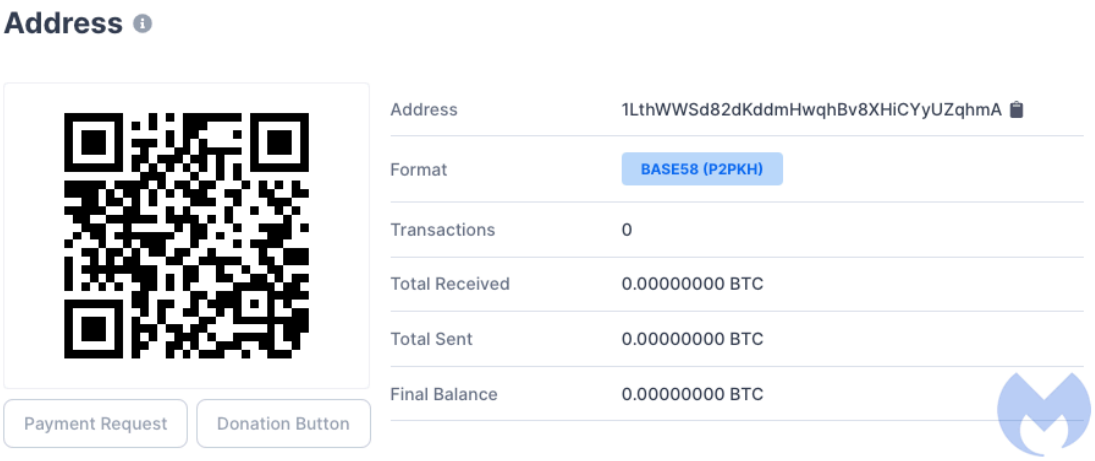
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

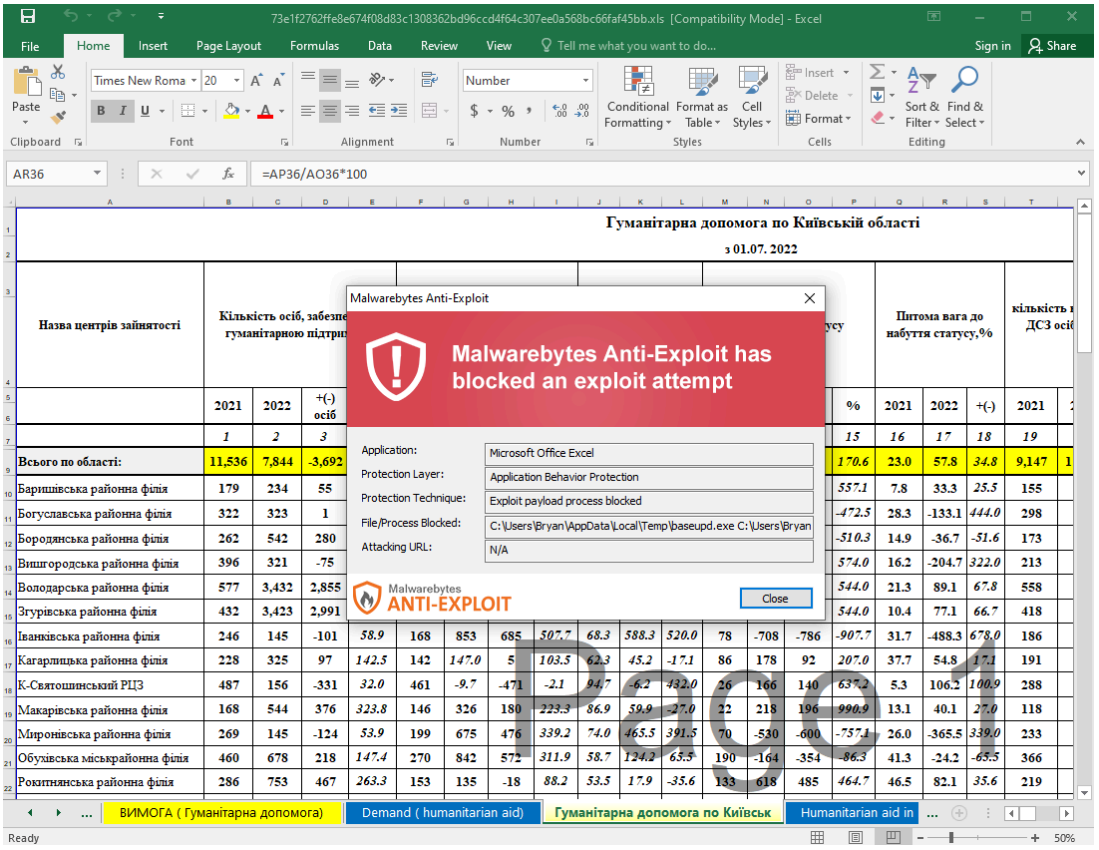
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

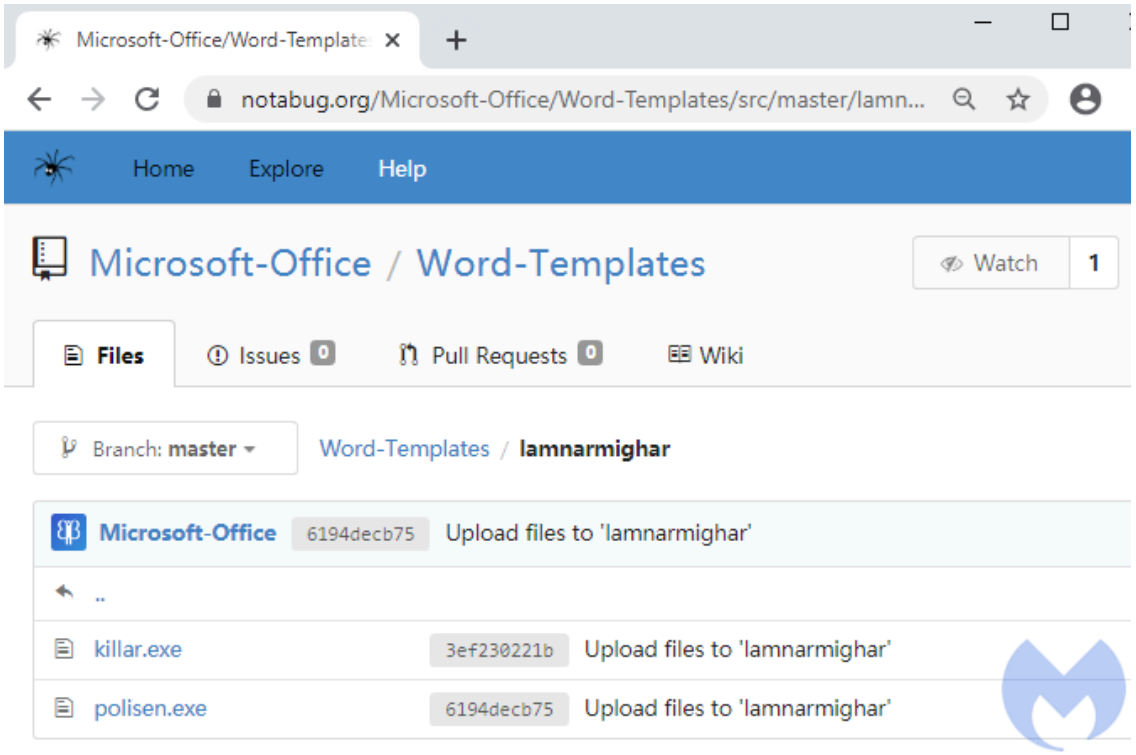
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



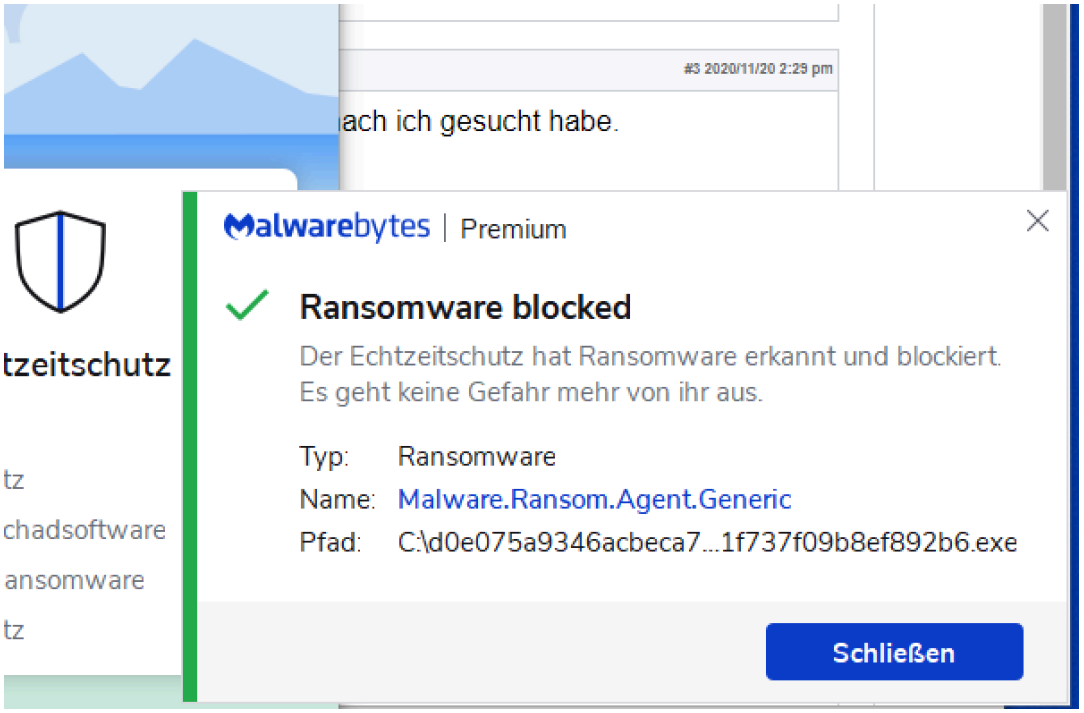
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

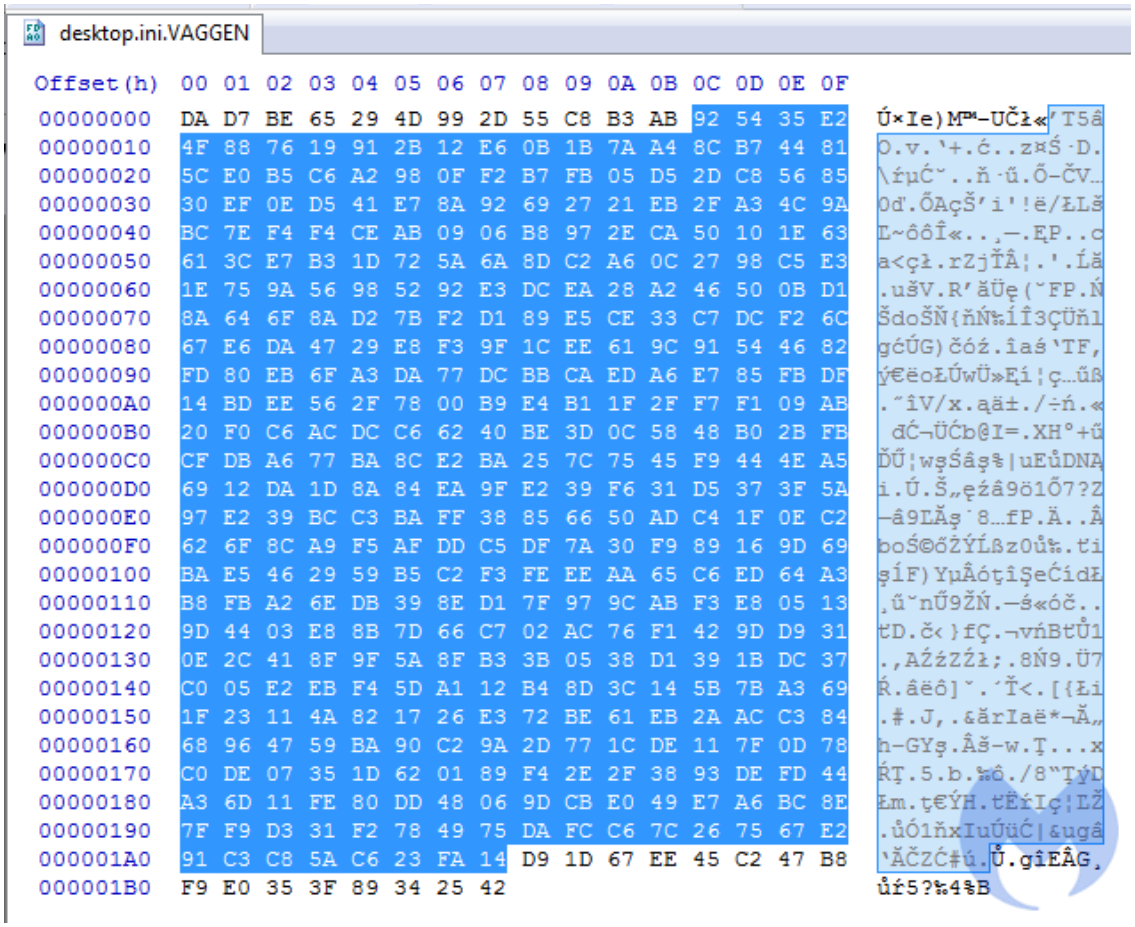
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

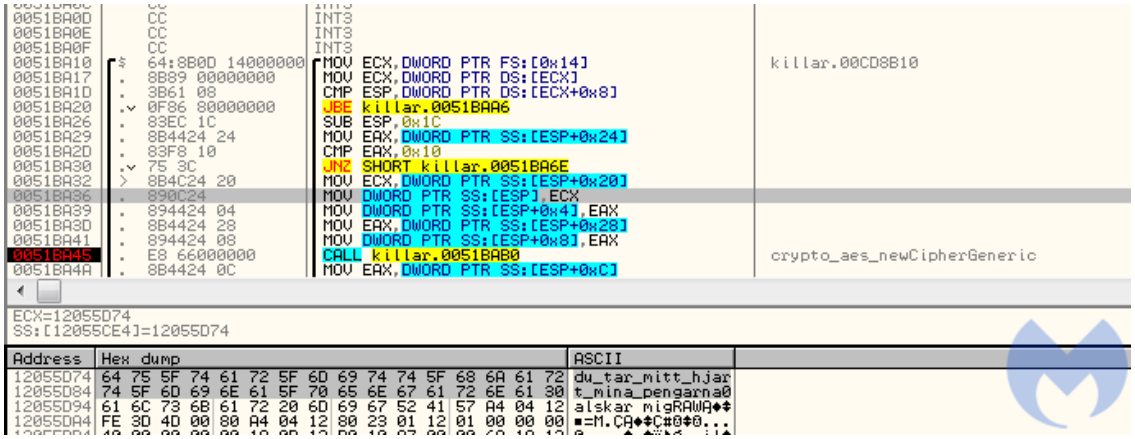


The content of the output file (with .VAGGEN extension) contains:

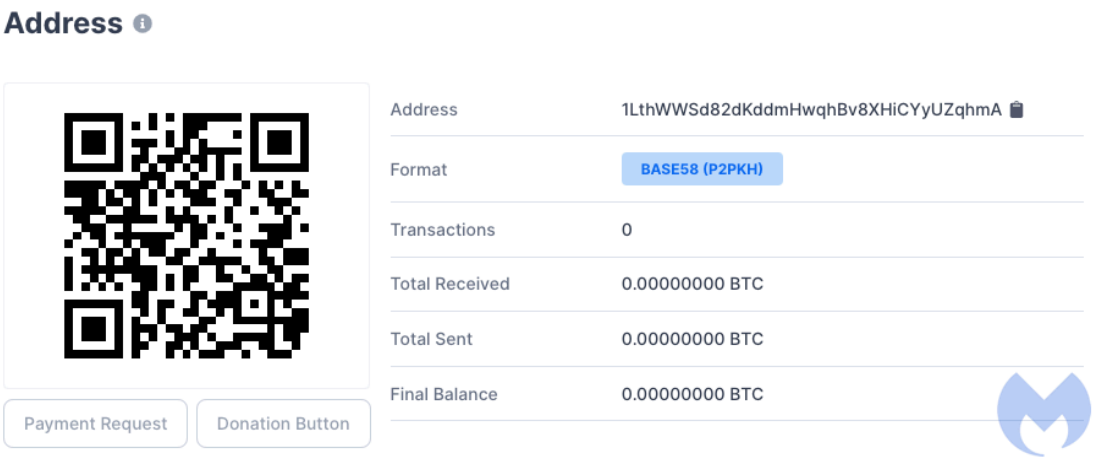
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

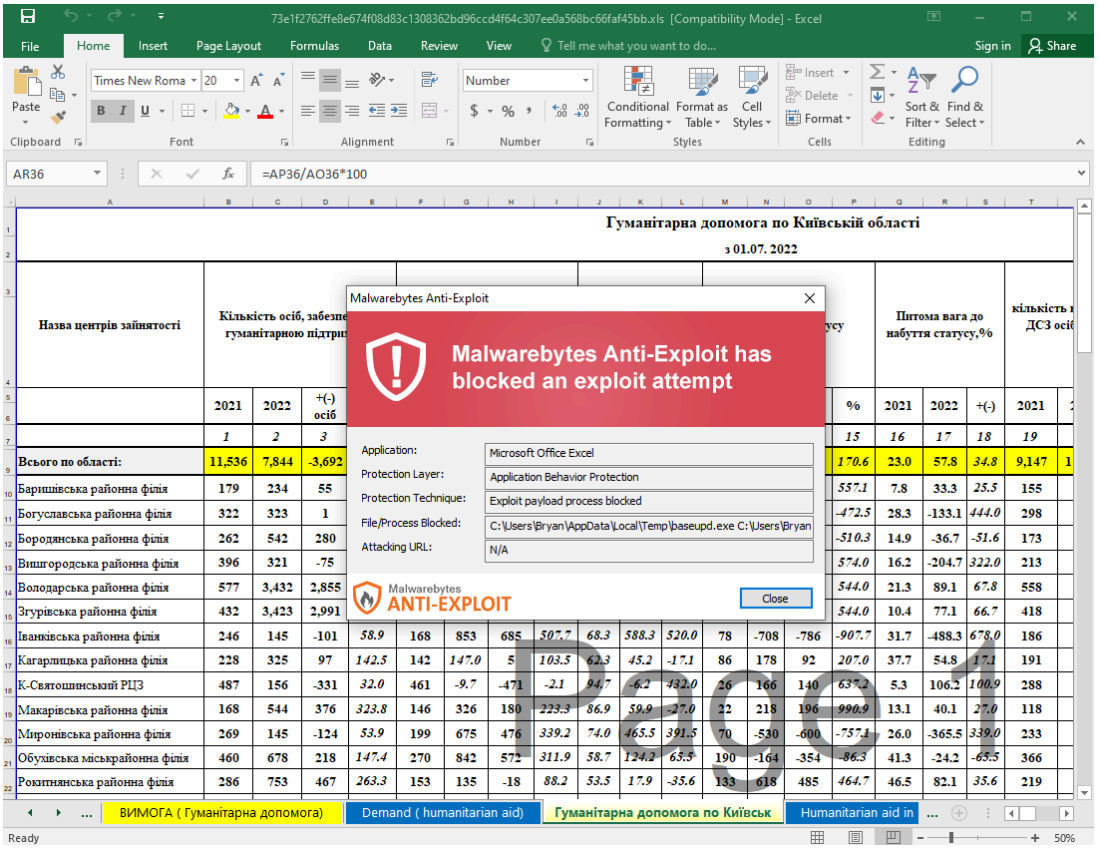
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

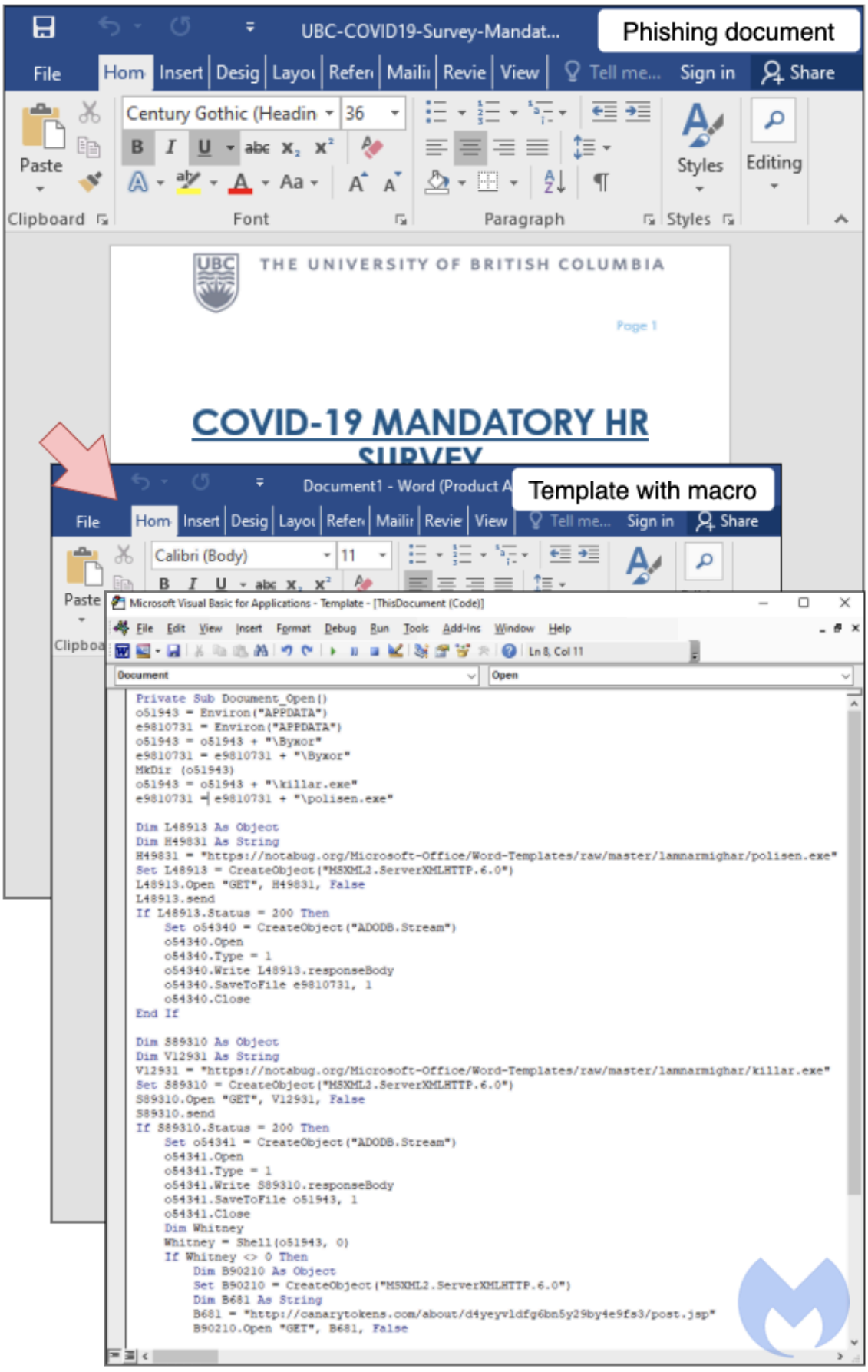
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

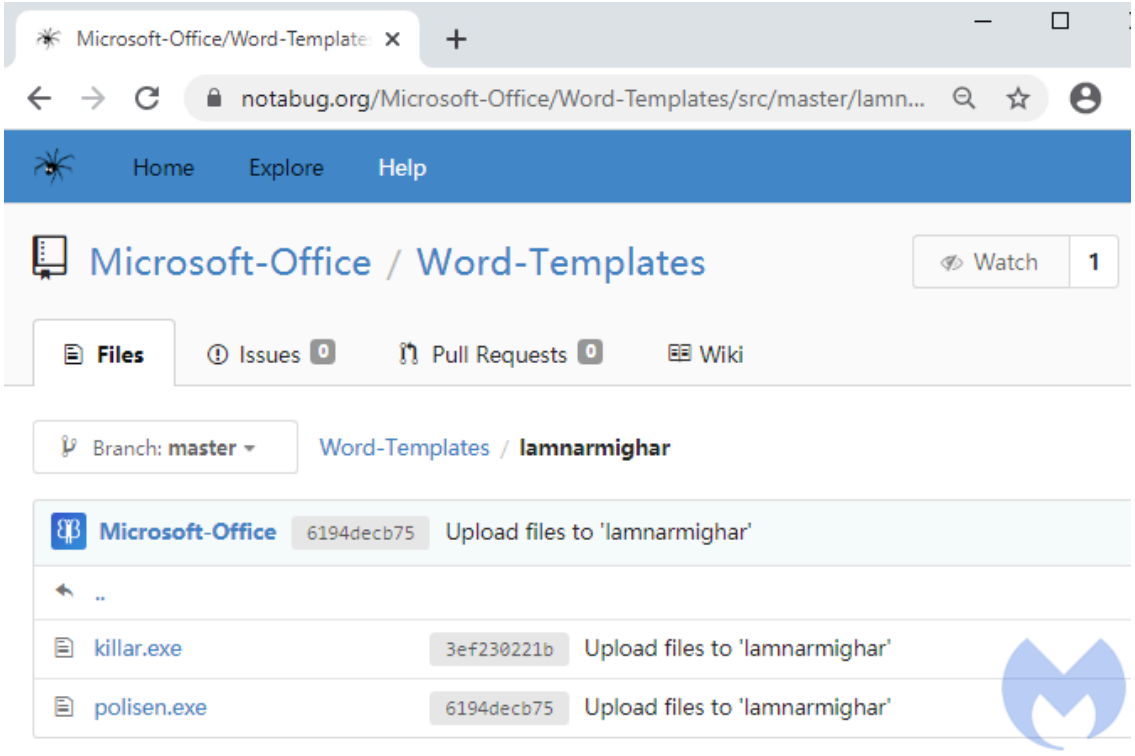
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



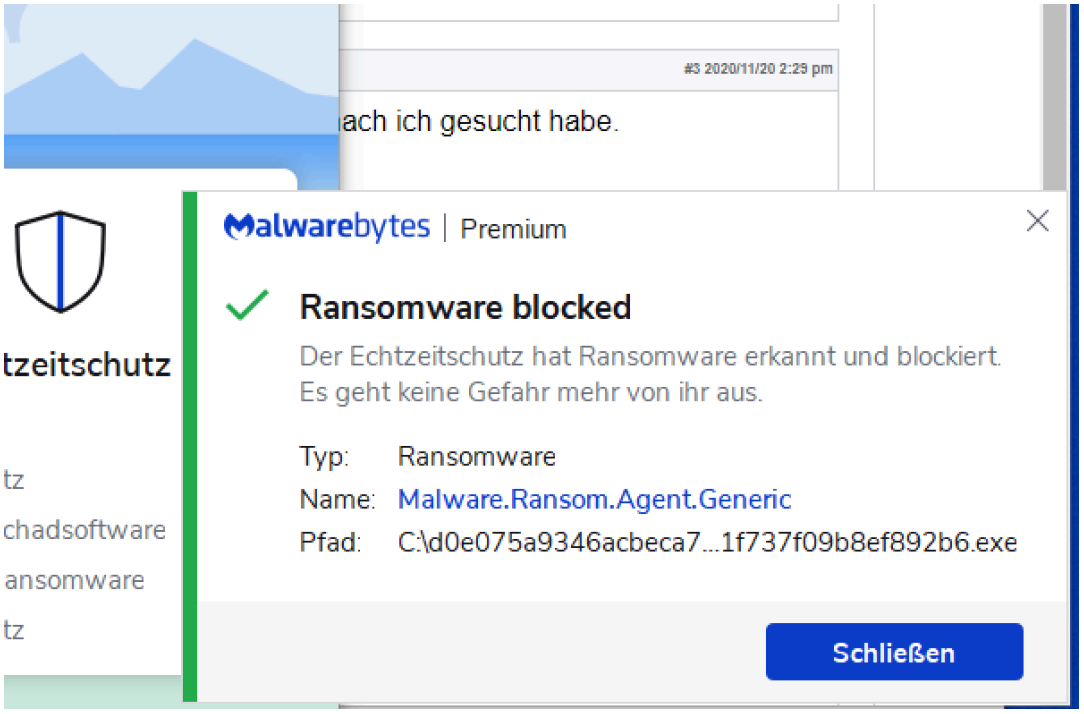
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

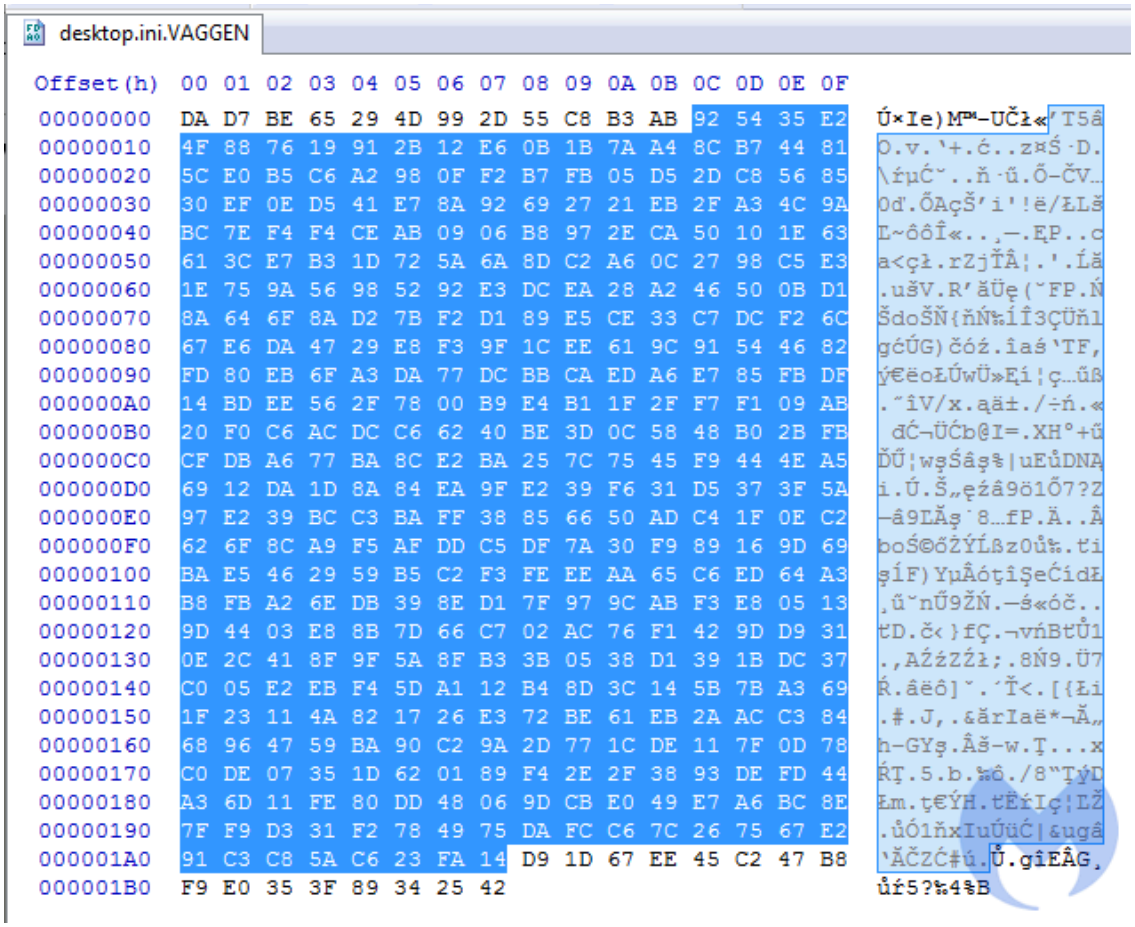
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



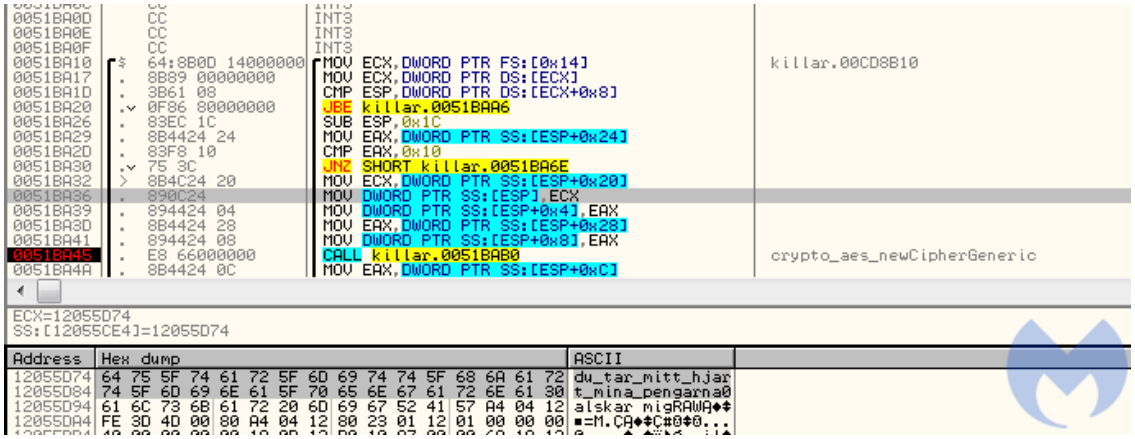
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

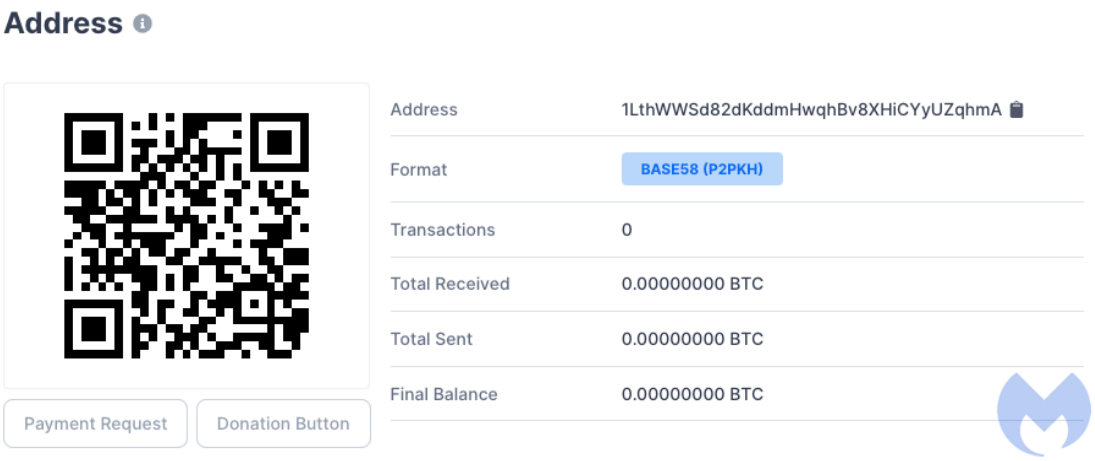
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

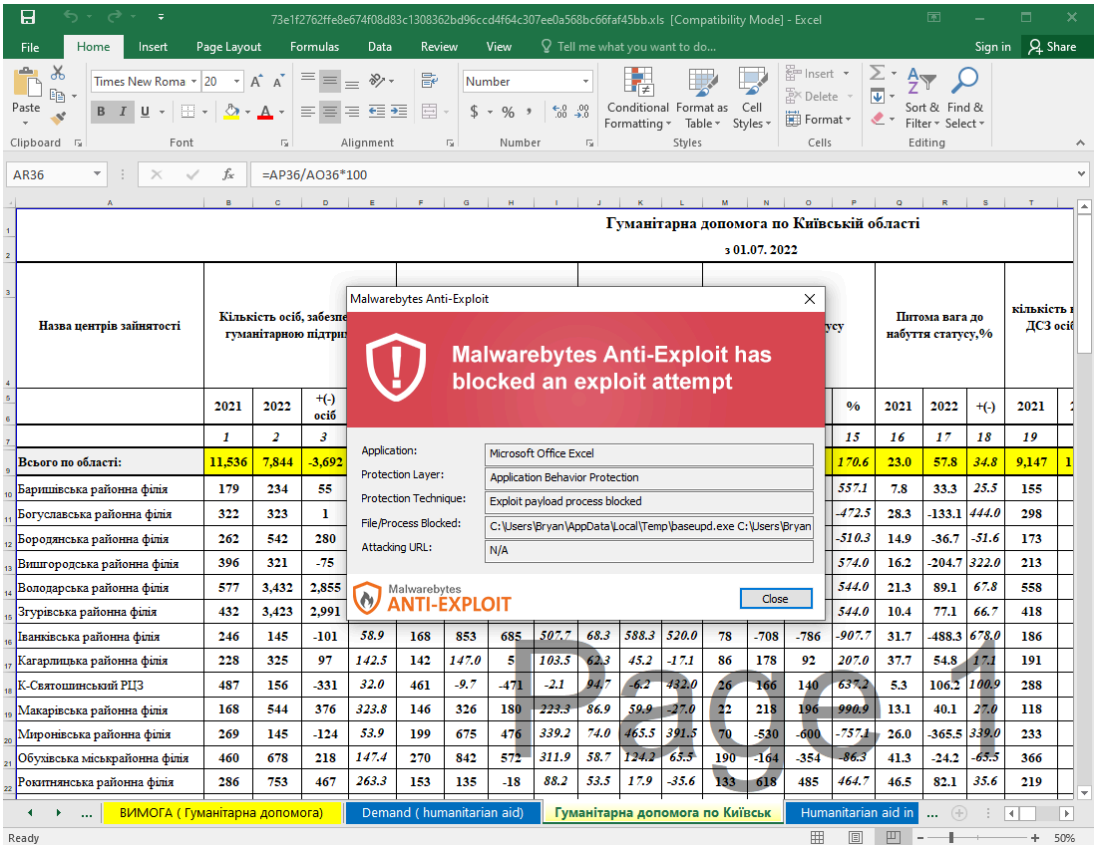
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

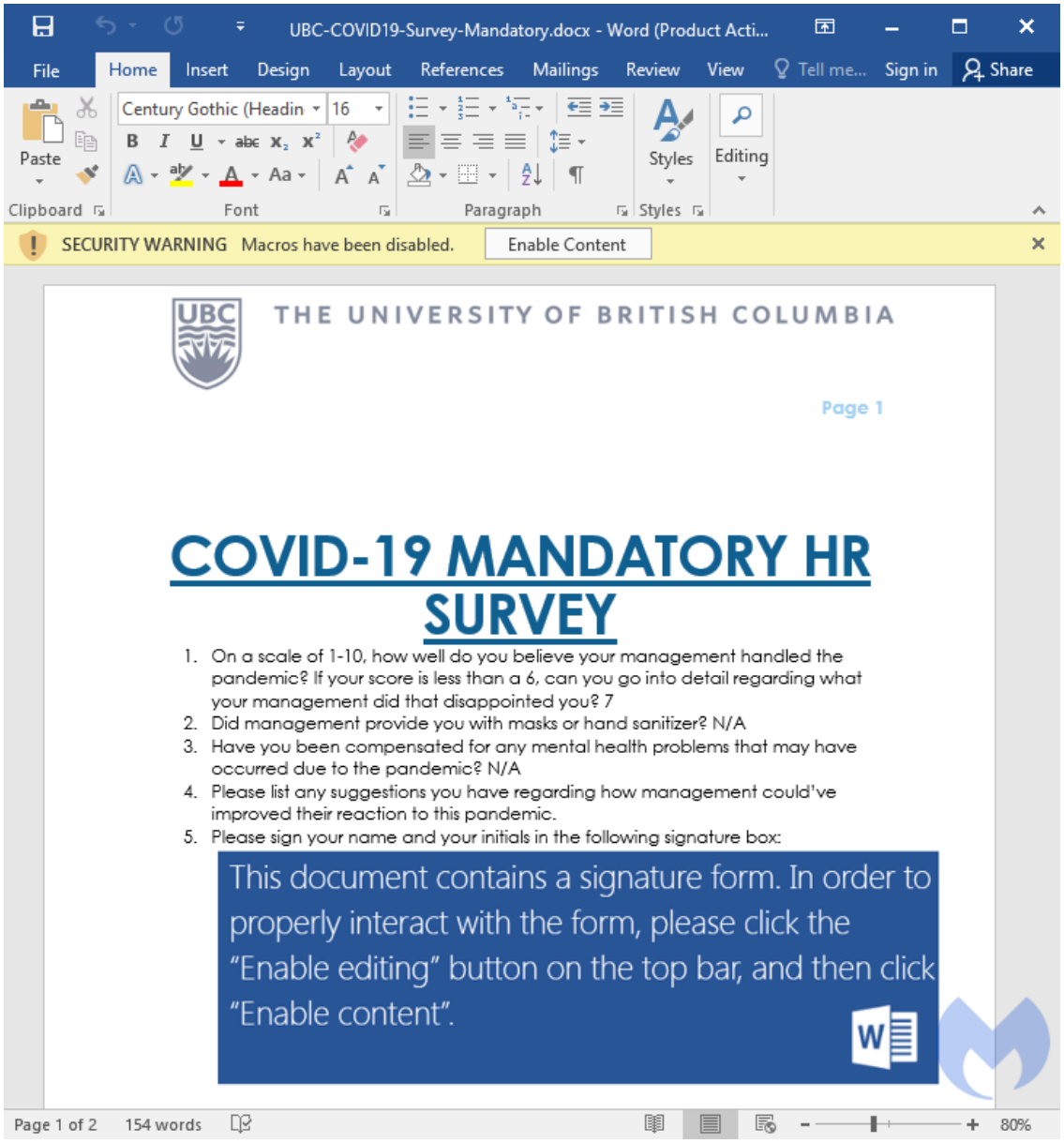
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

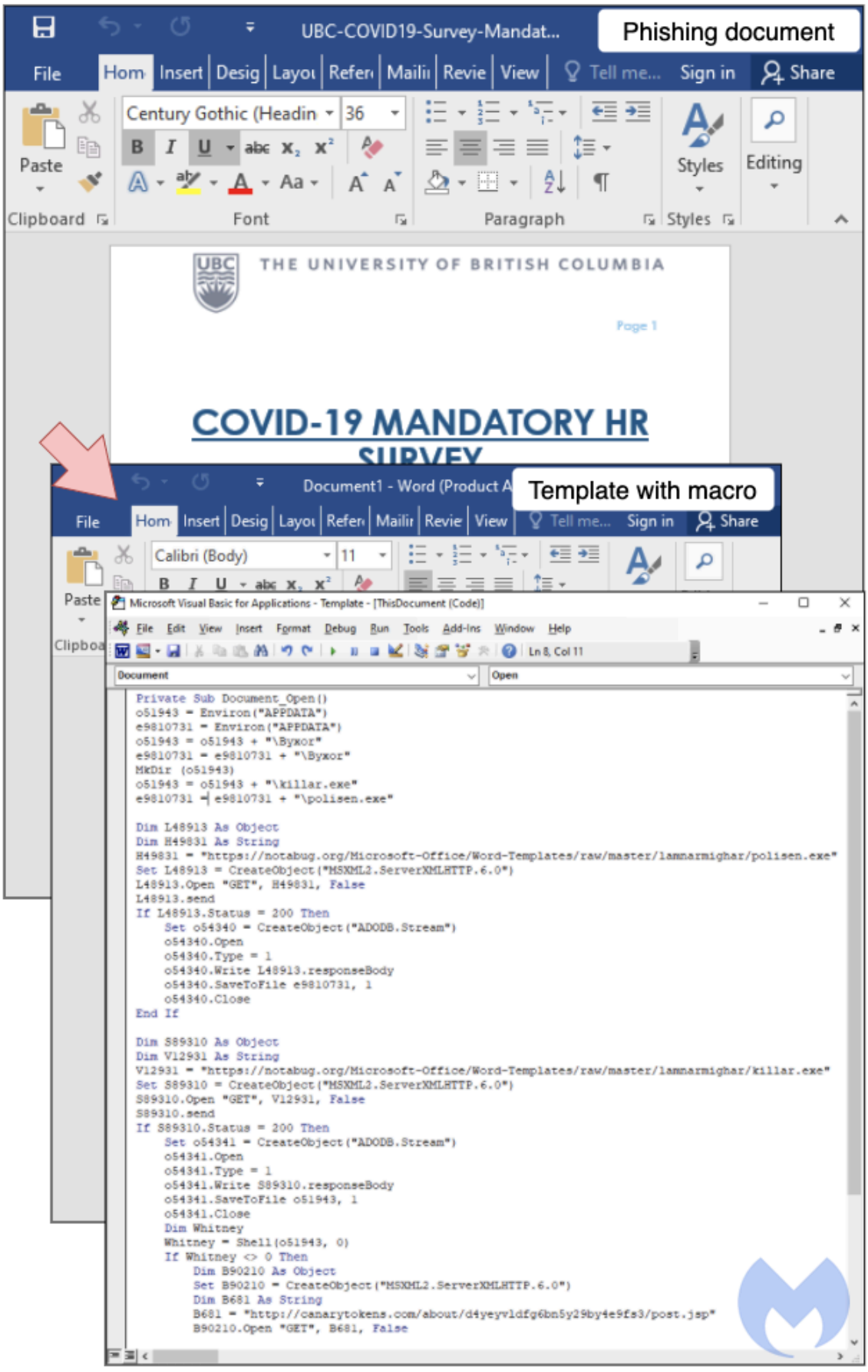
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

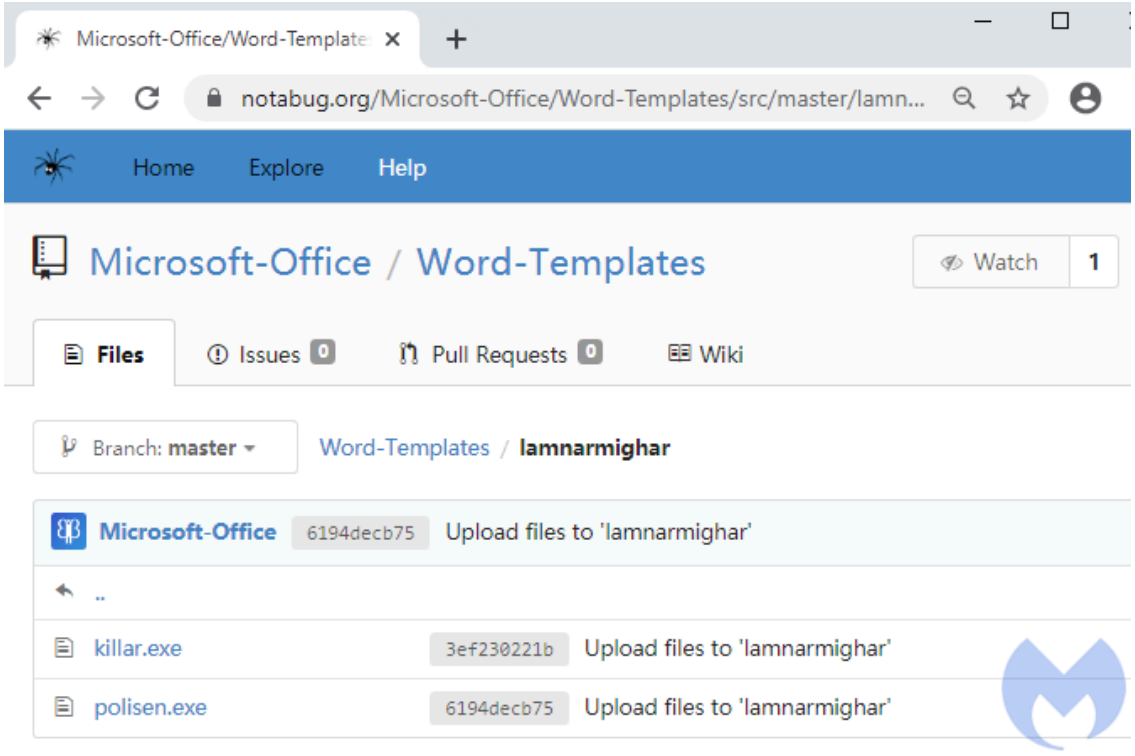
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



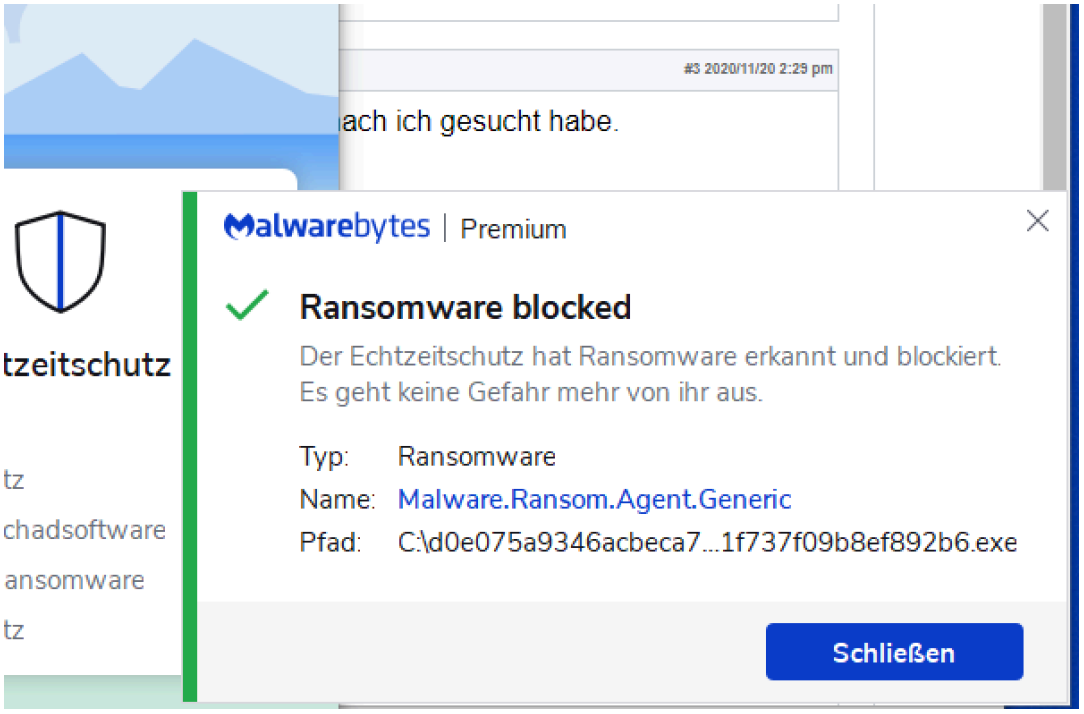
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

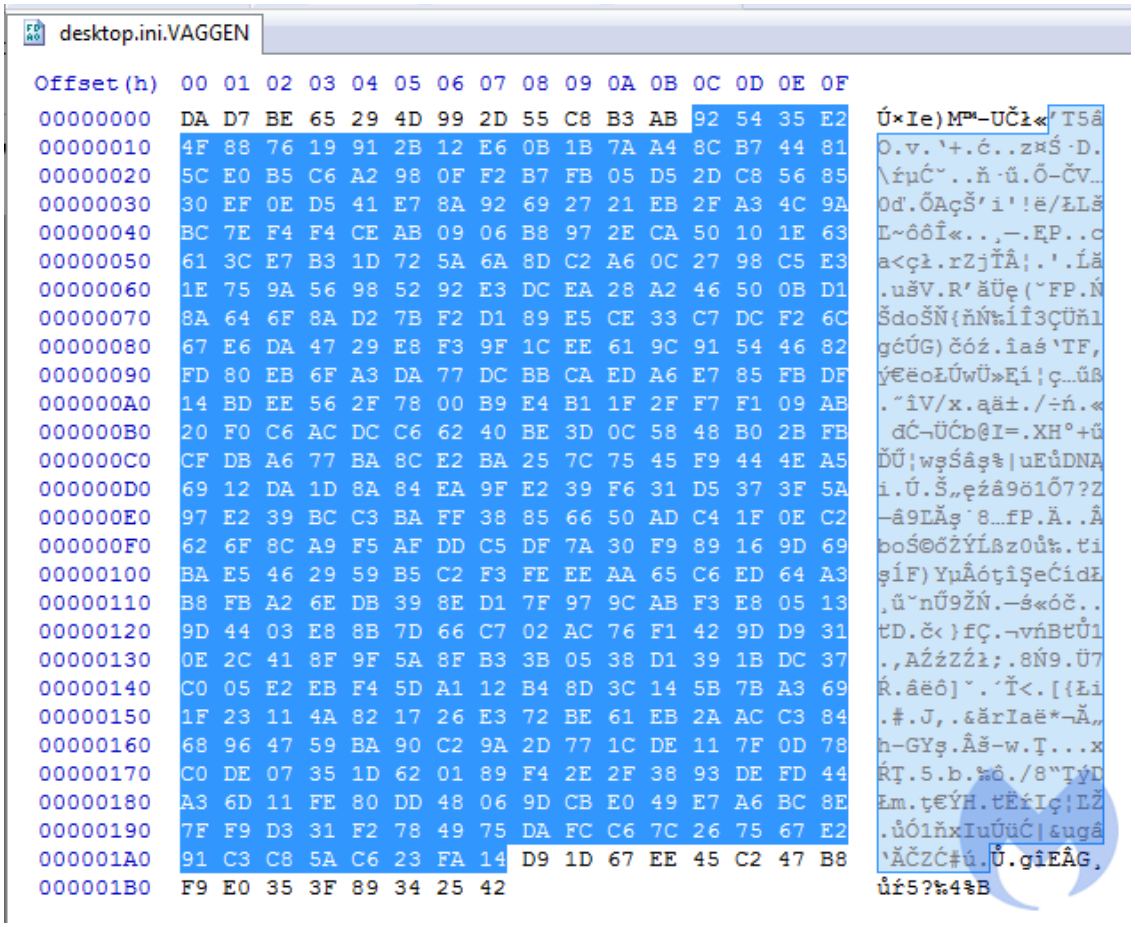
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

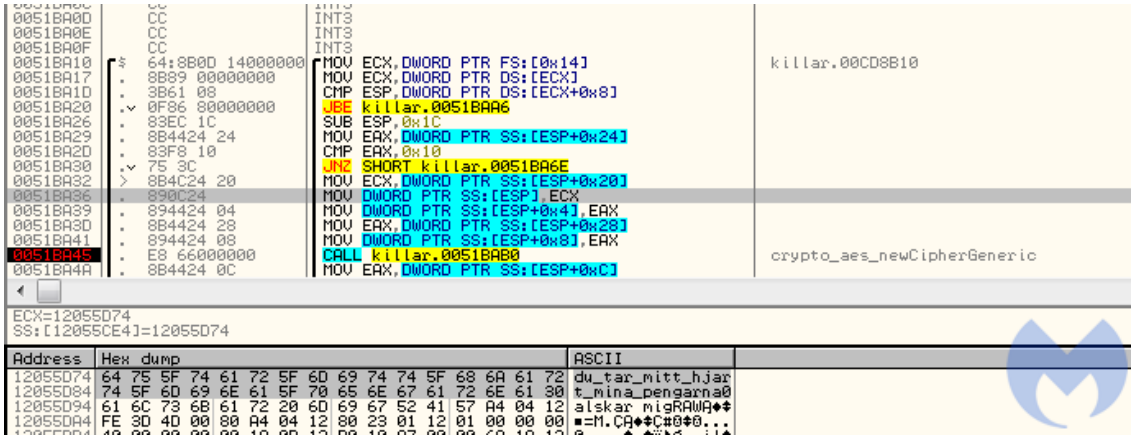


The content of the output file (with .VAGGEN extension) contains:

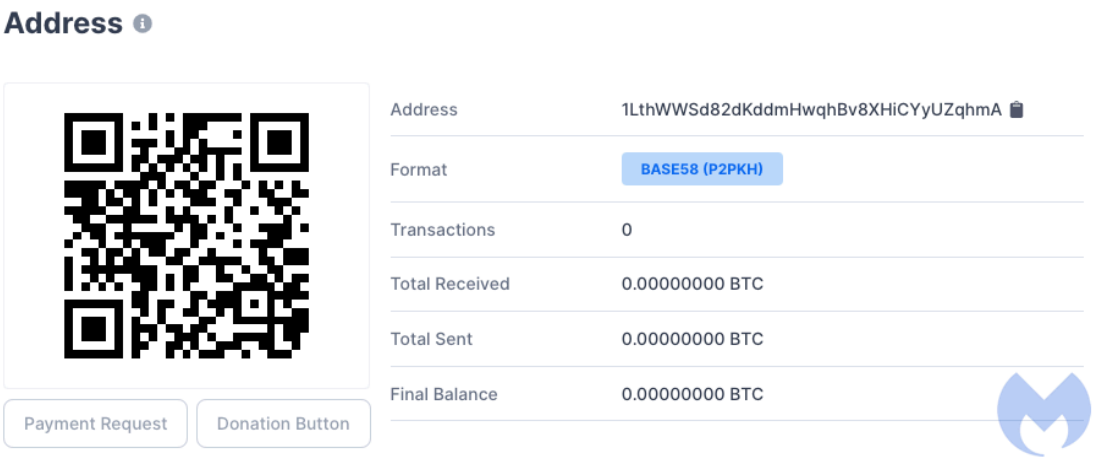
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

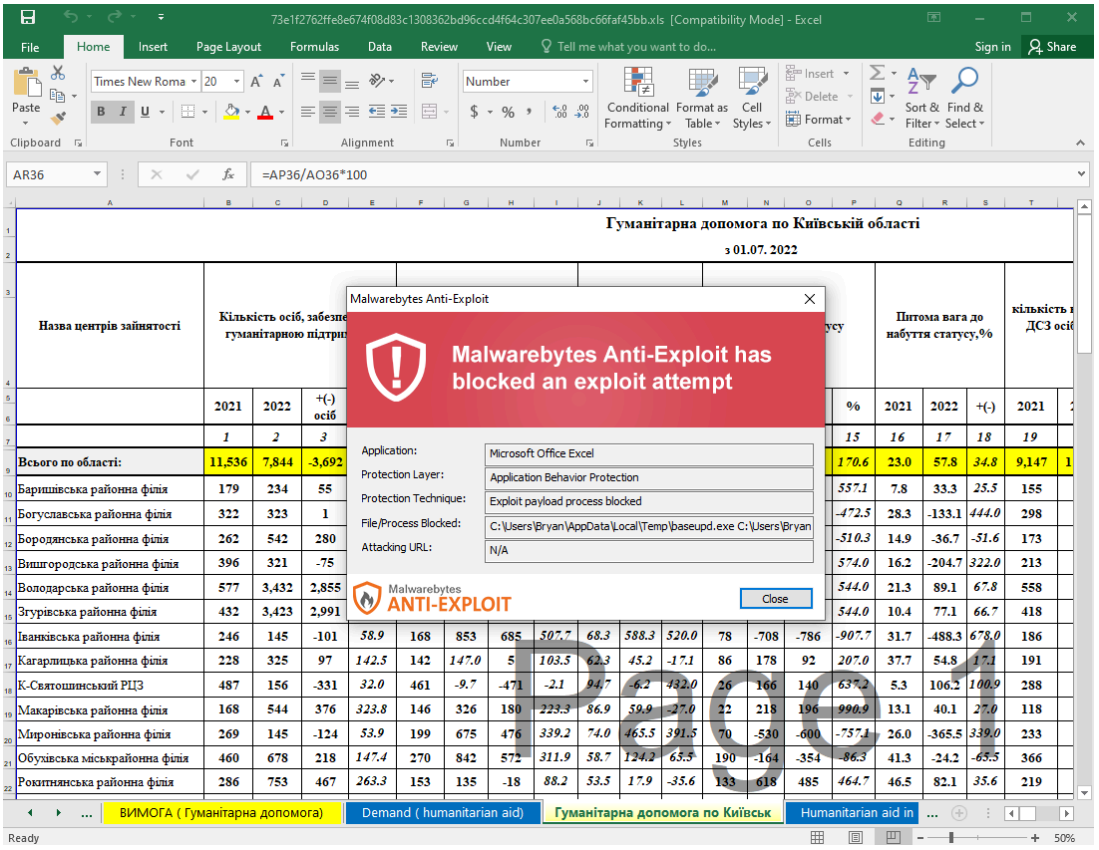
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

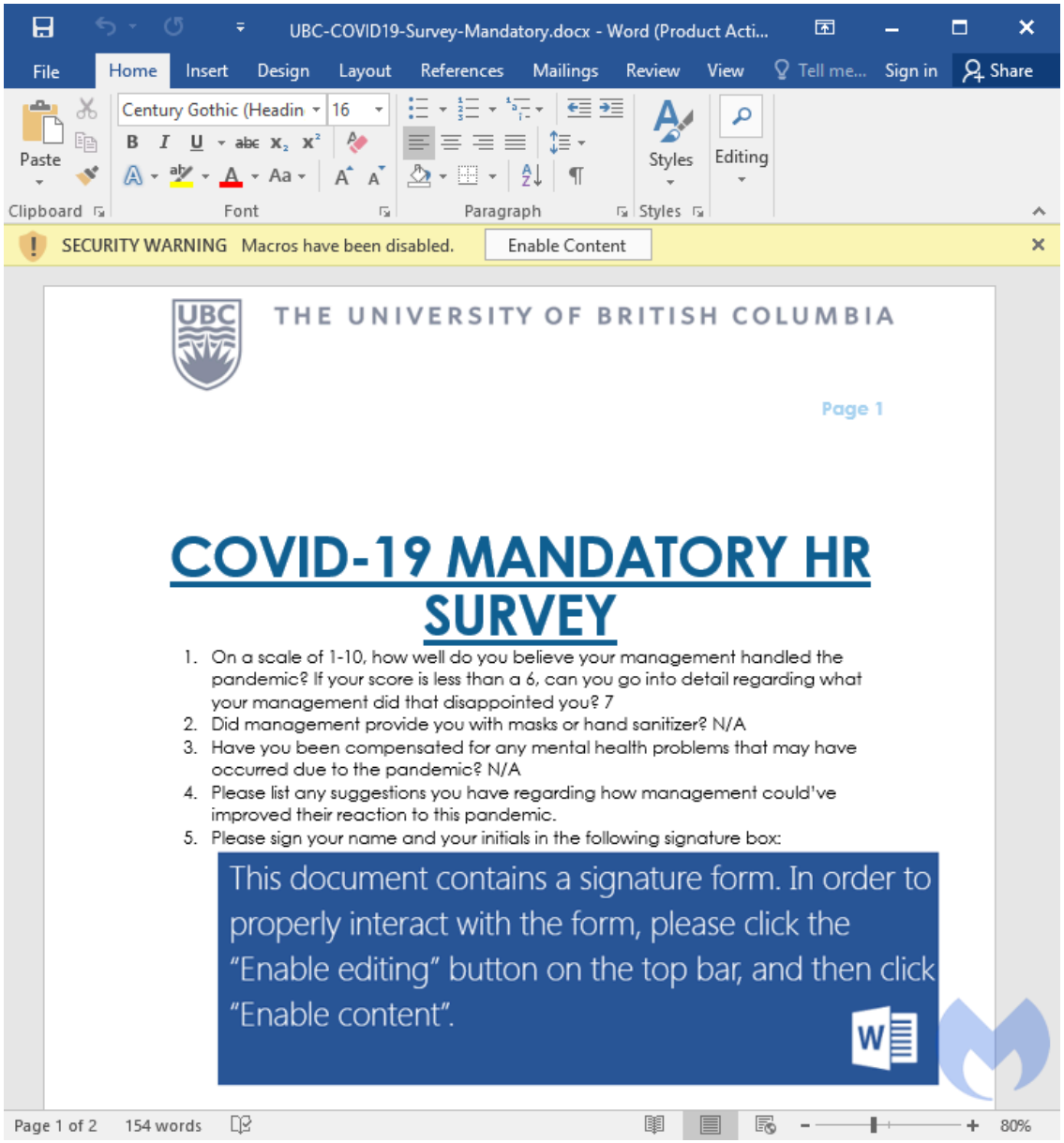
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

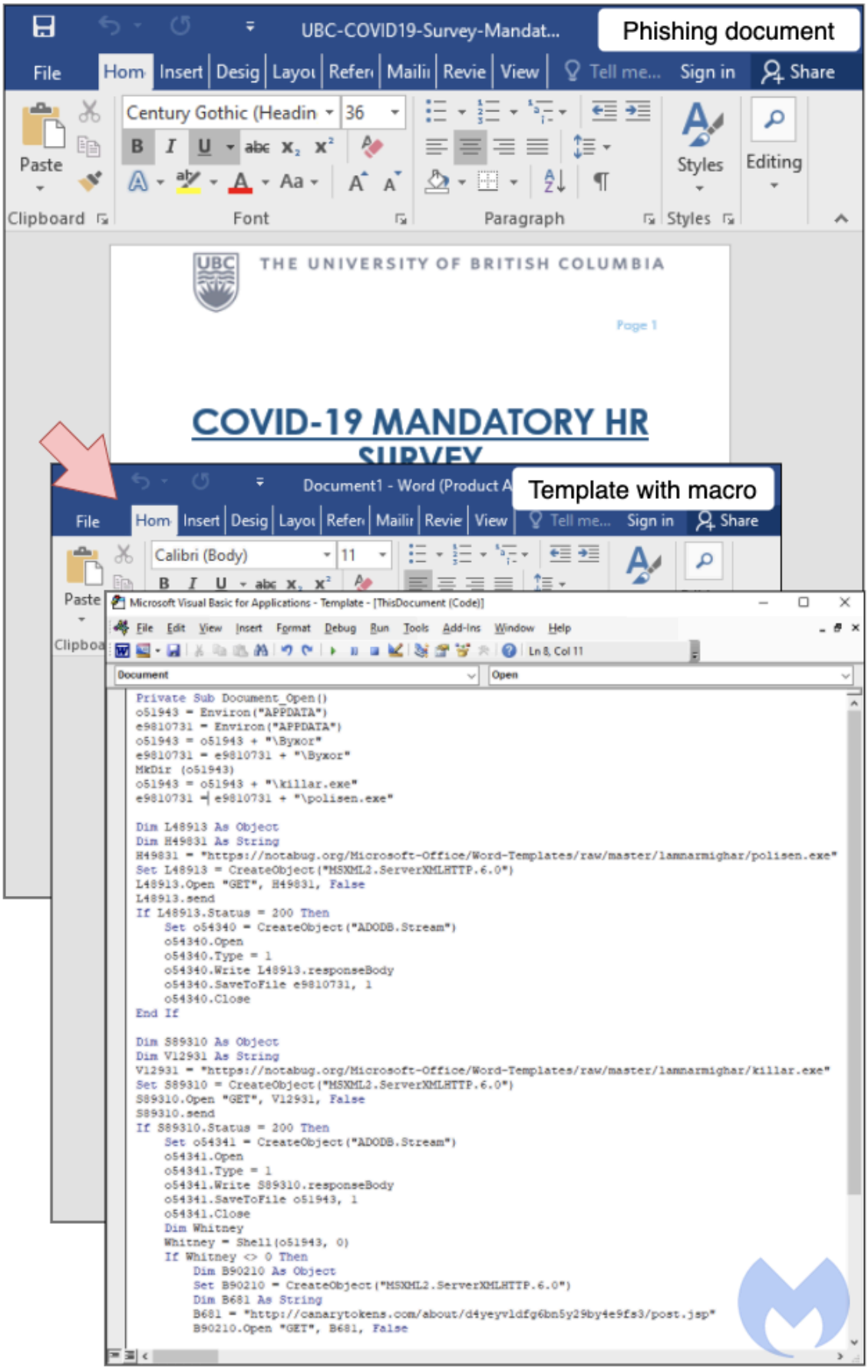
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

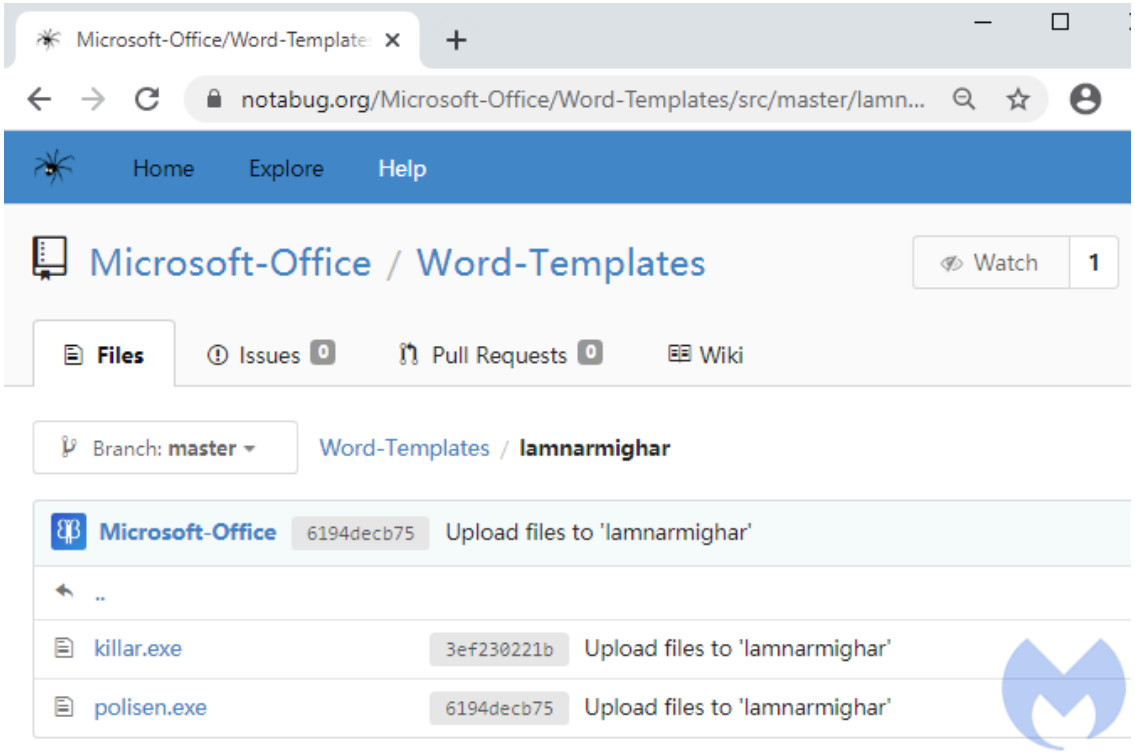
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



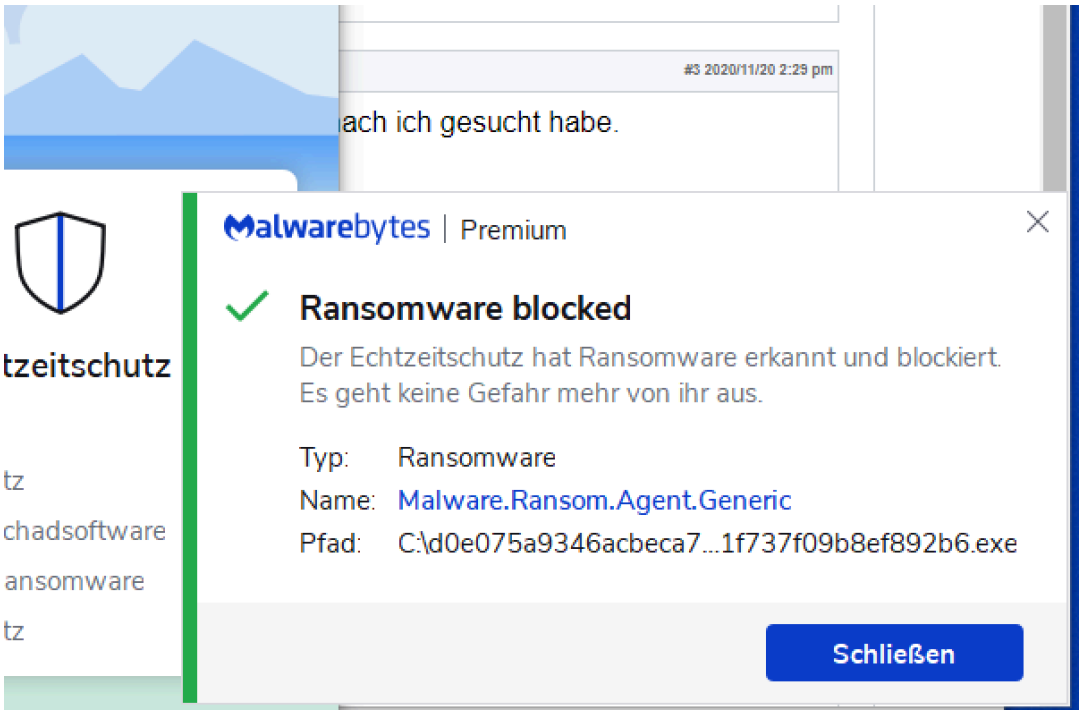
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

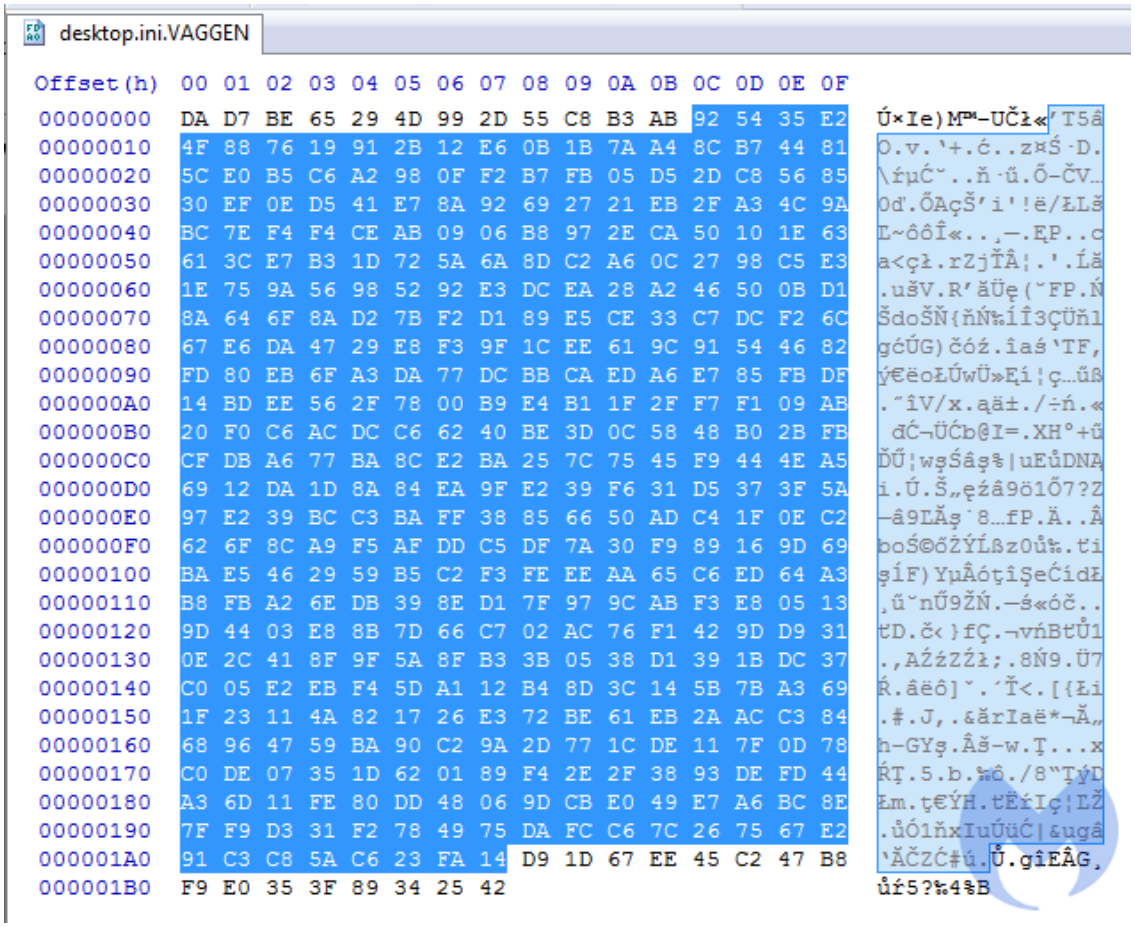
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

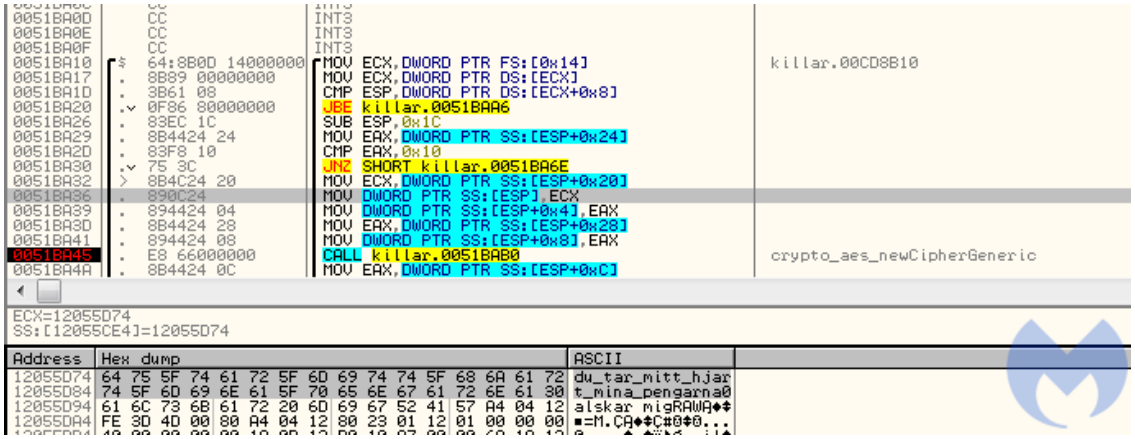


The content of the output file (with .VAGGEN extension) contains:

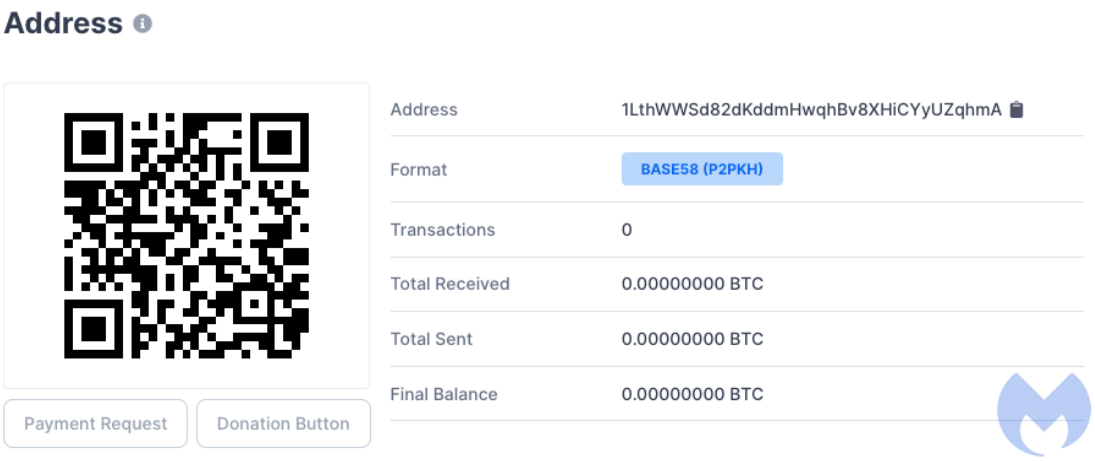
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

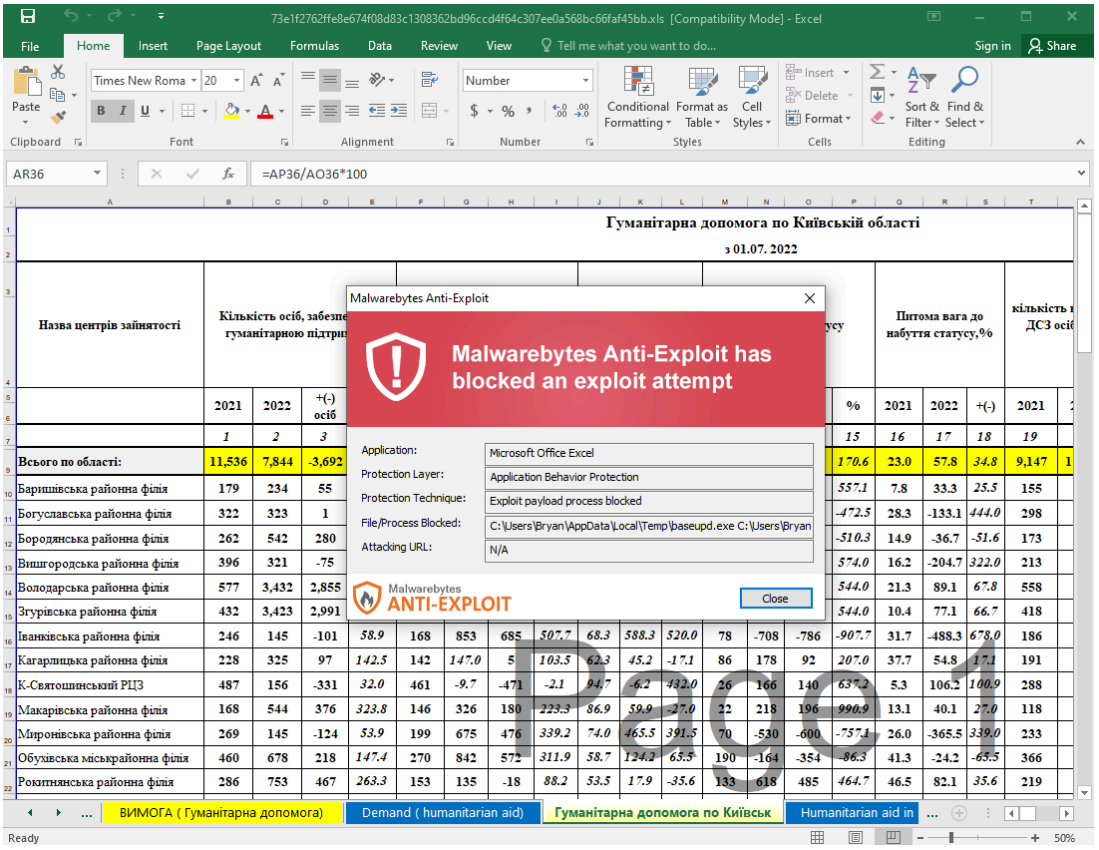
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

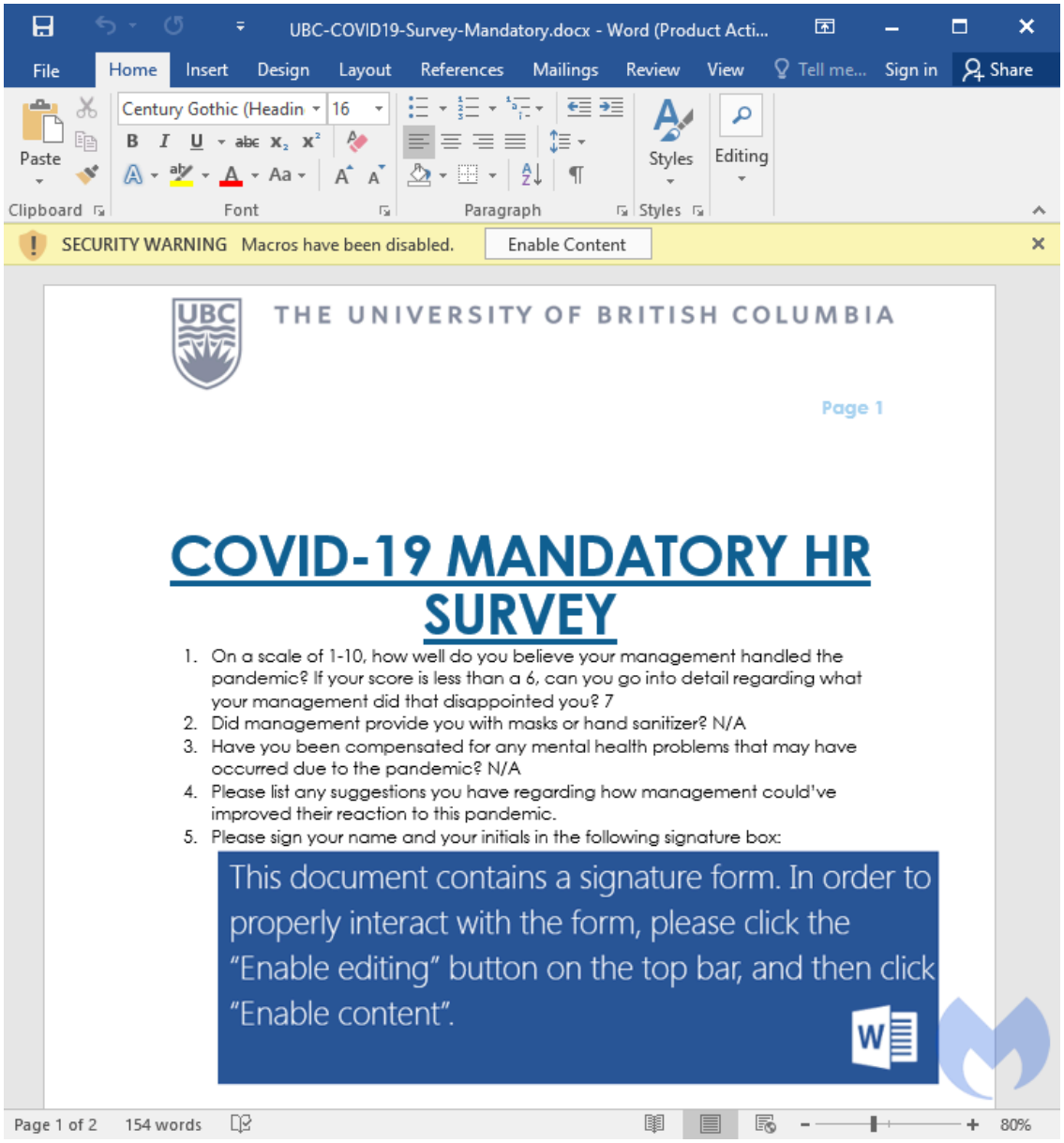
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

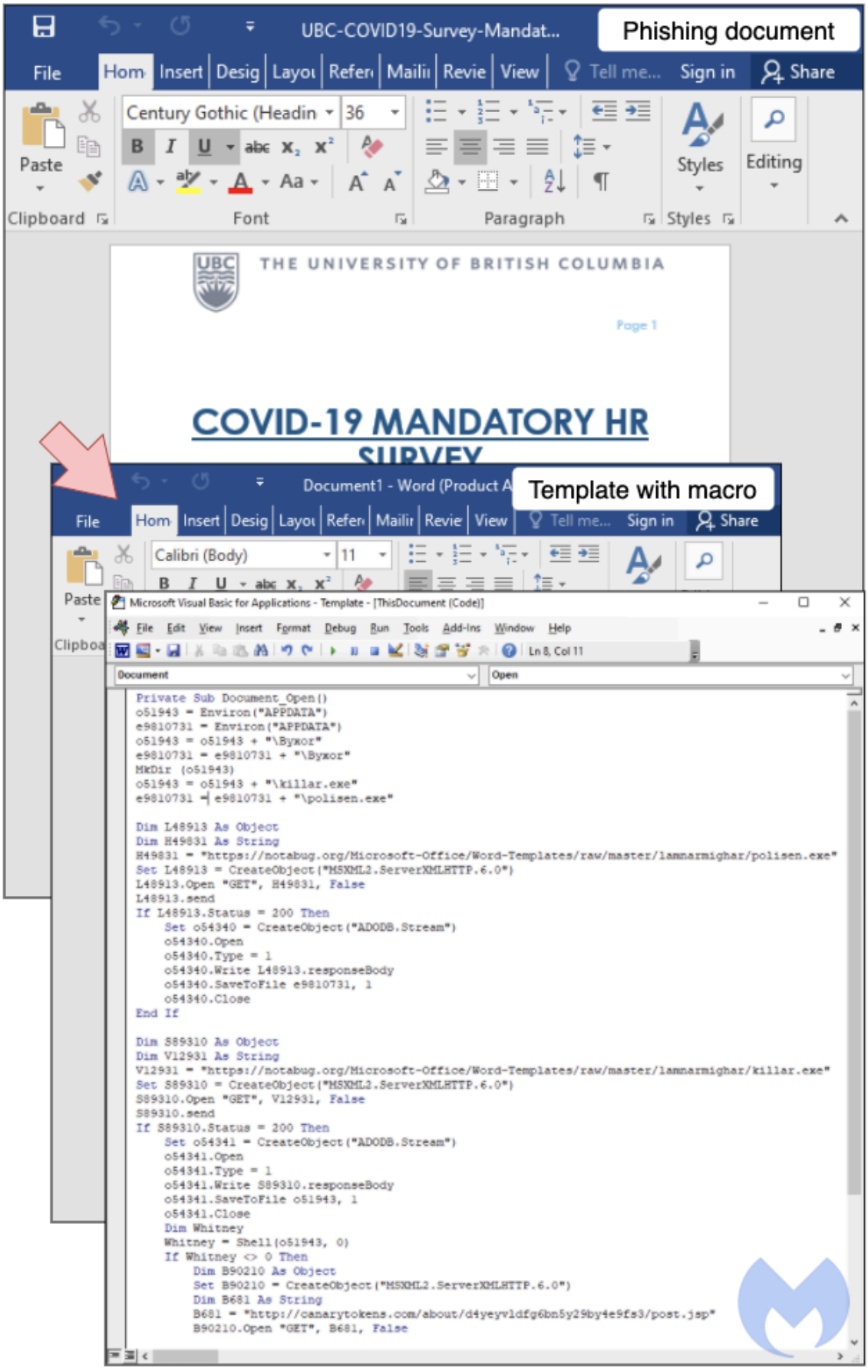
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

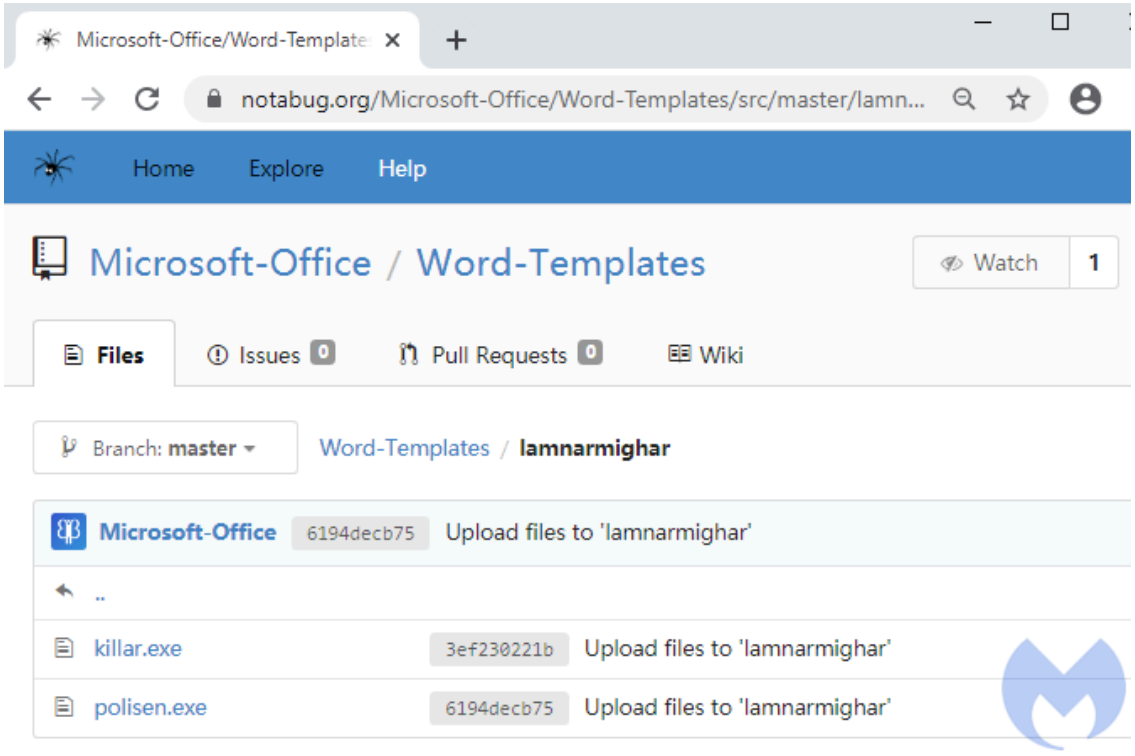
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



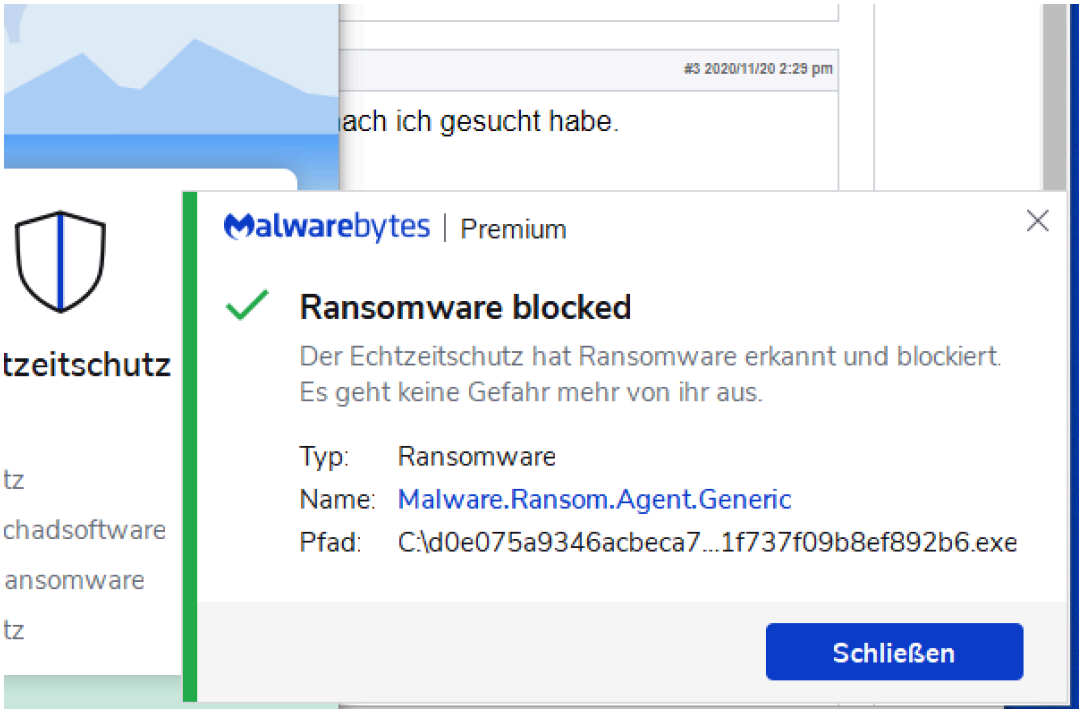
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

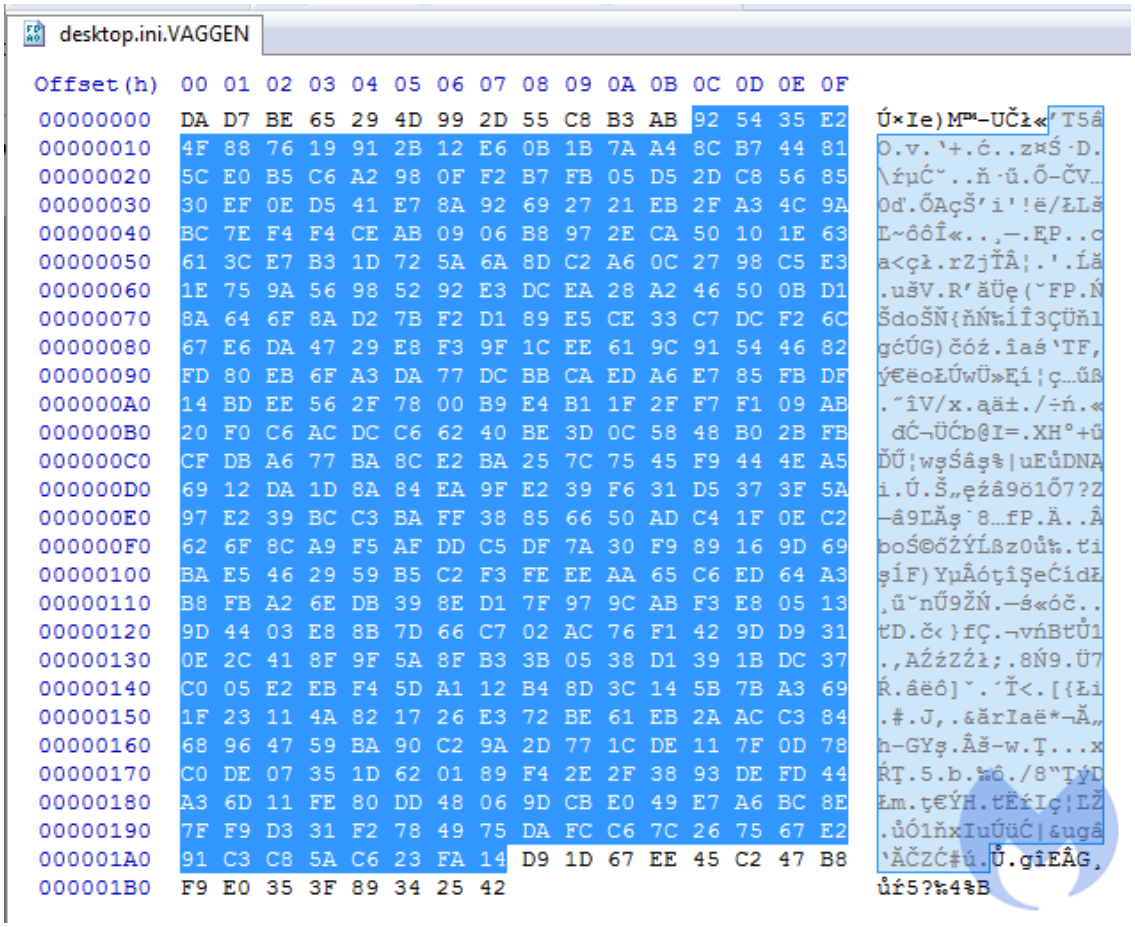
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

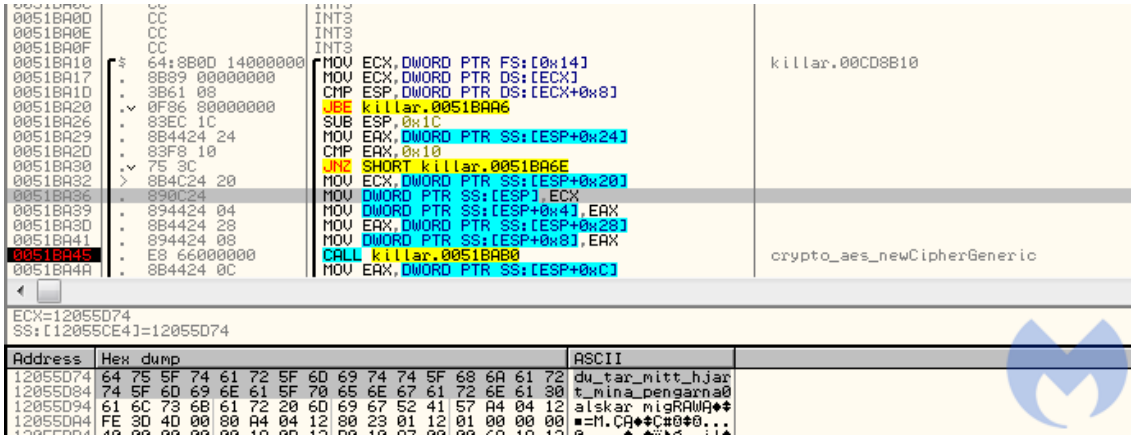


The content of the output file (with .VAGGEN extension) contains:

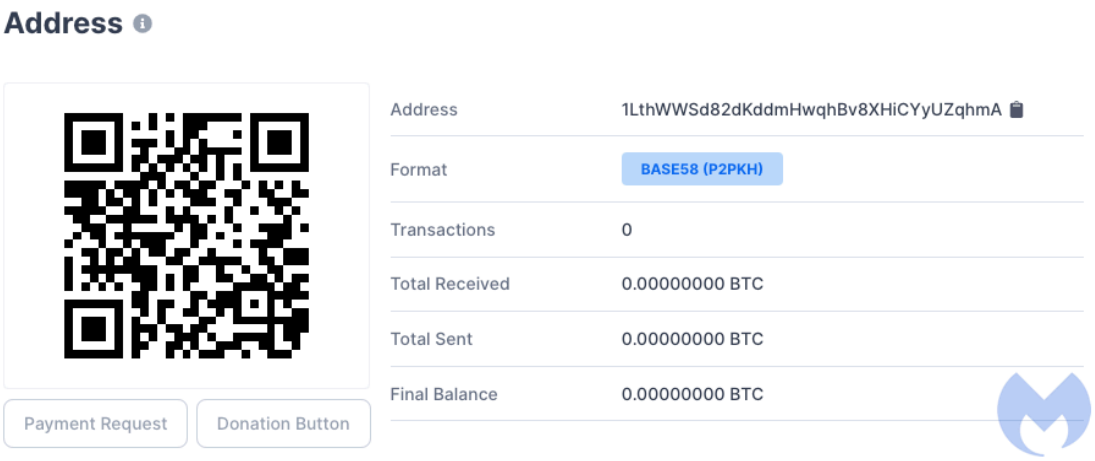
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

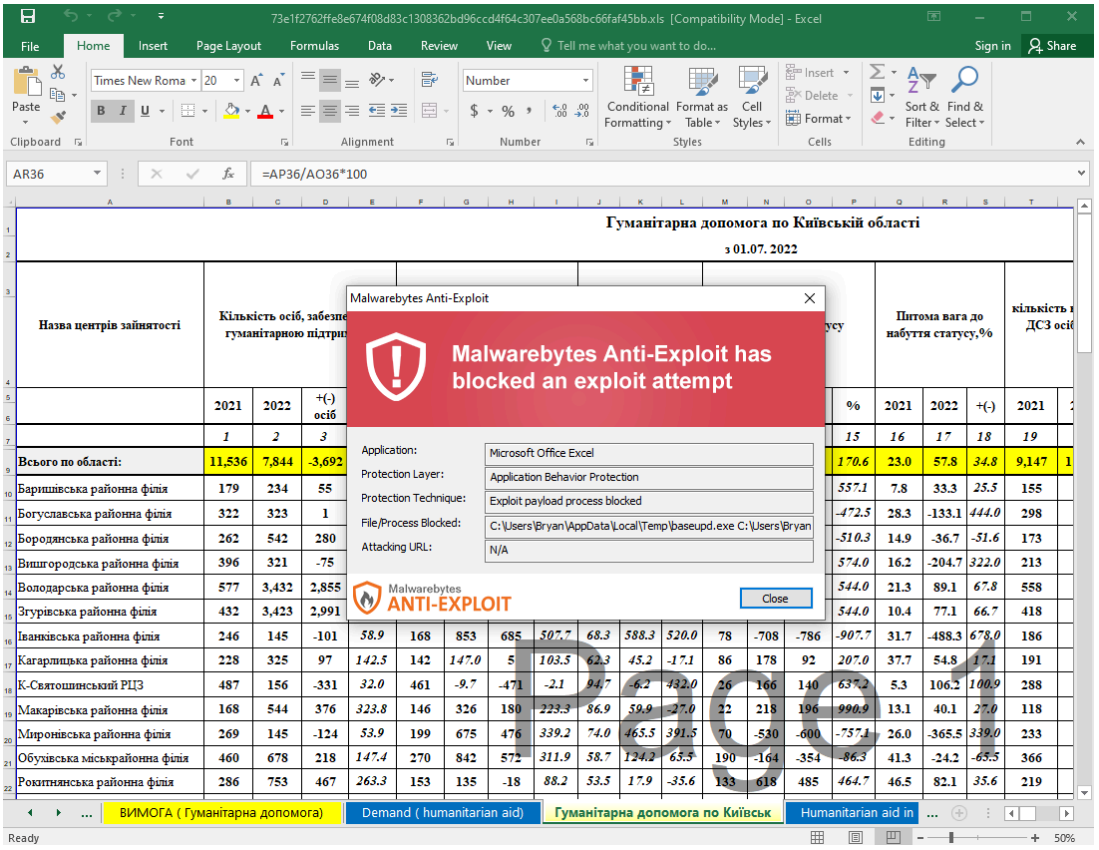
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

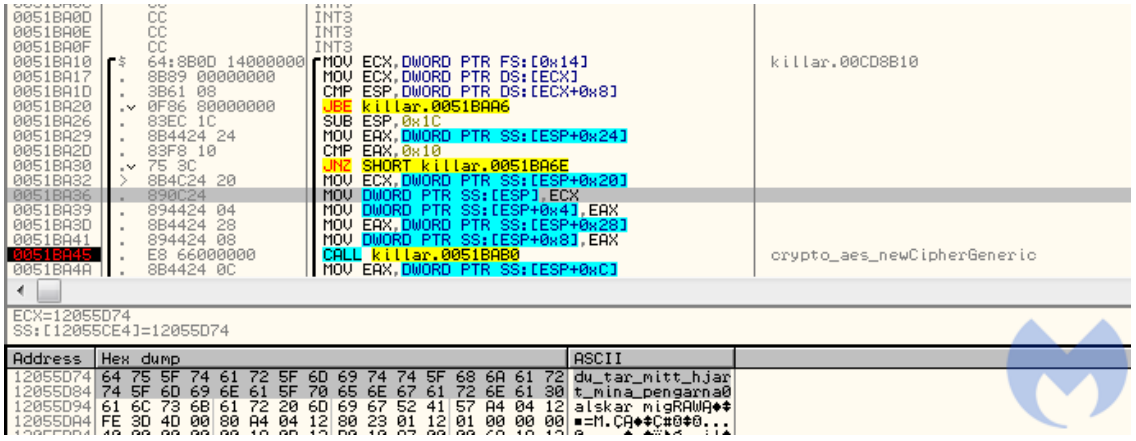
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

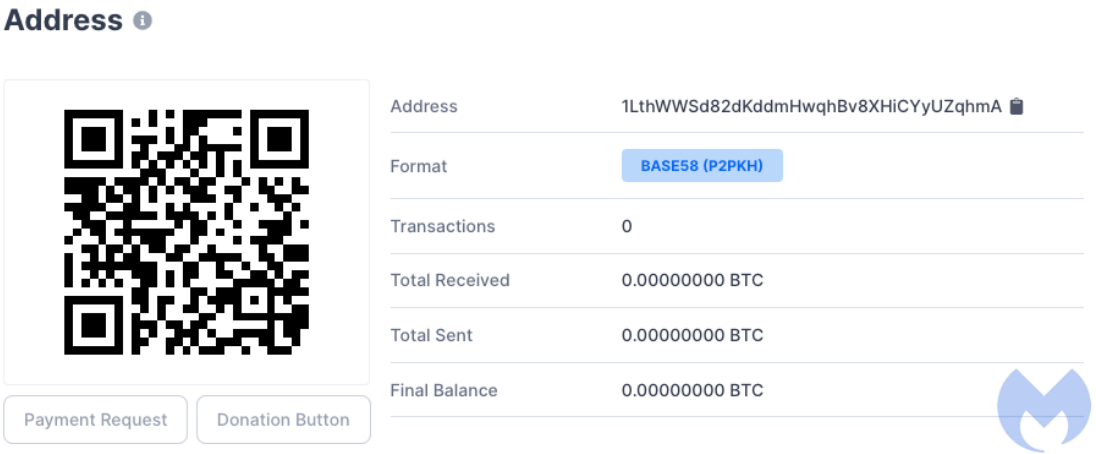
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

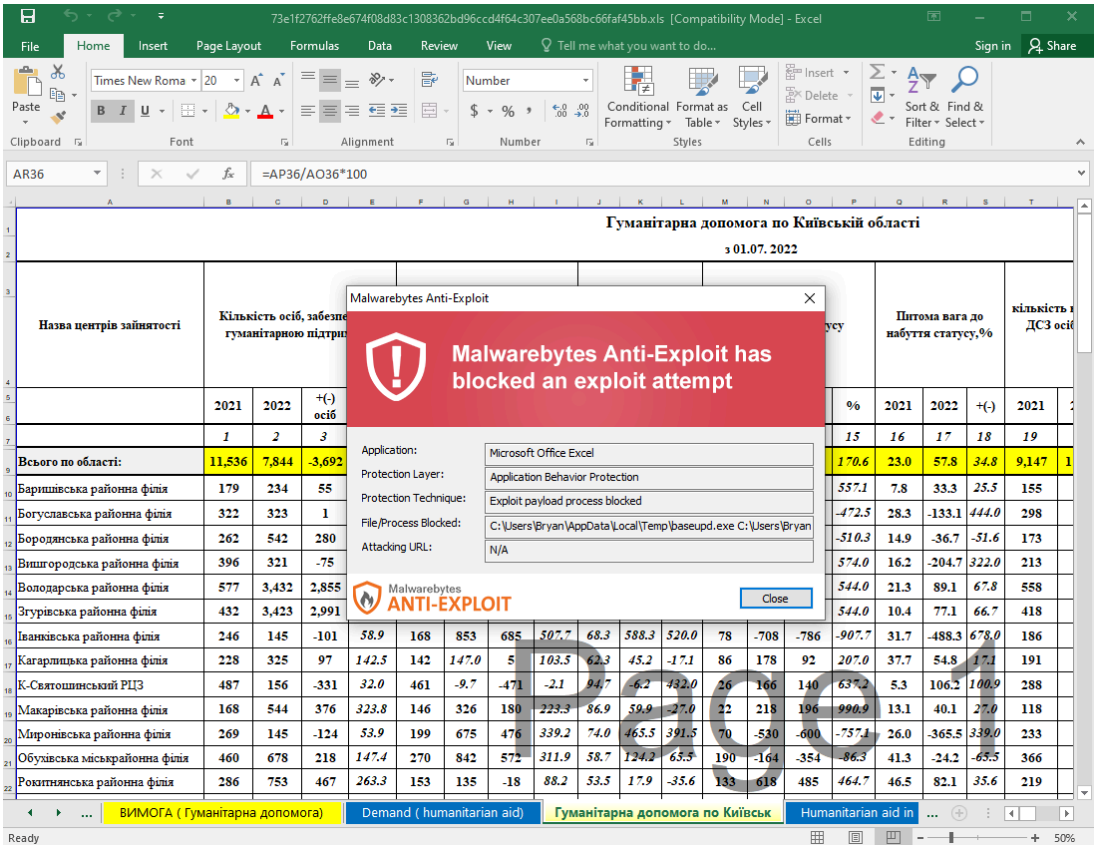
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

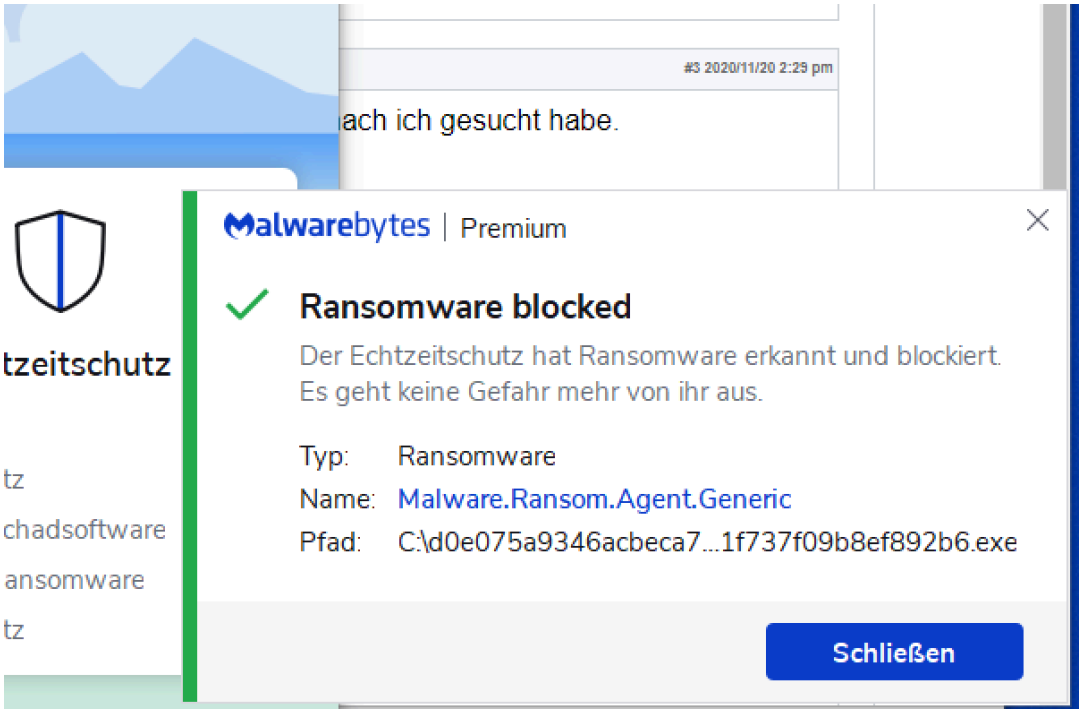
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

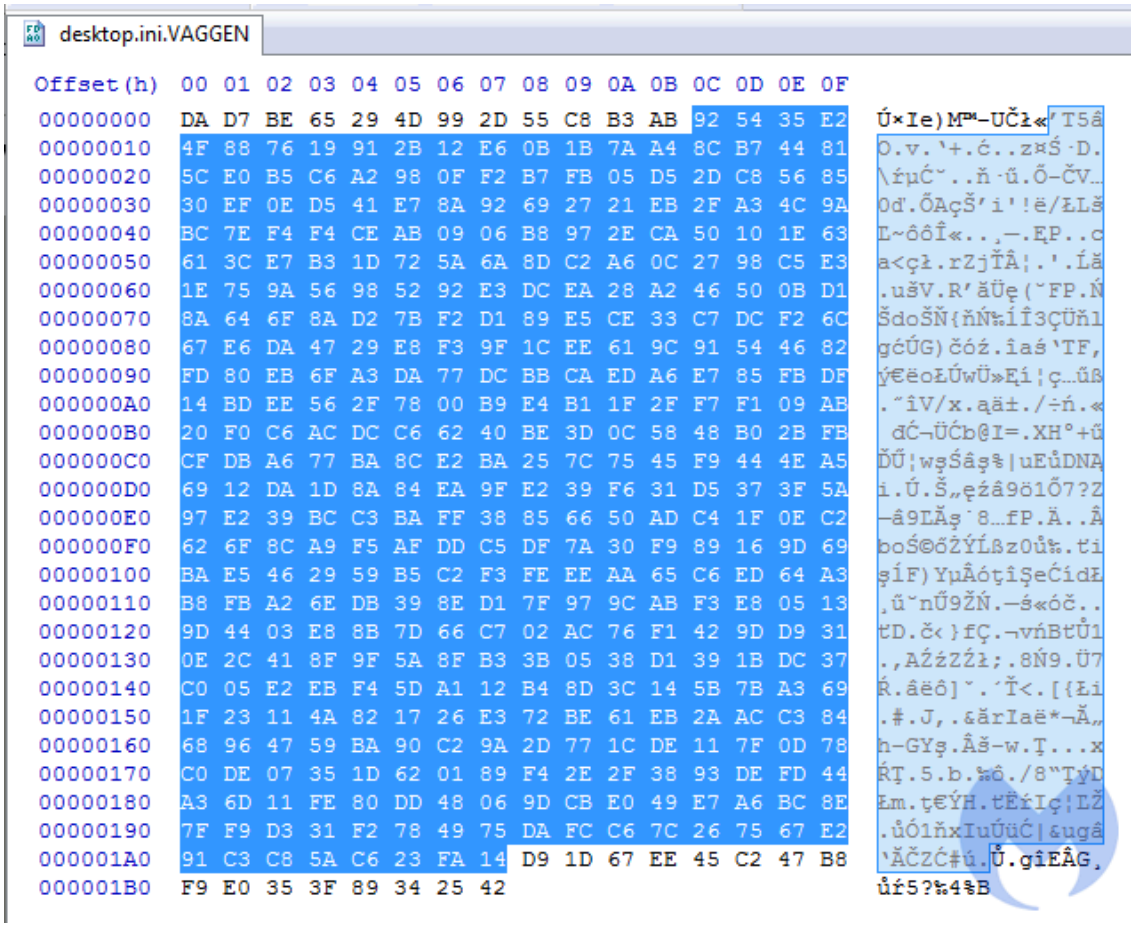
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

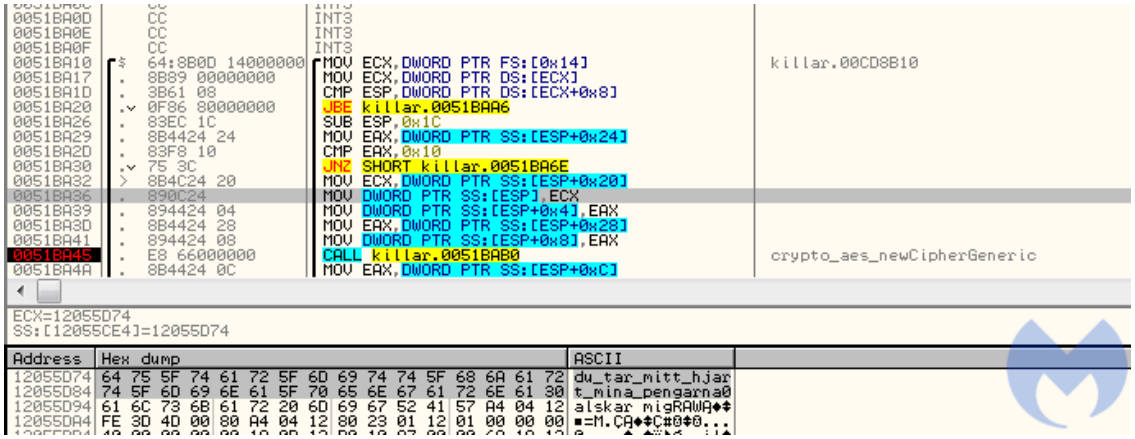


The content of the output file (with .VAGGEN extension) contains:

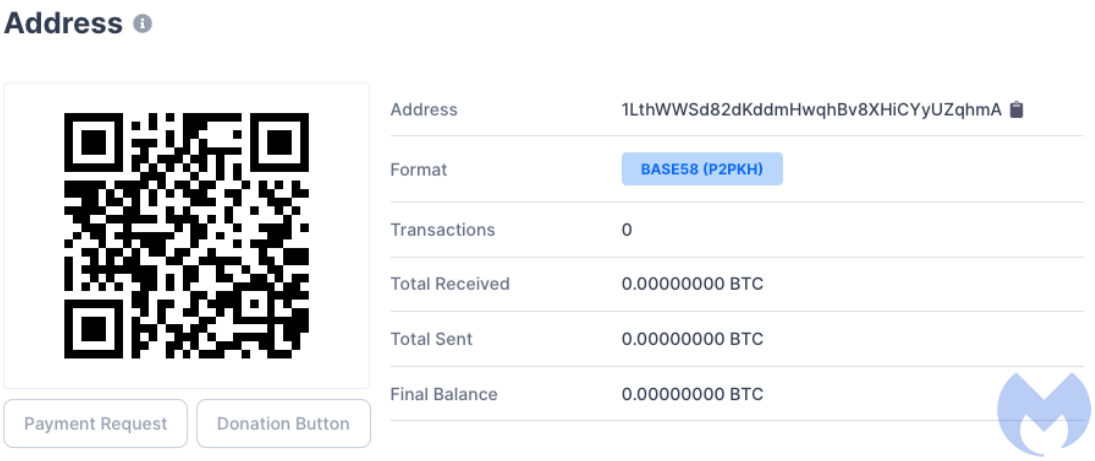
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

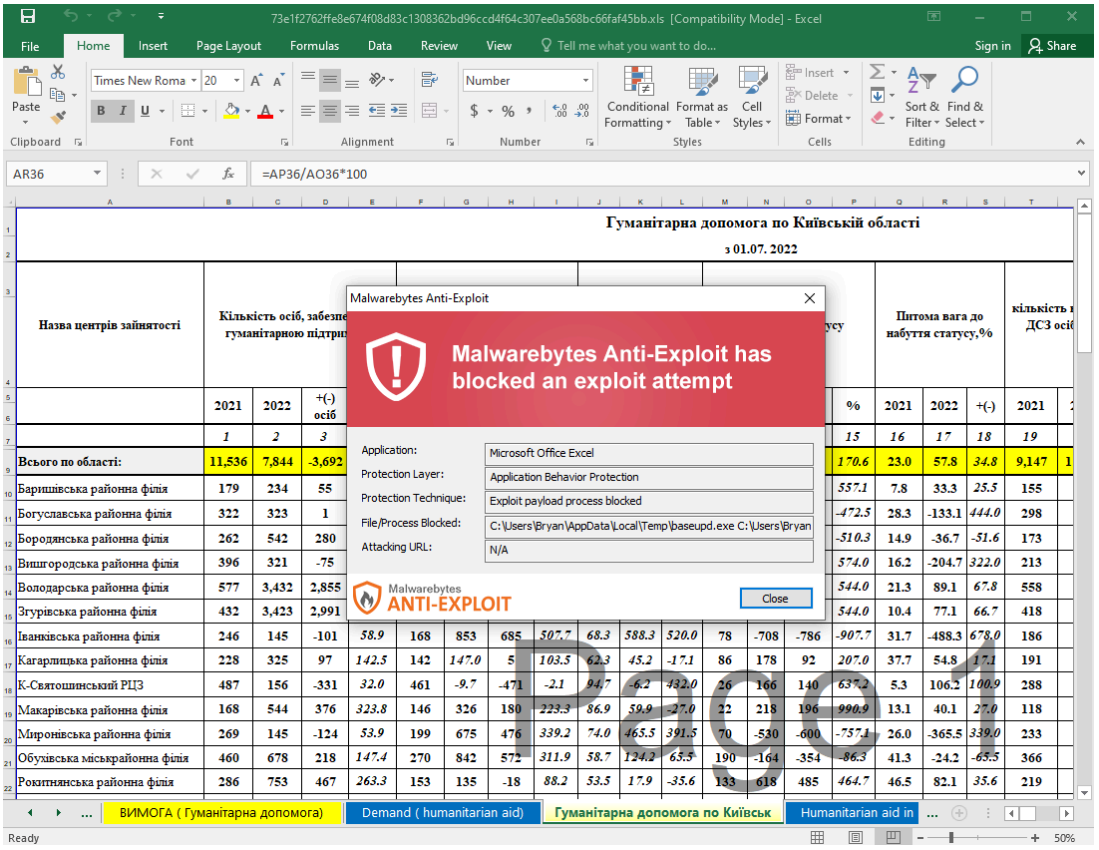
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

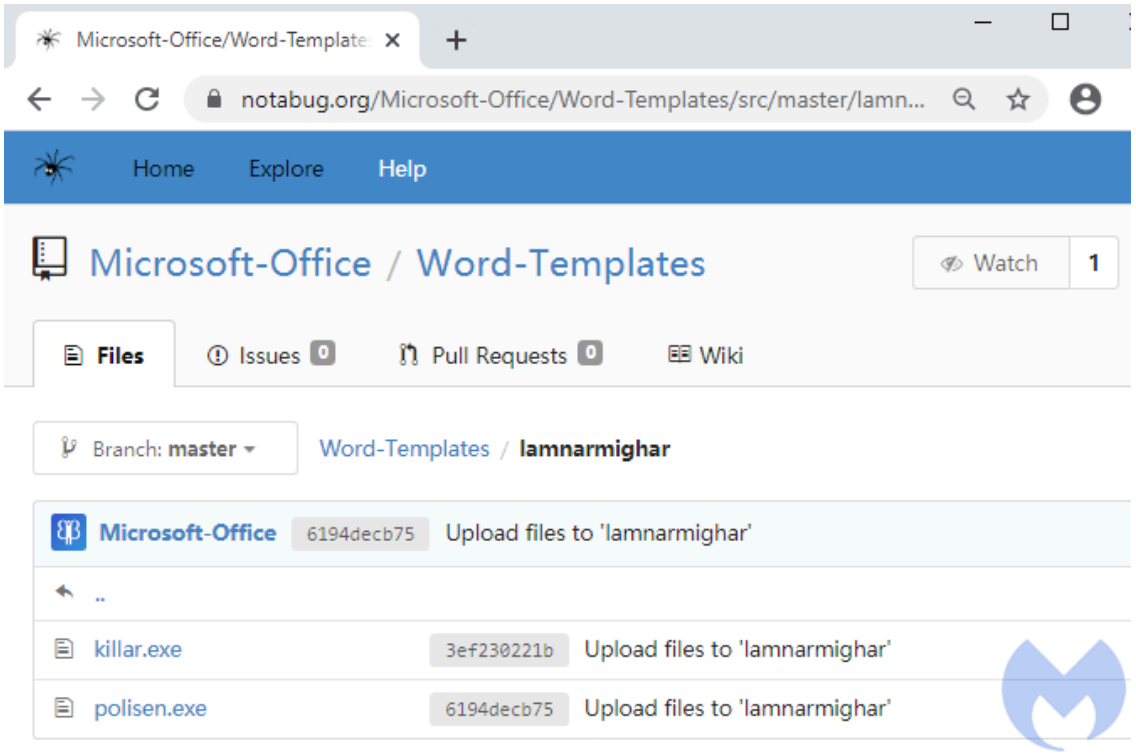
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



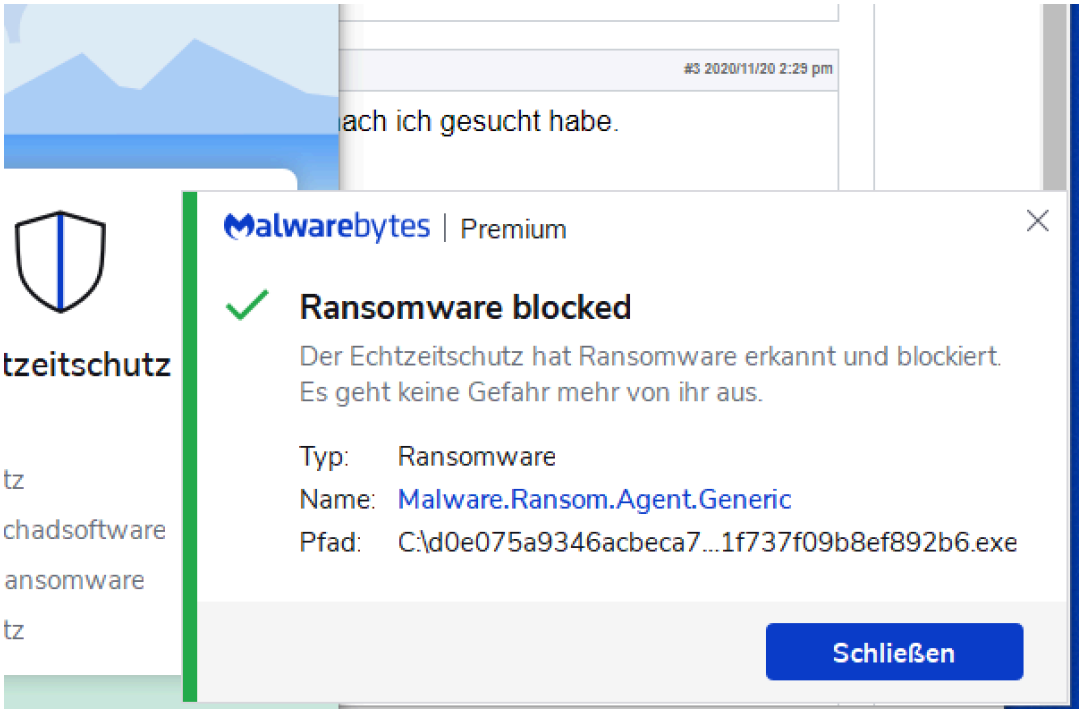
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

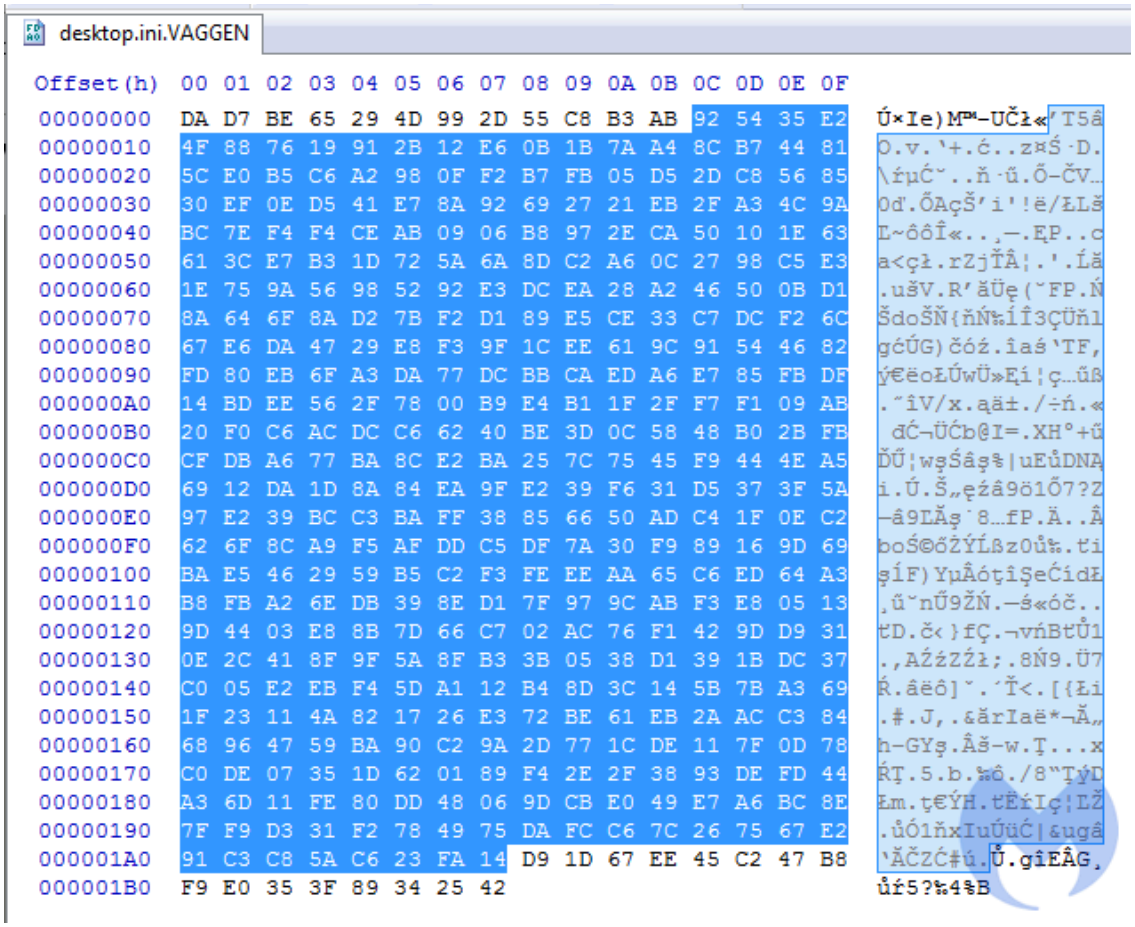
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

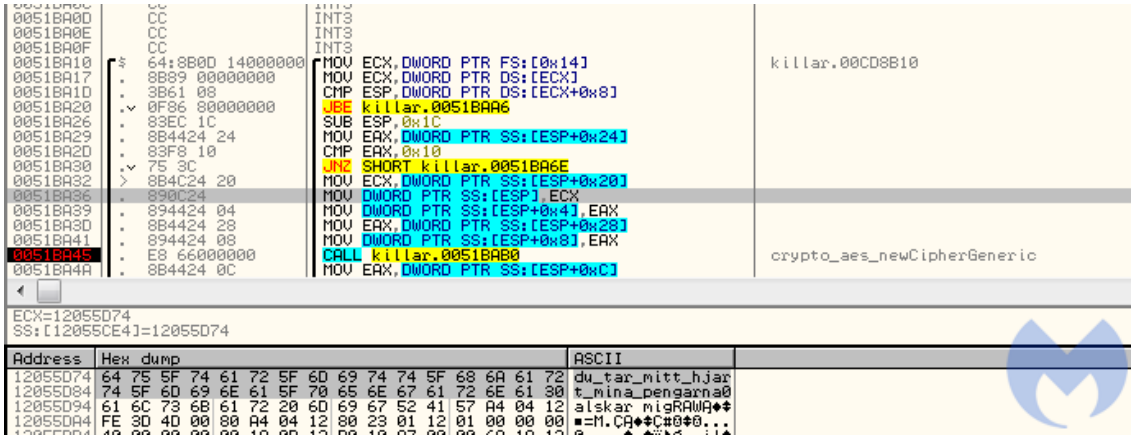


The content of the output file (with .VAGGEN extension) contains:

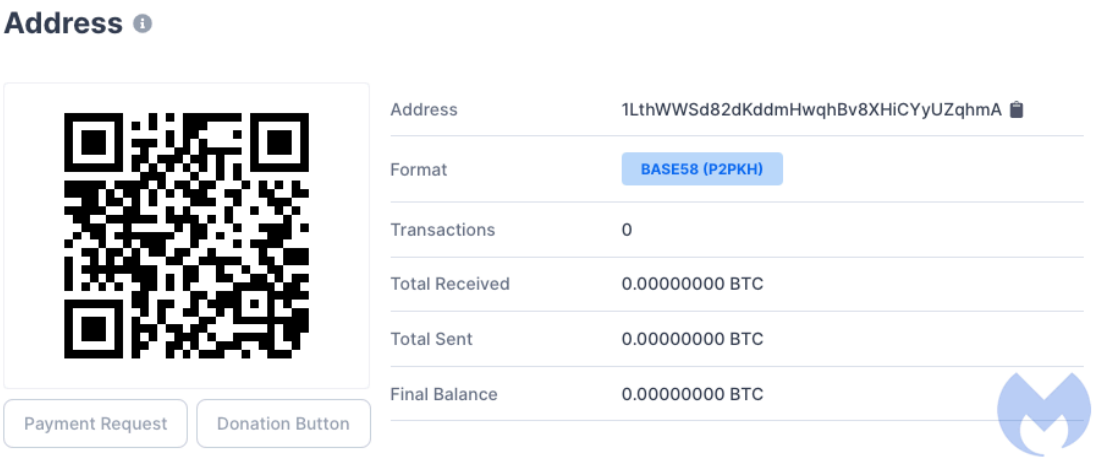
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

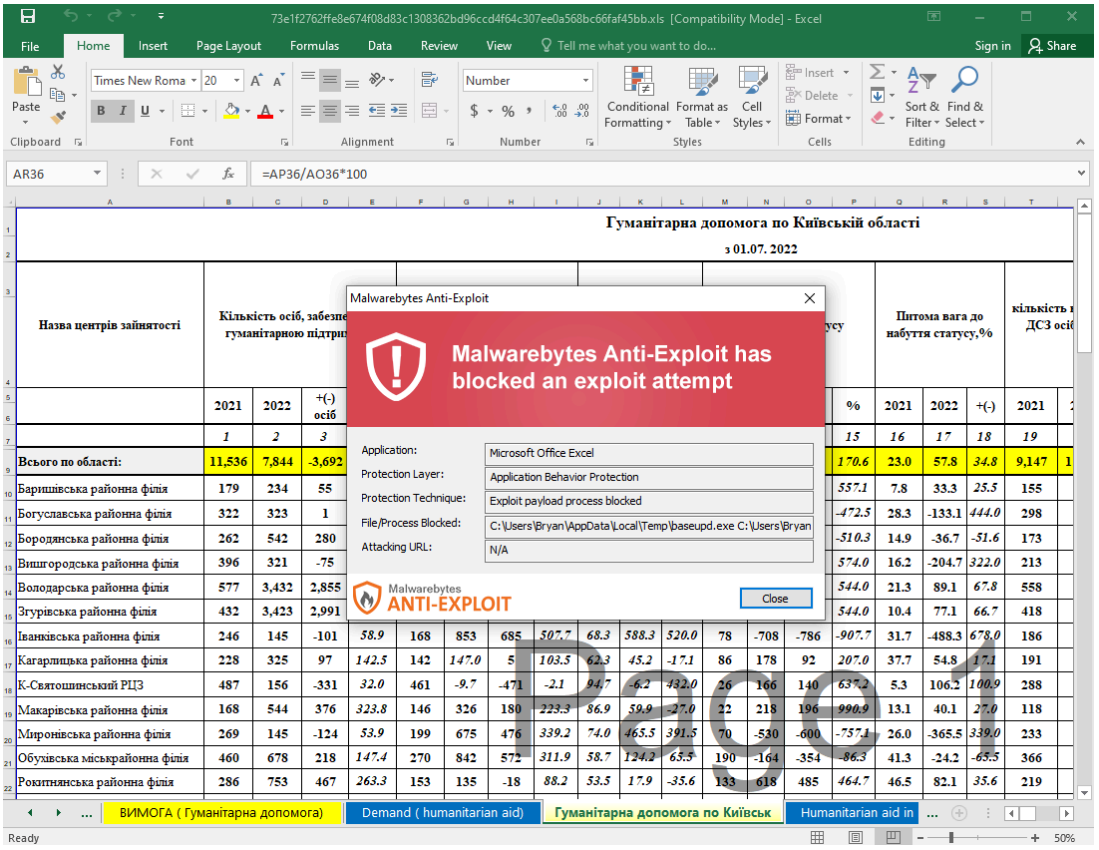
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

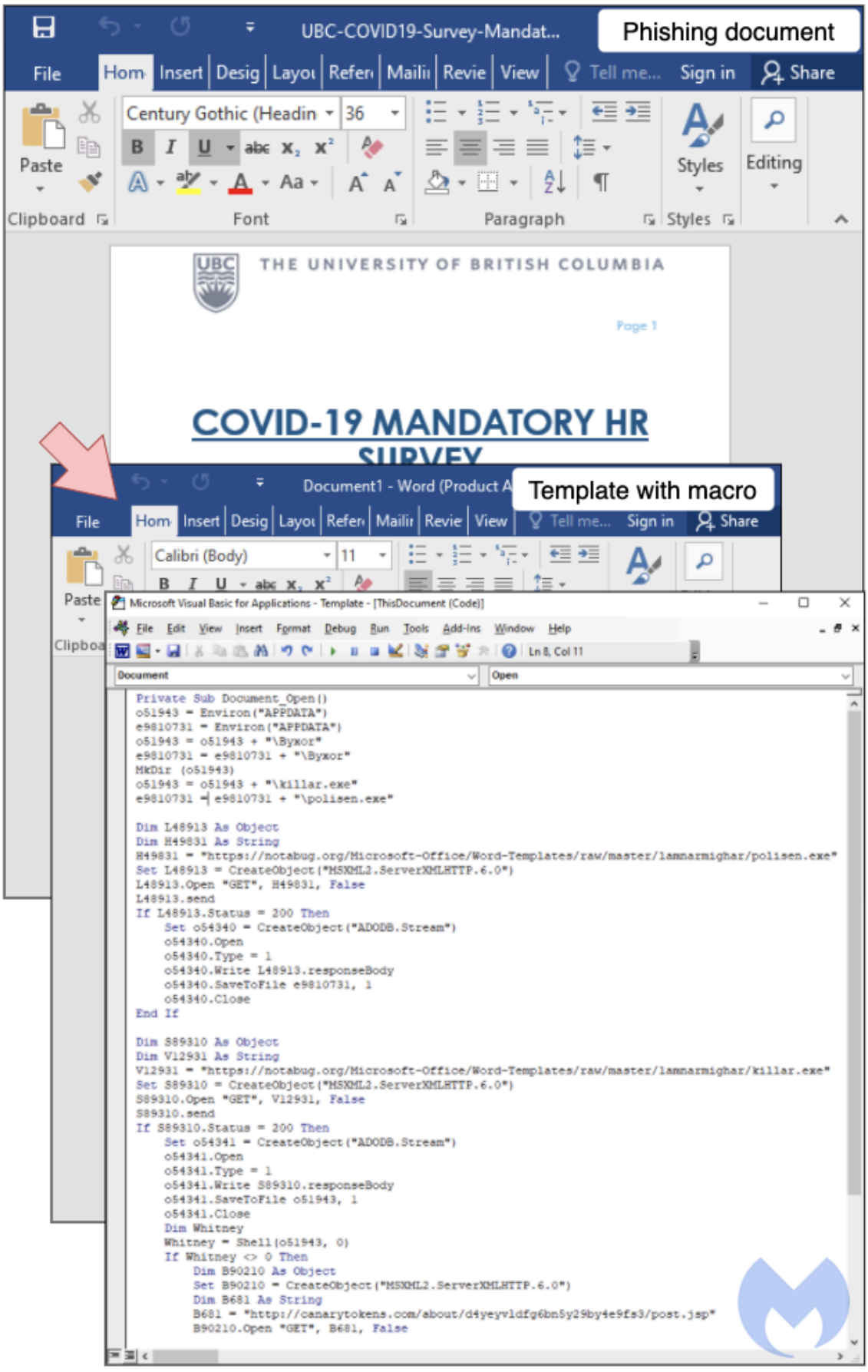
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

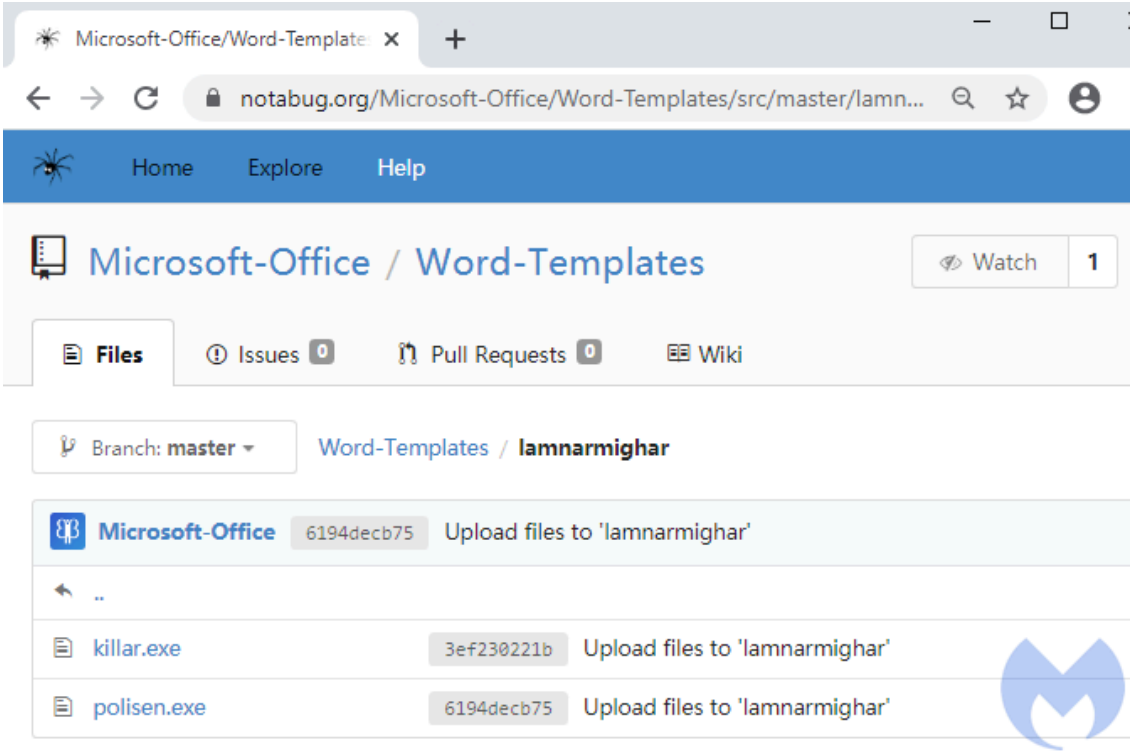
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



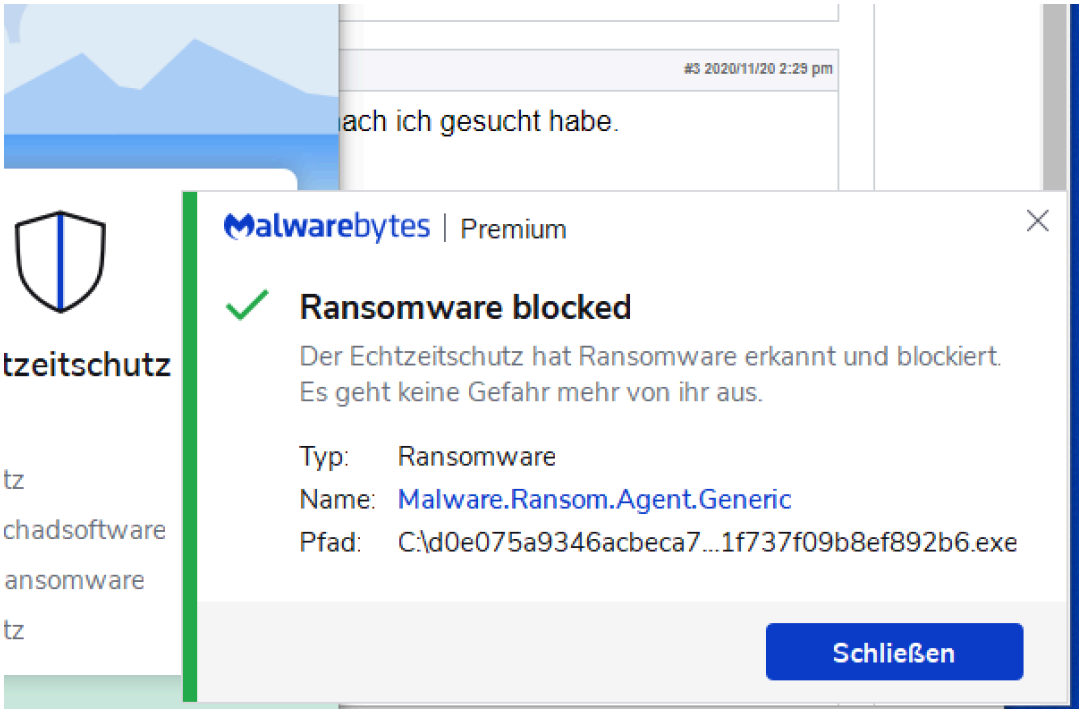
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

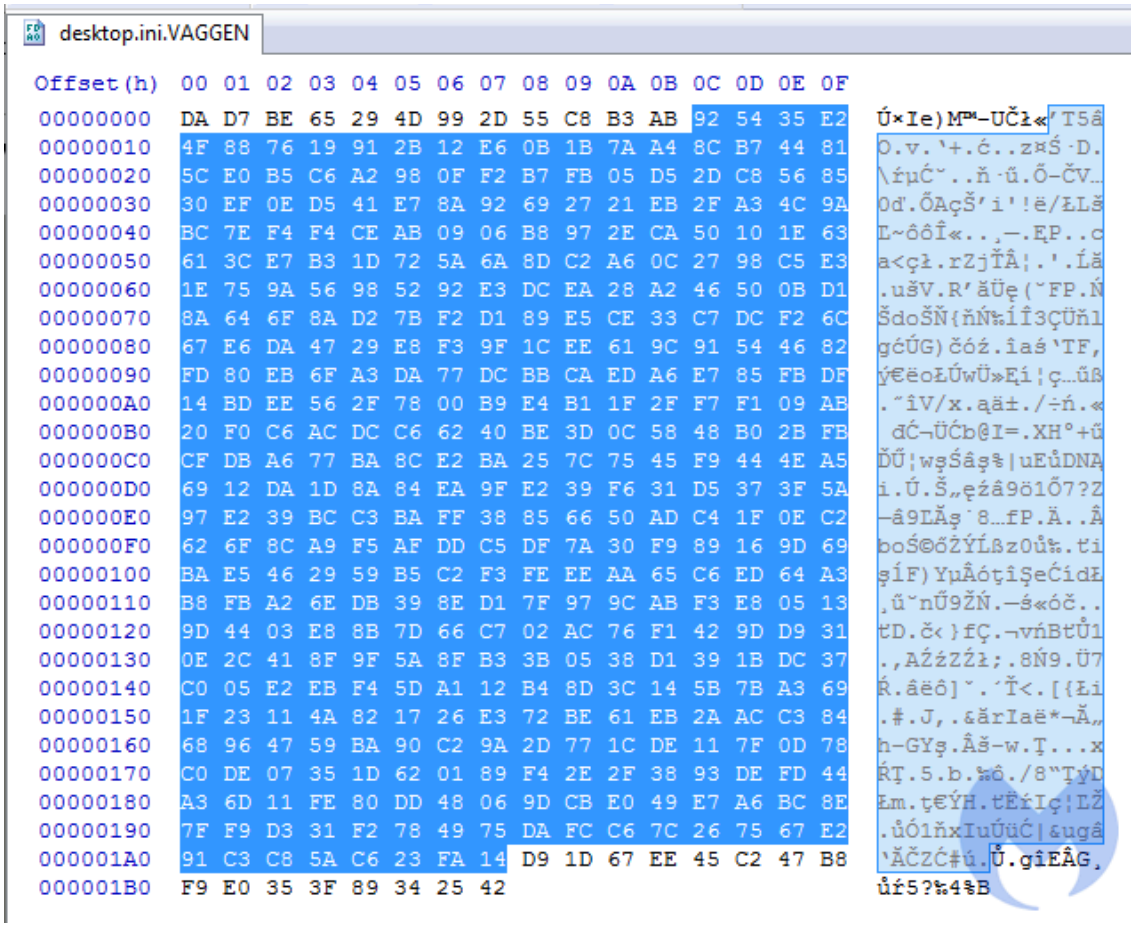
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



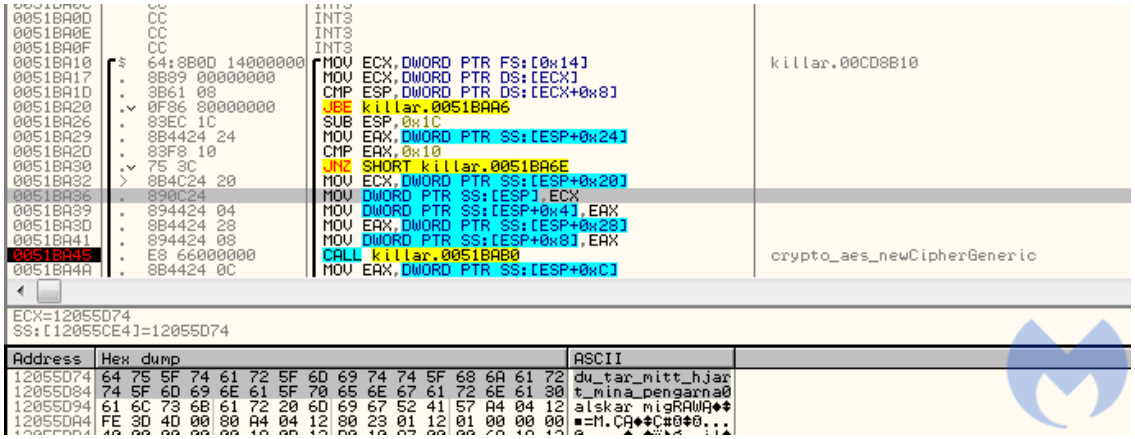
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

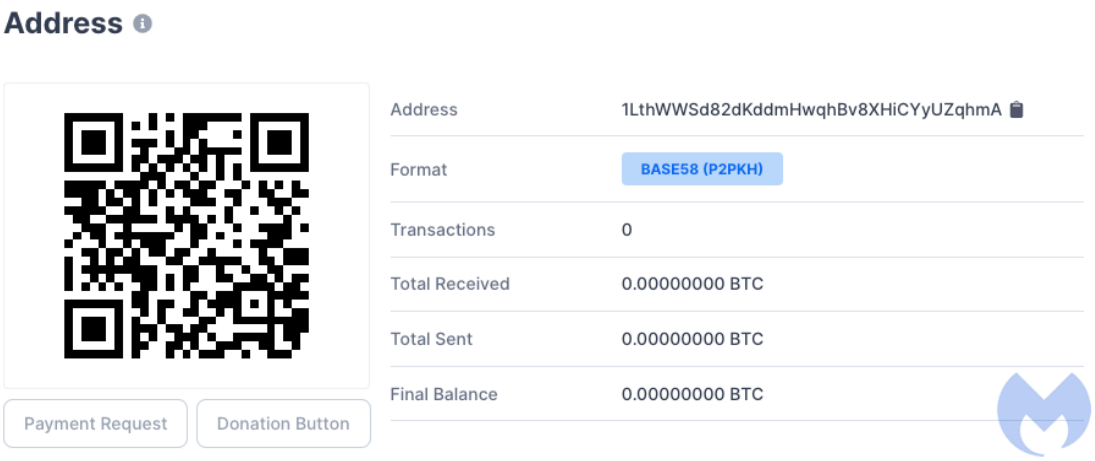
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

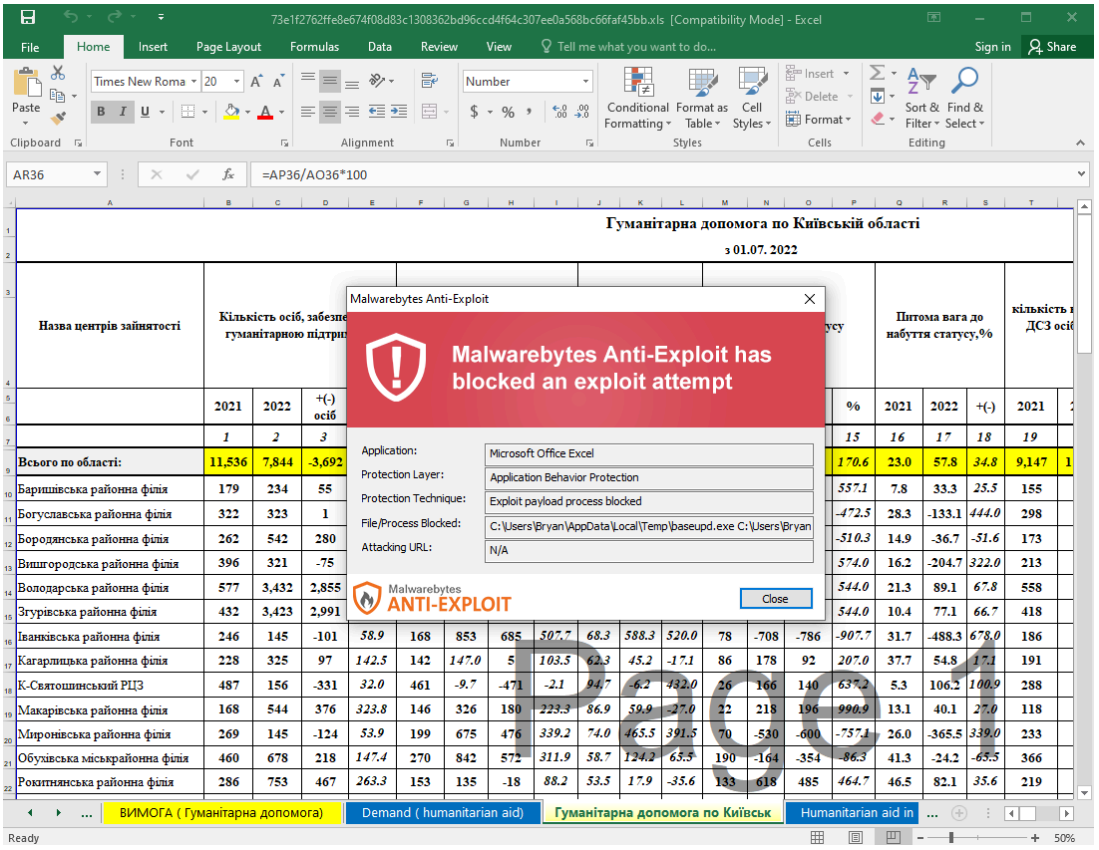
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

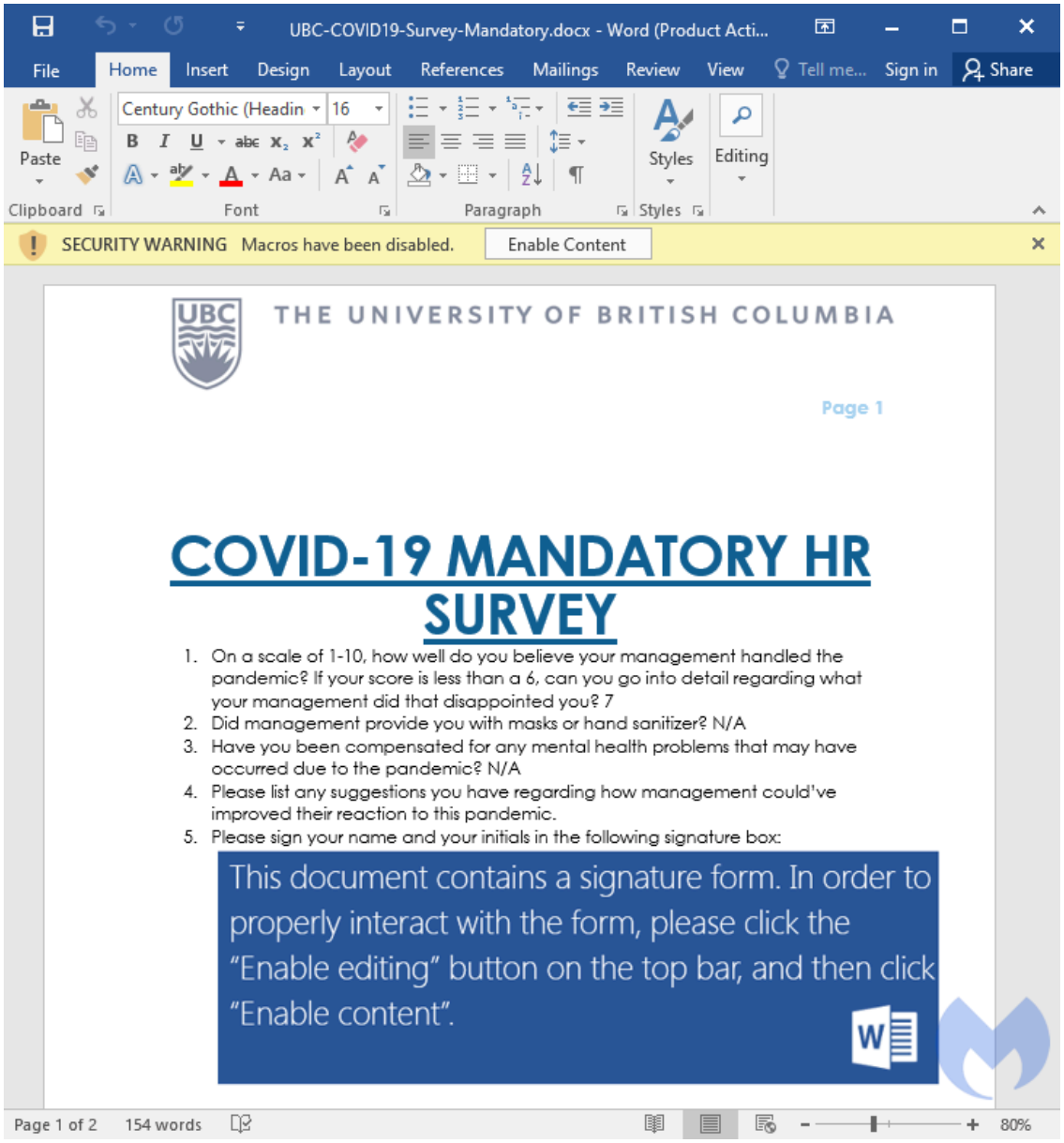
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

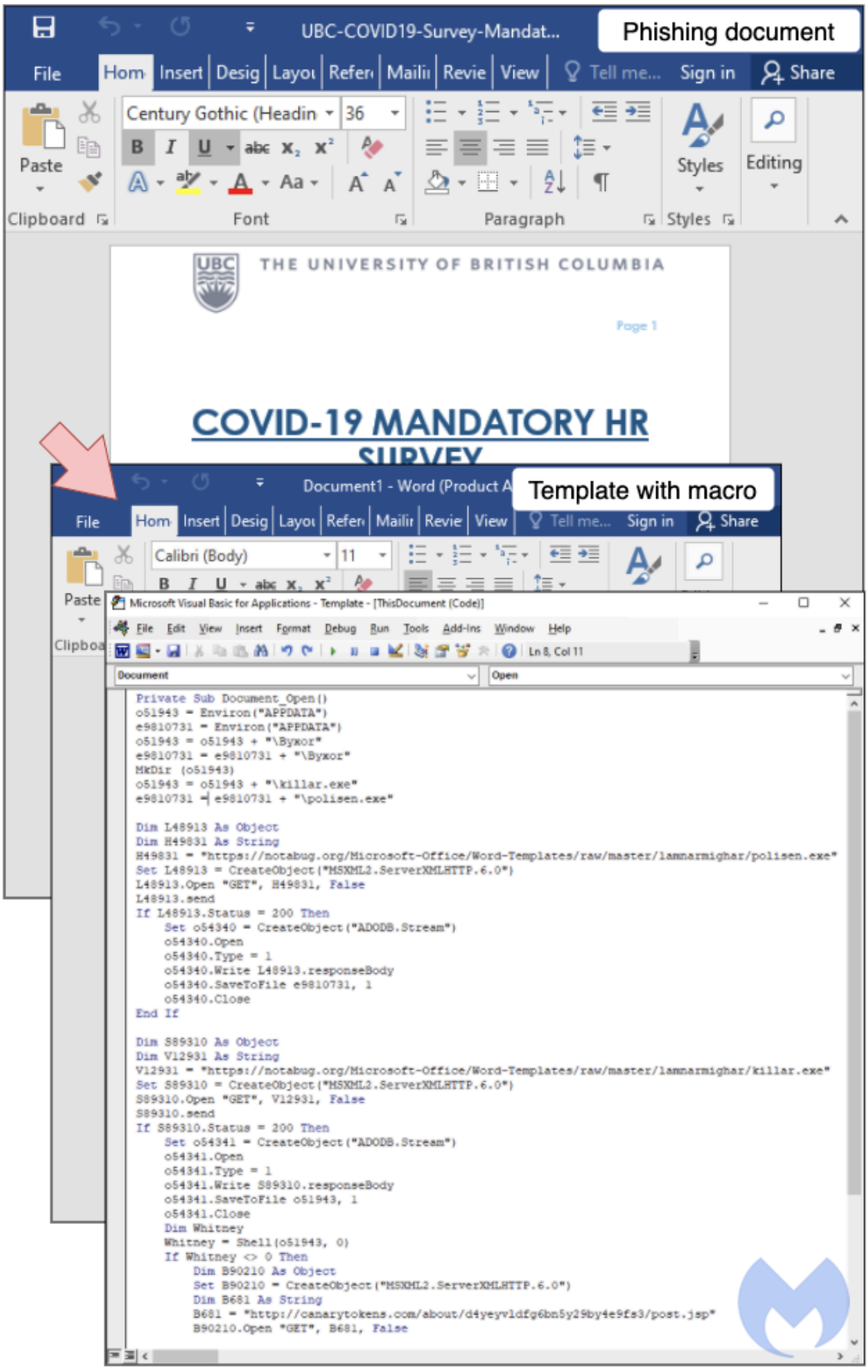
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

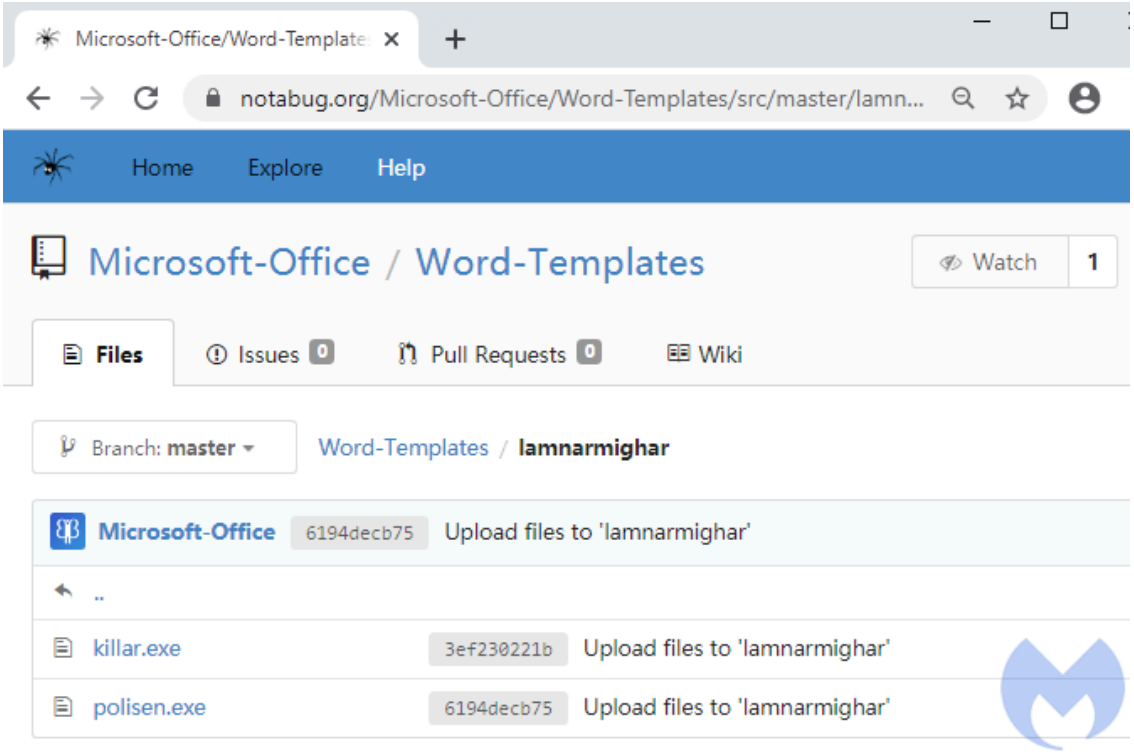
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



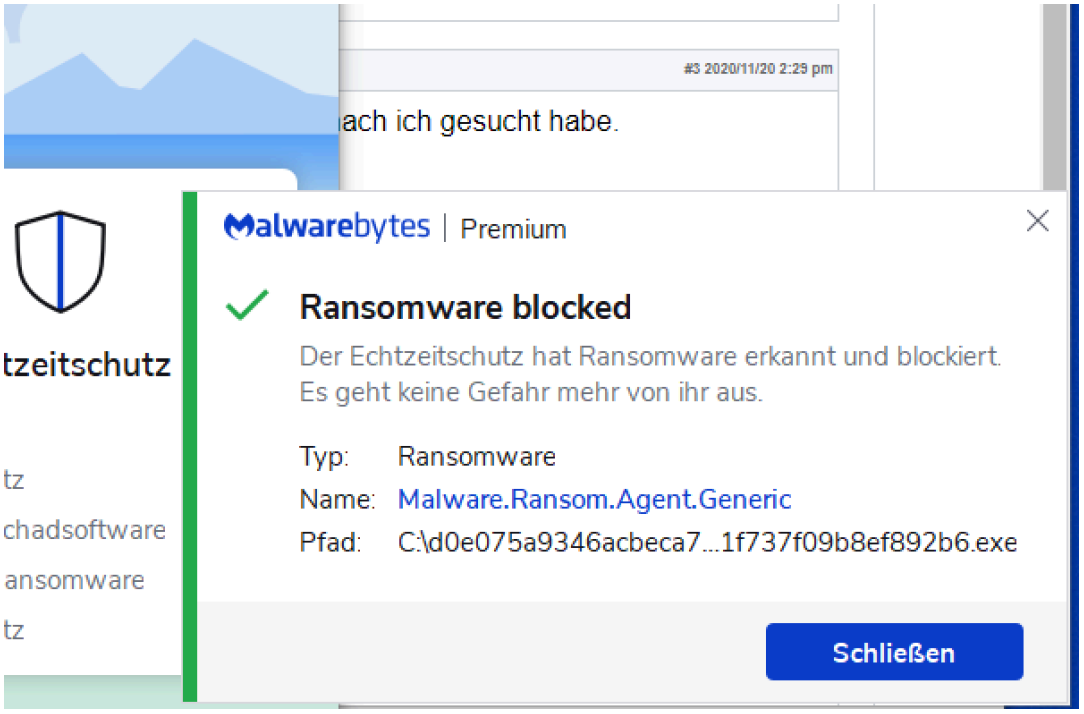
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

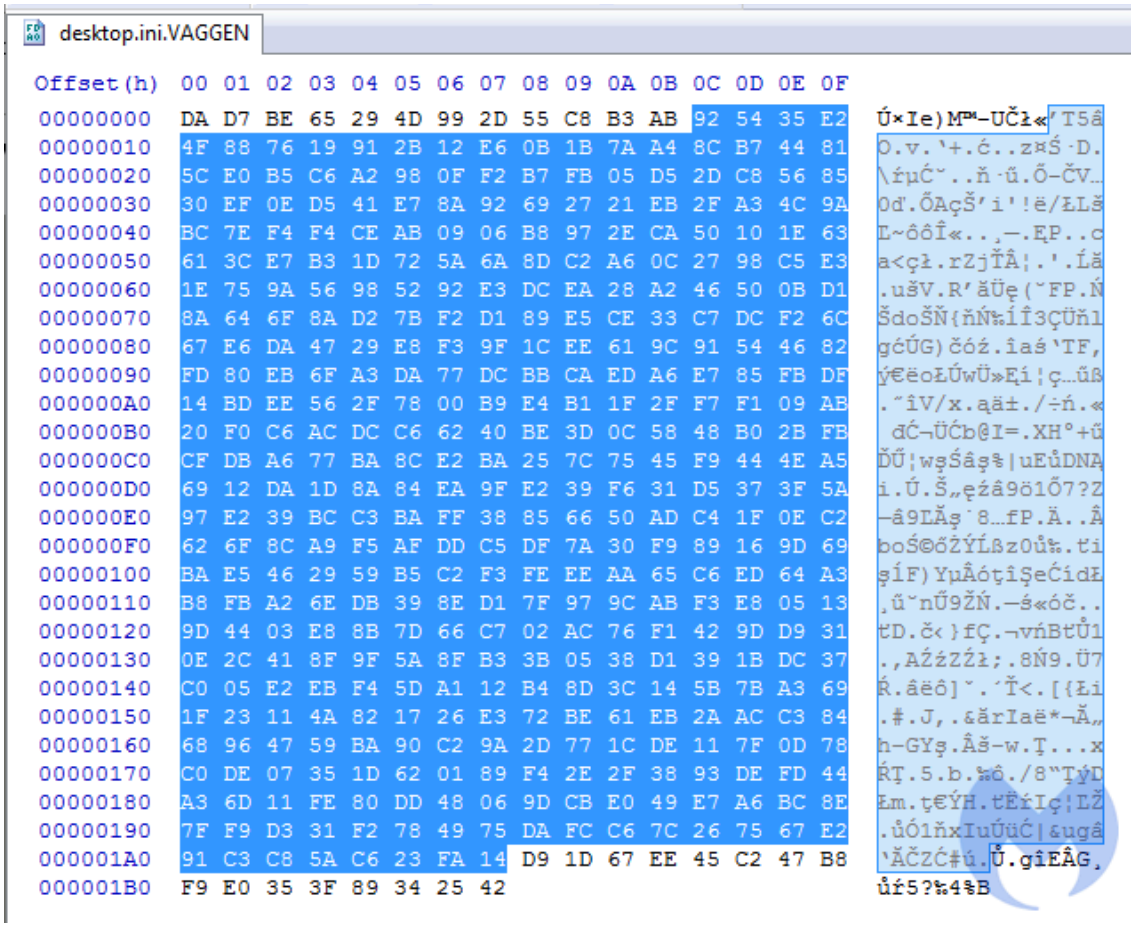
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

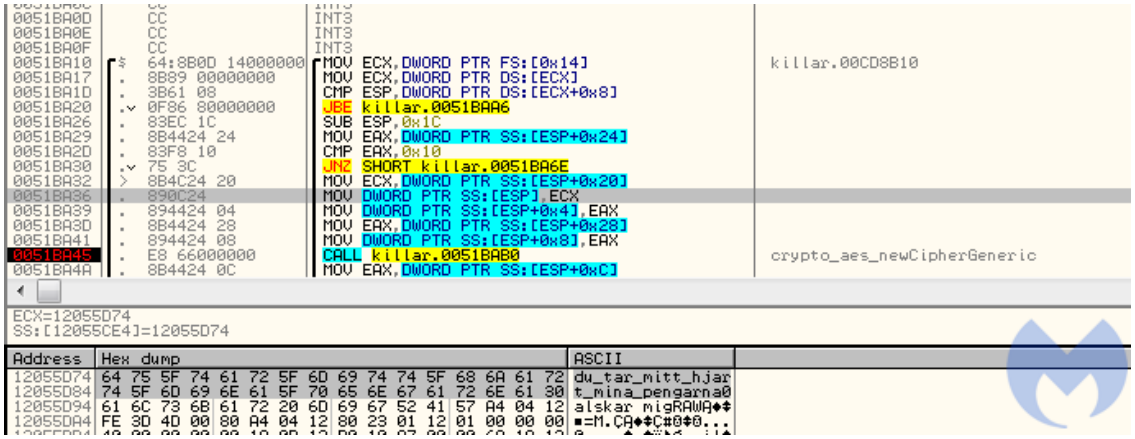


The content of the output file (with .VAGGEN extension) contains:

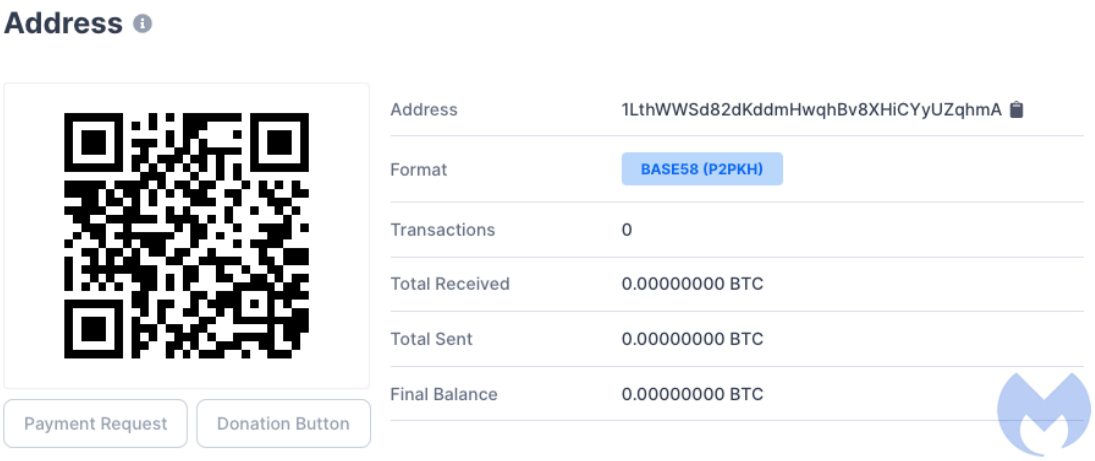
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

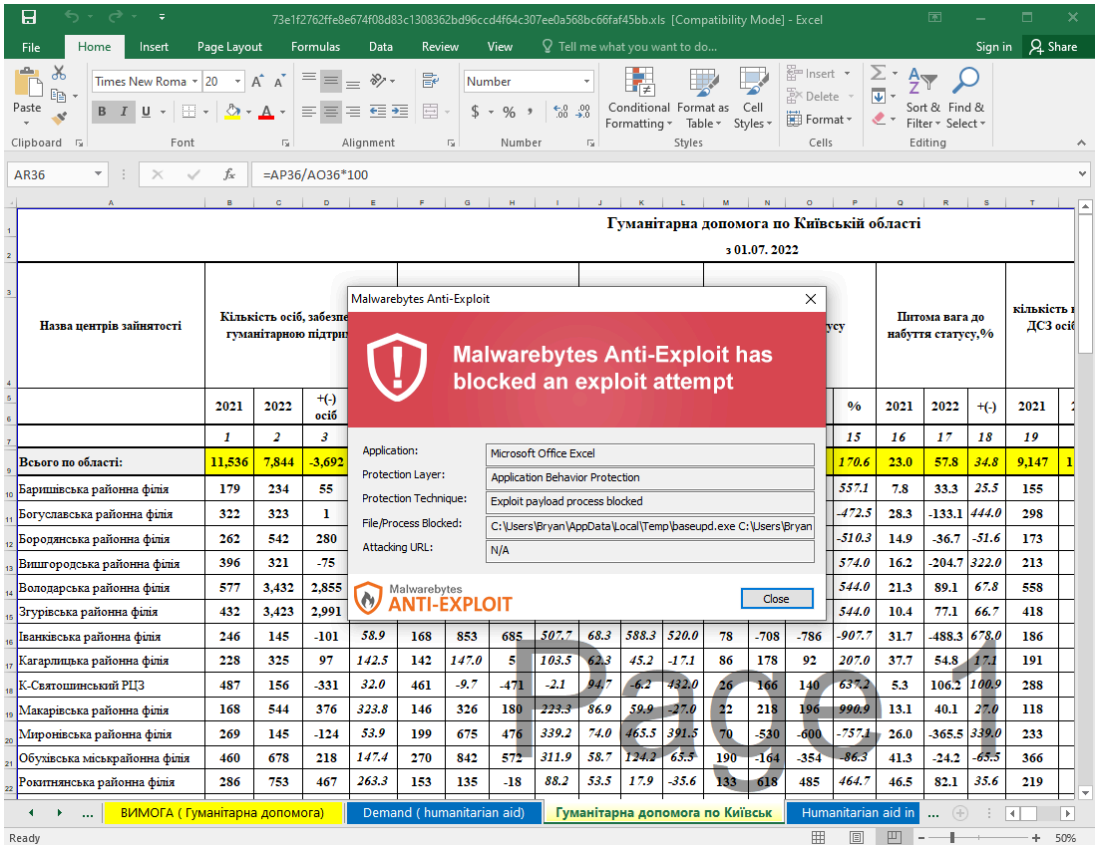
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

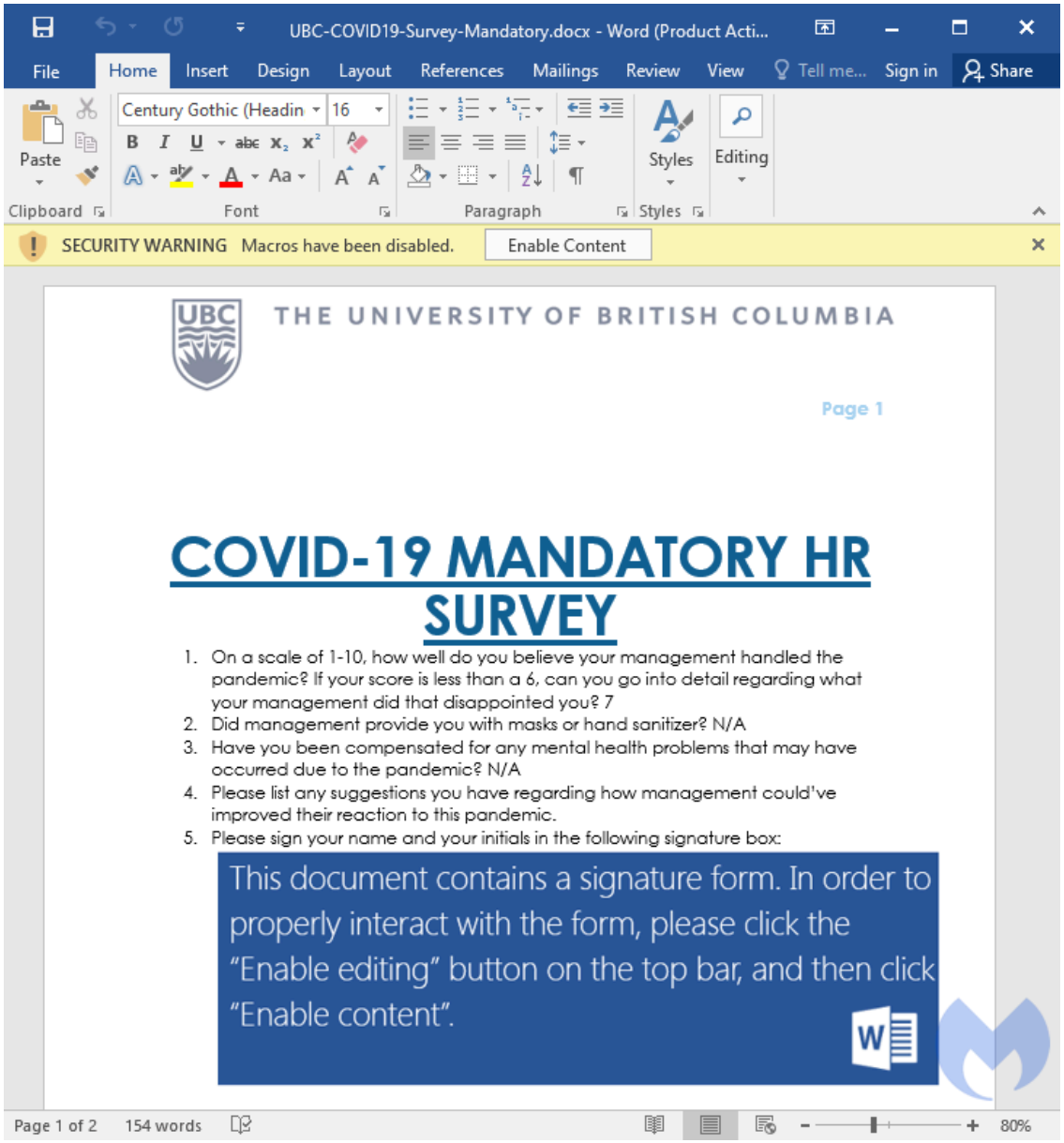
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

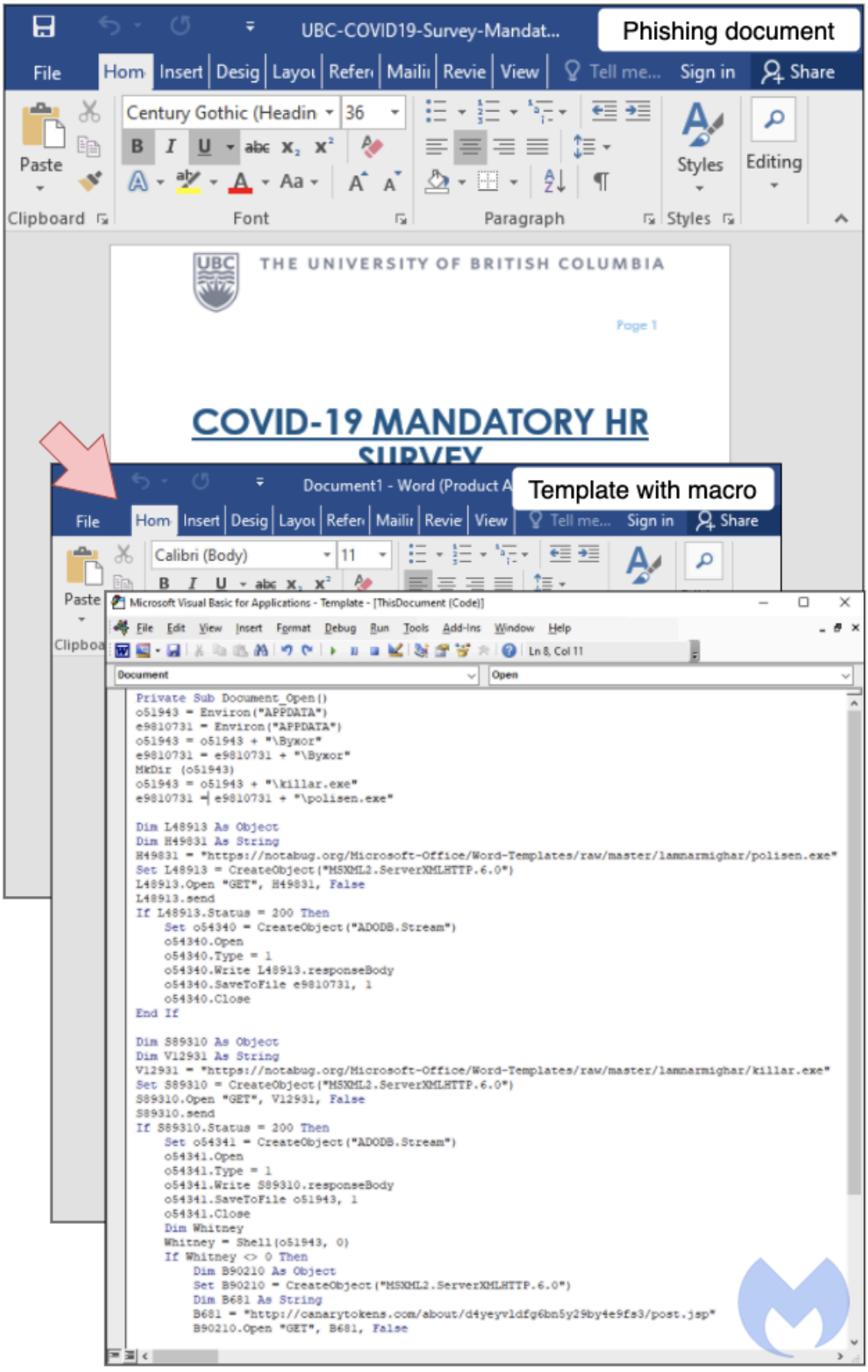
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

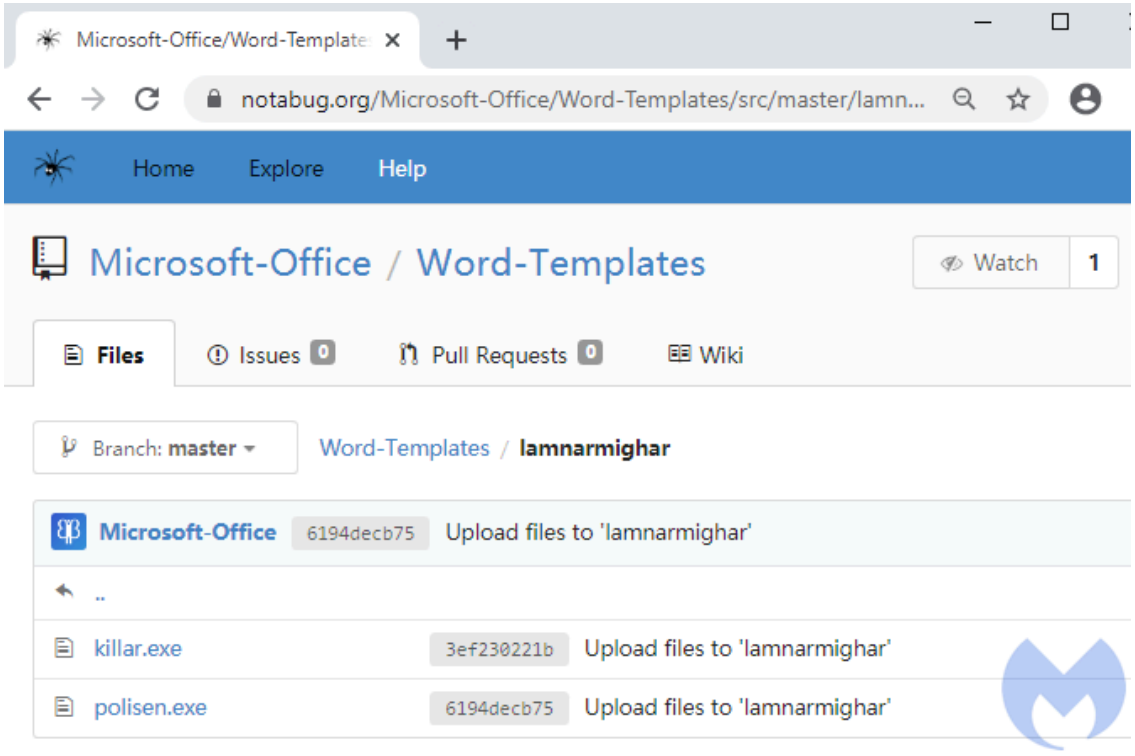
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



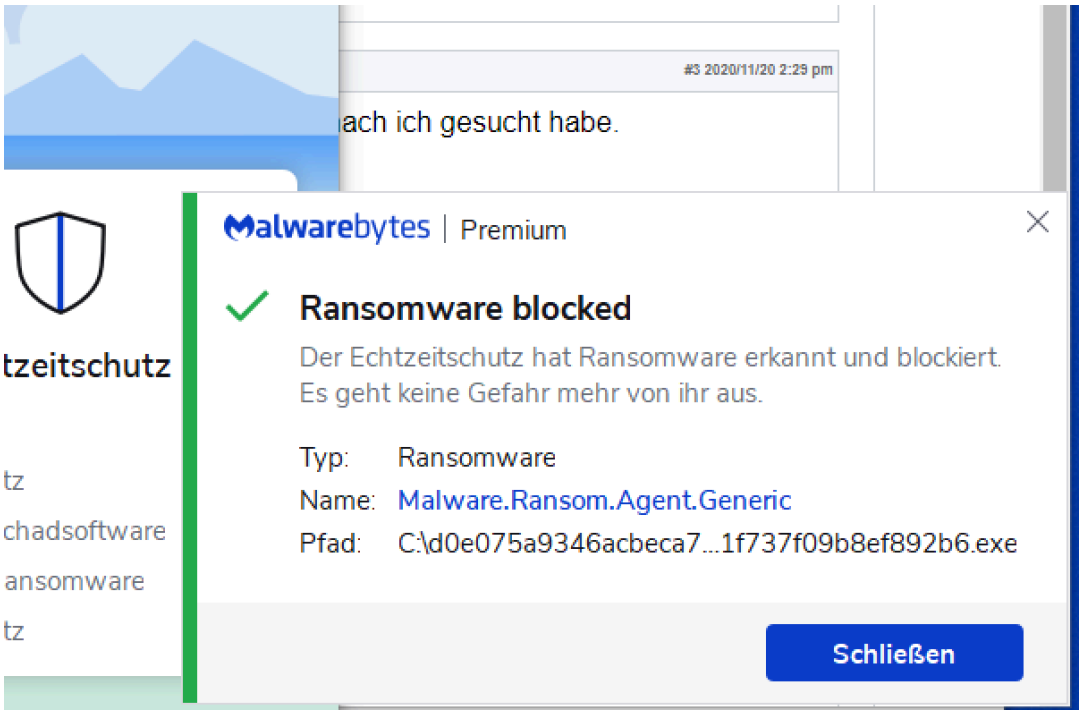
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

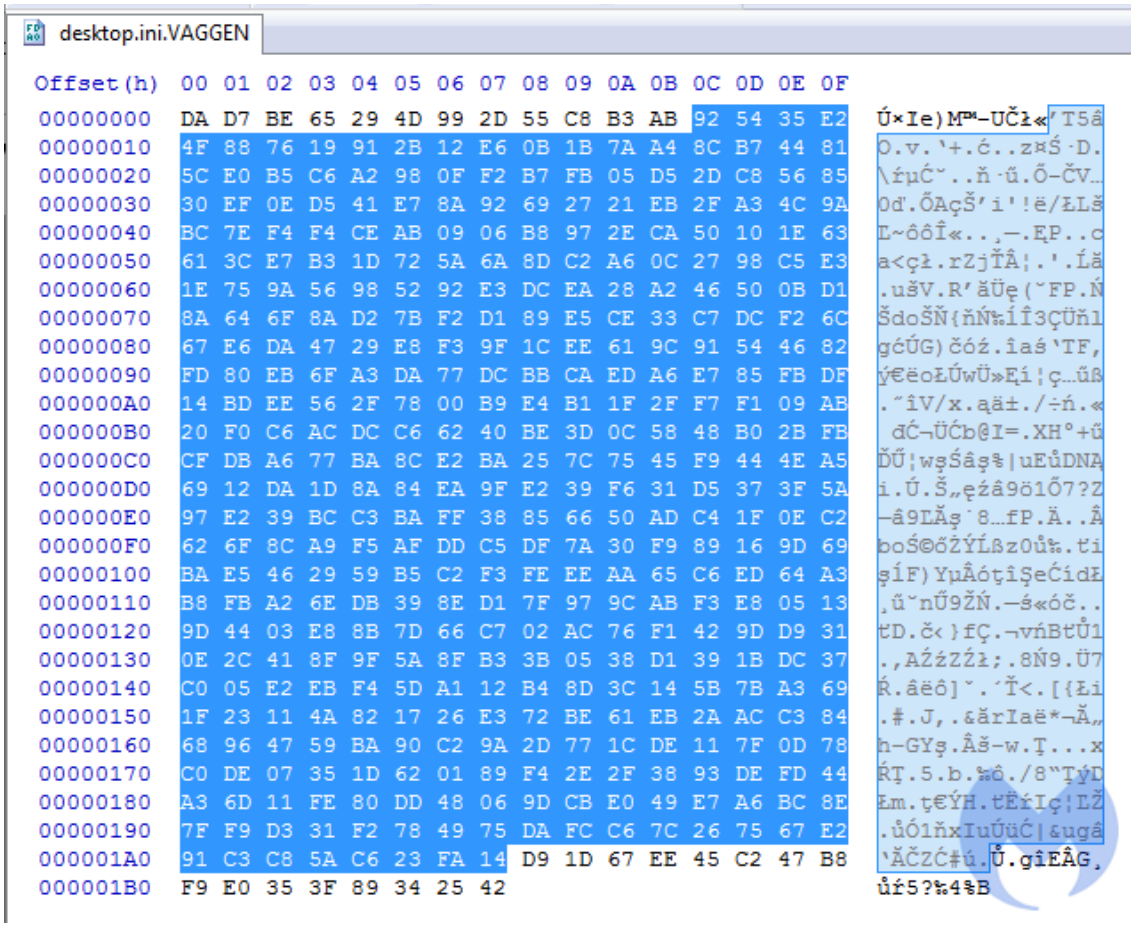
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

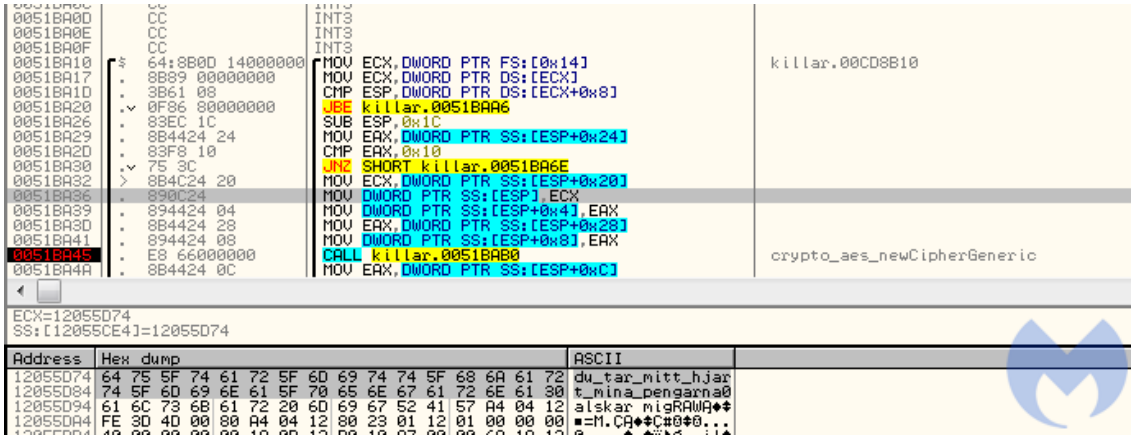


The content of the output file (with .VAGGEN extension) contains:

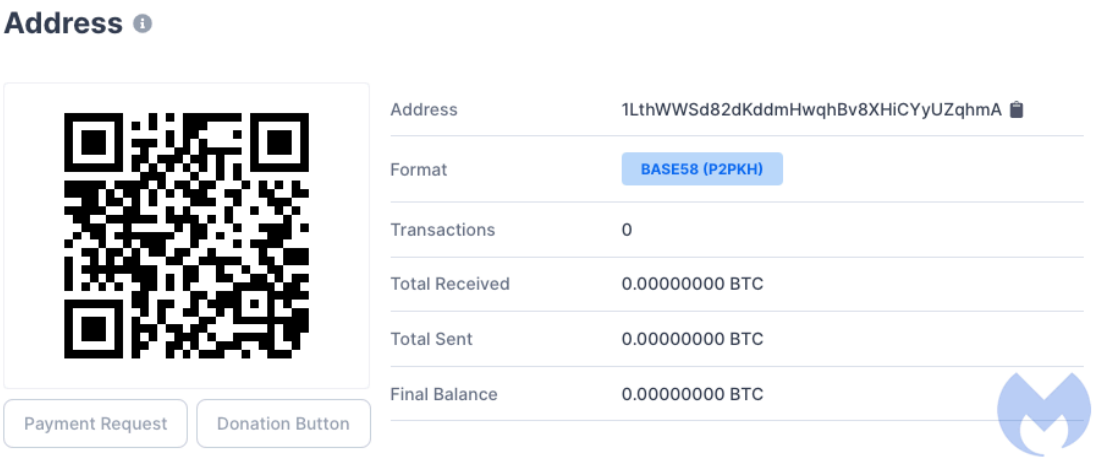
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

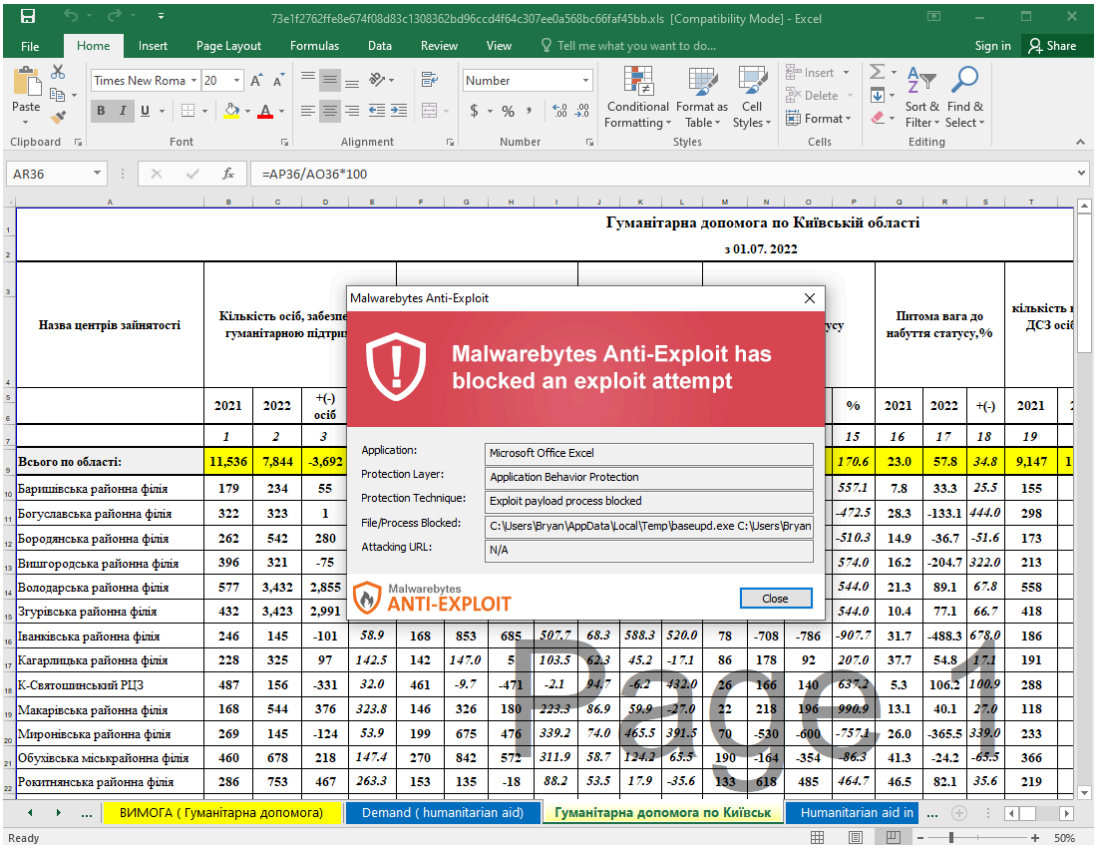
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

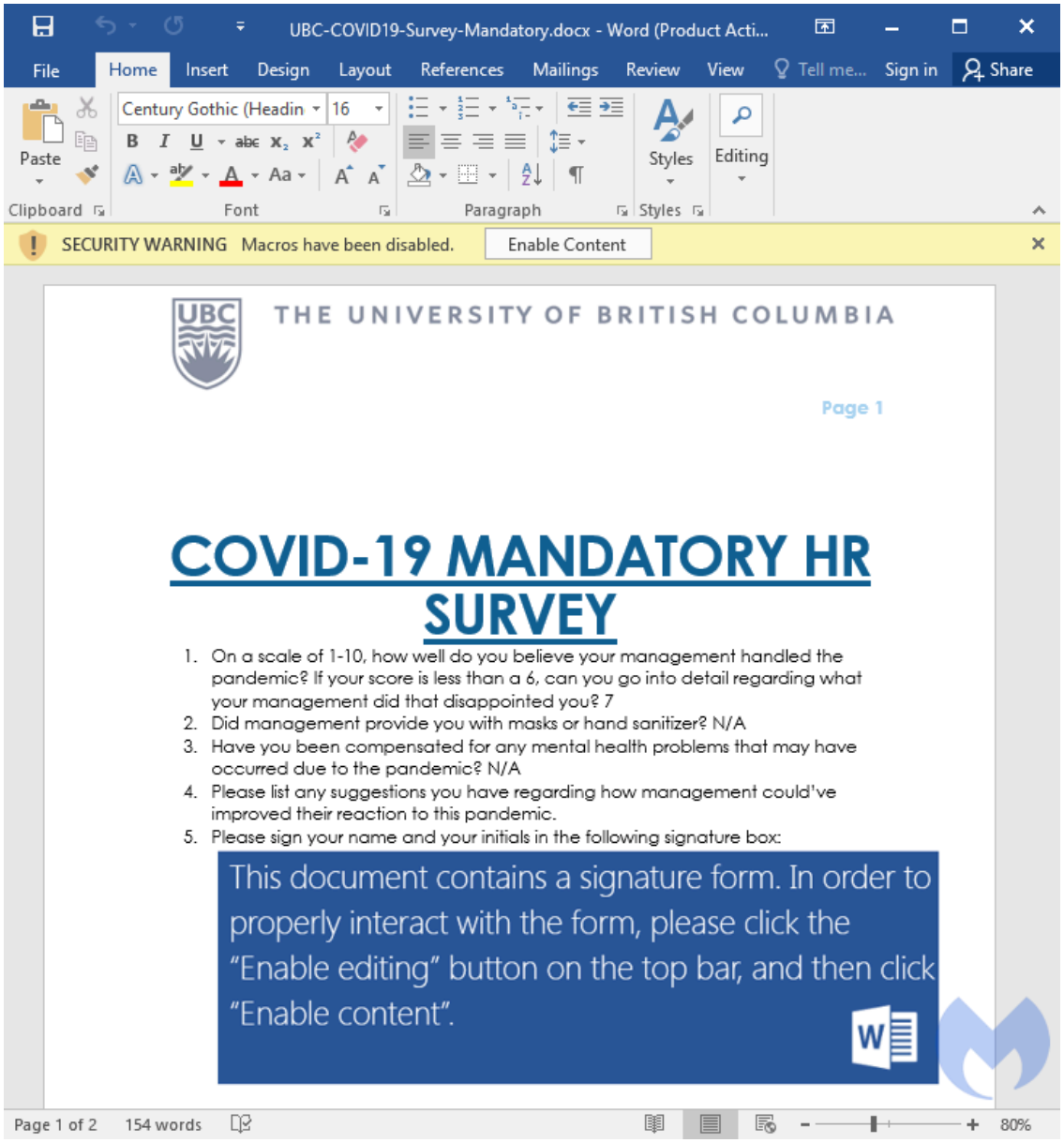
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

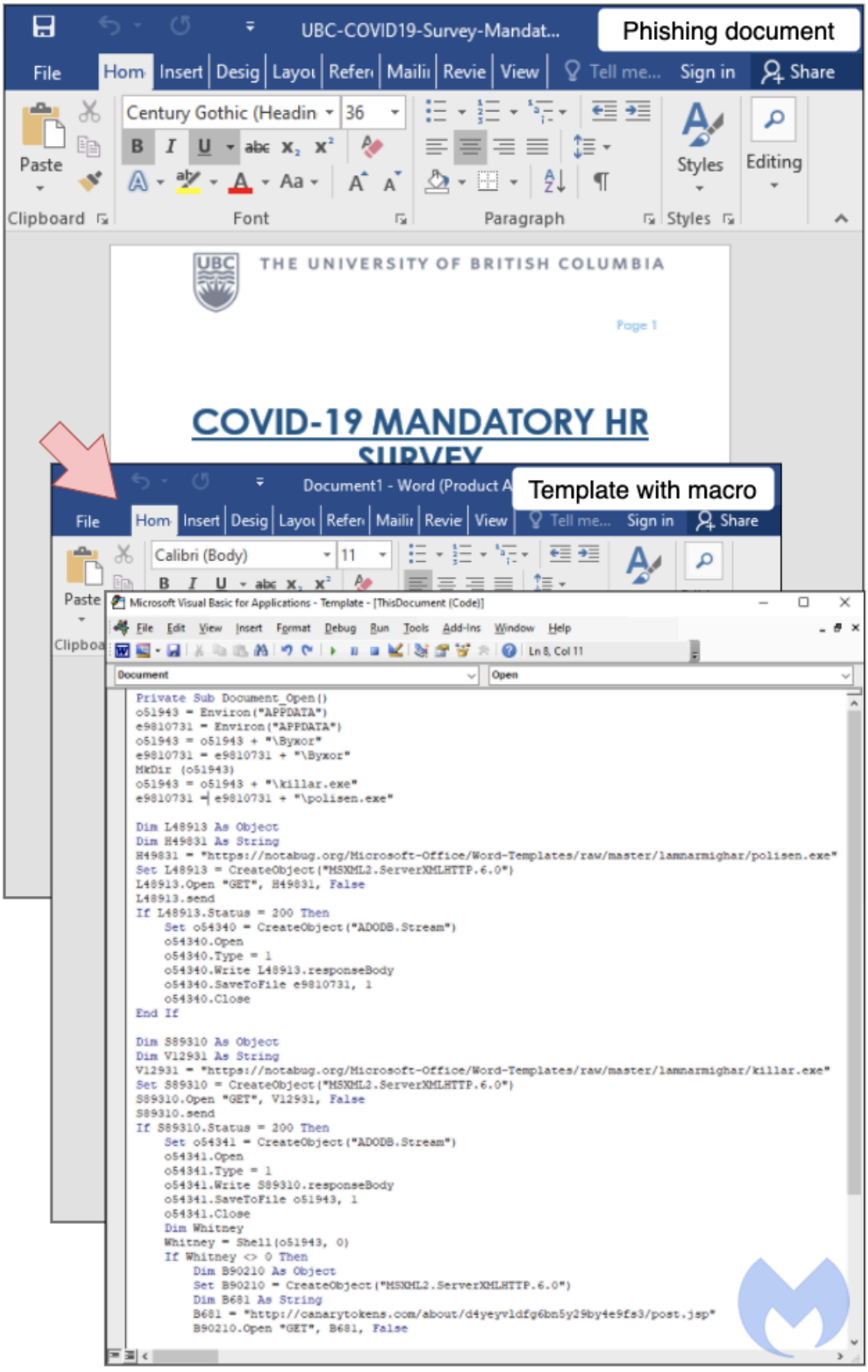
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

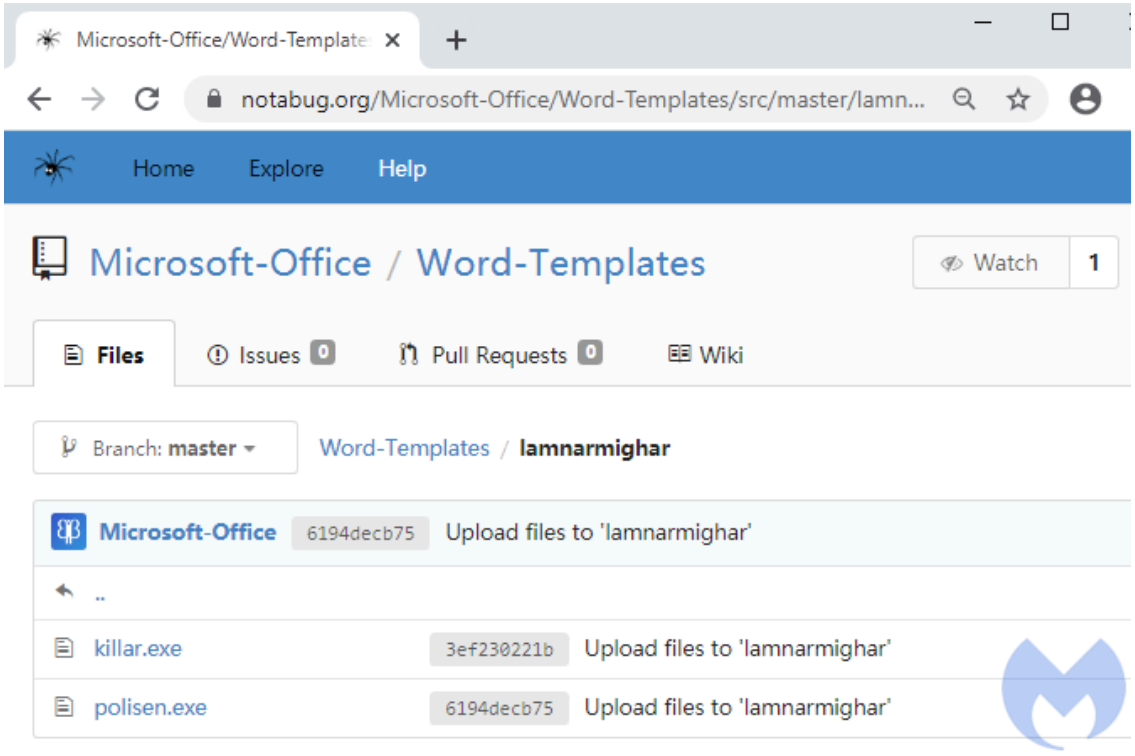
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



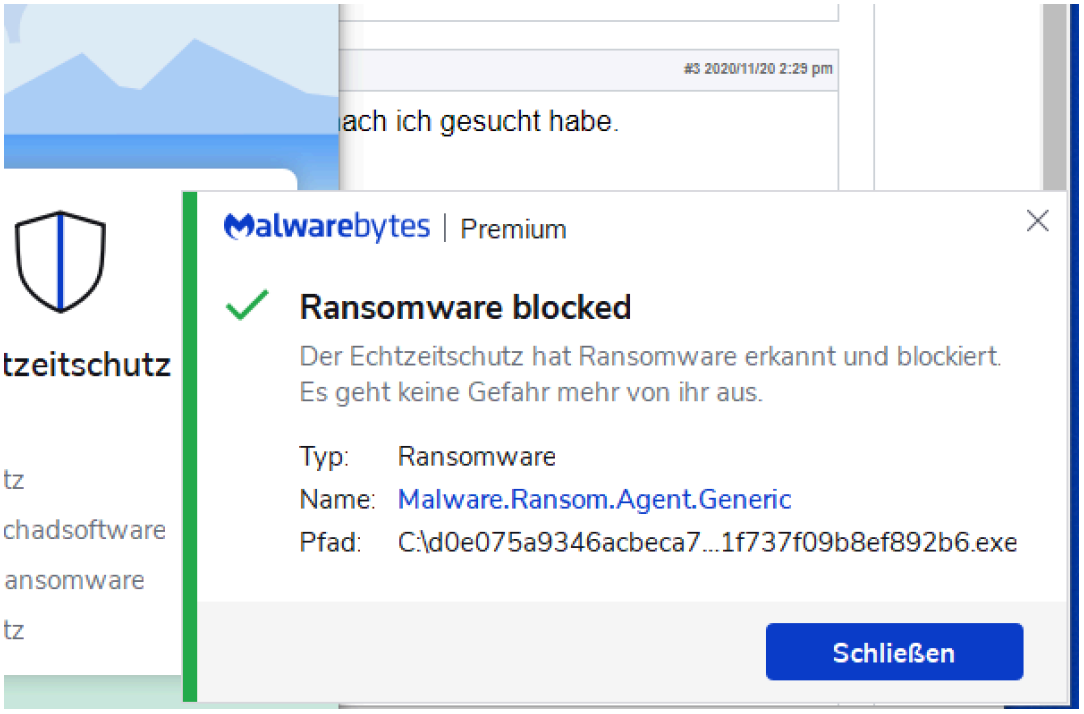
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

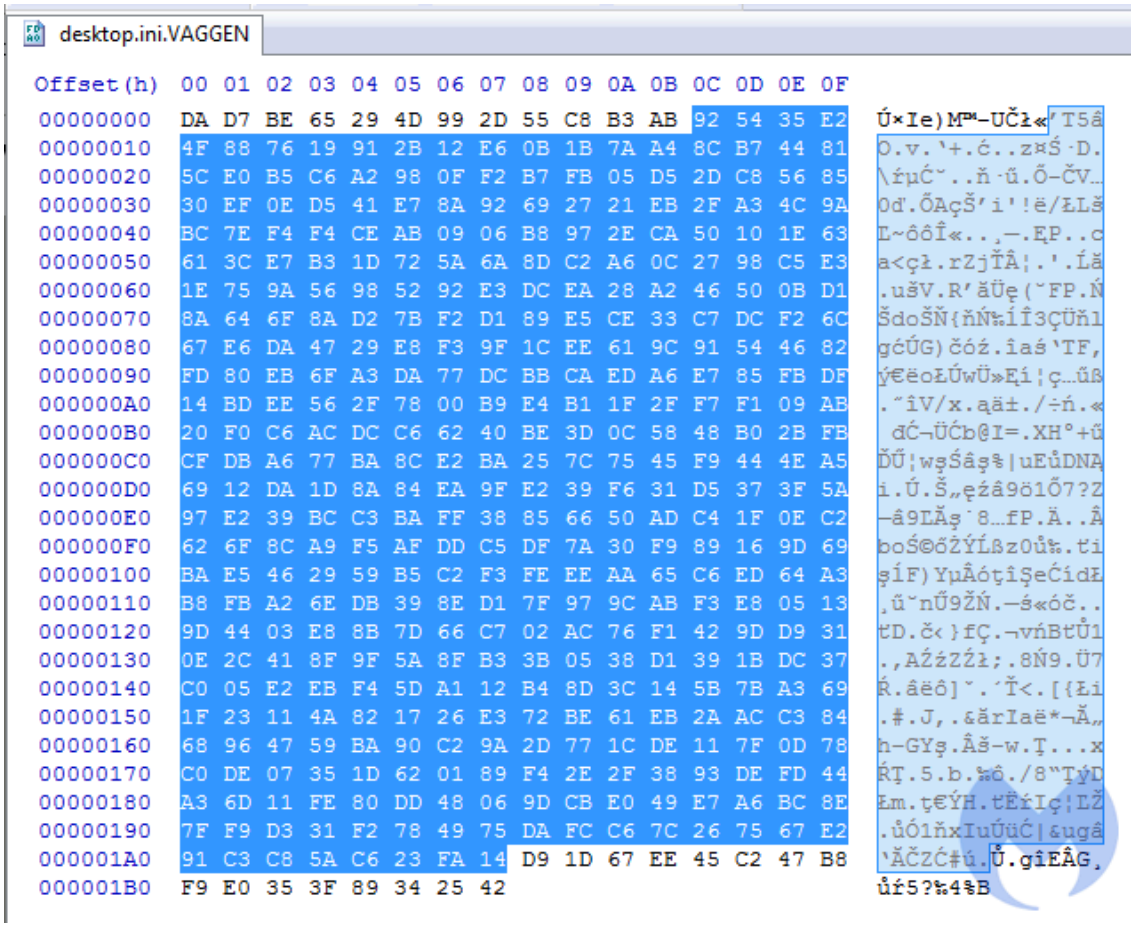
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



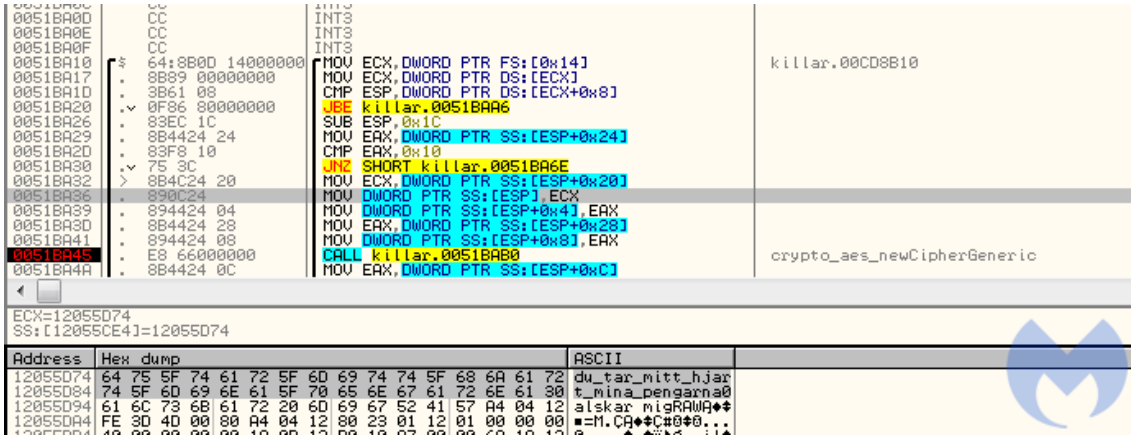
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

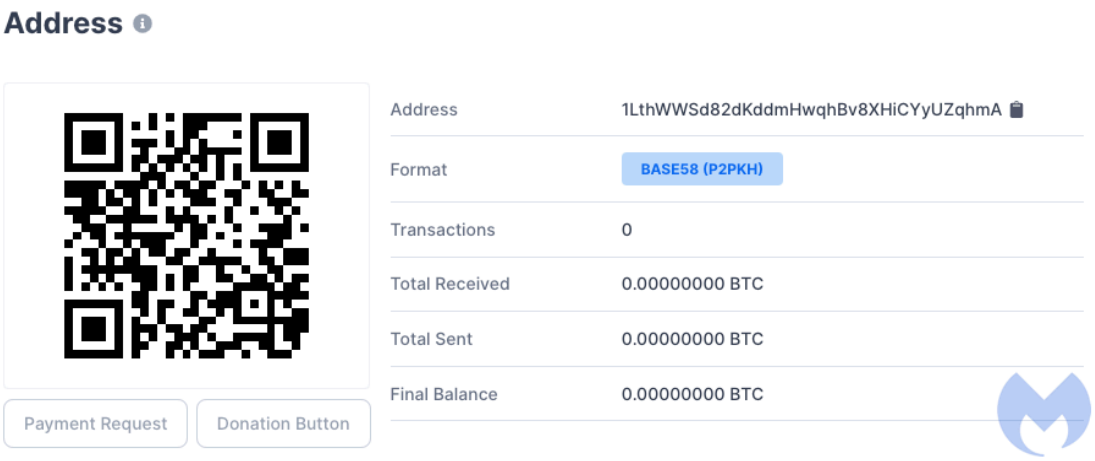
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

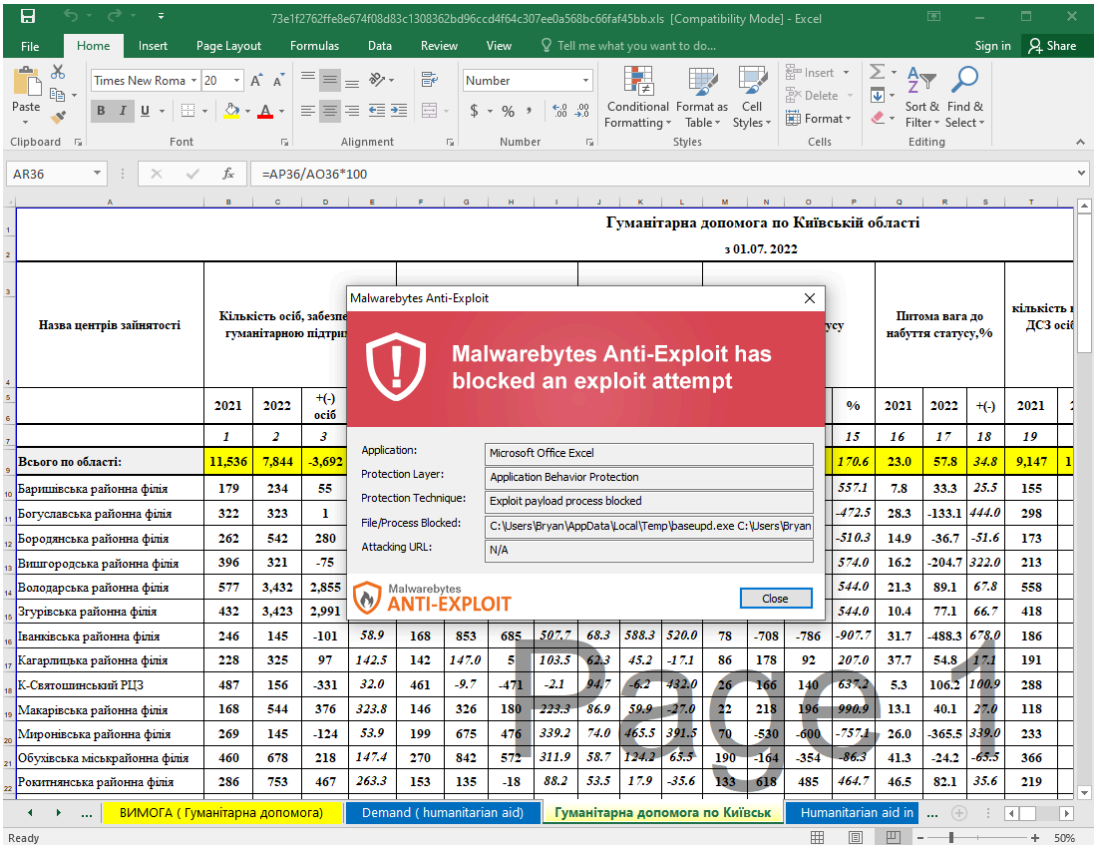
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

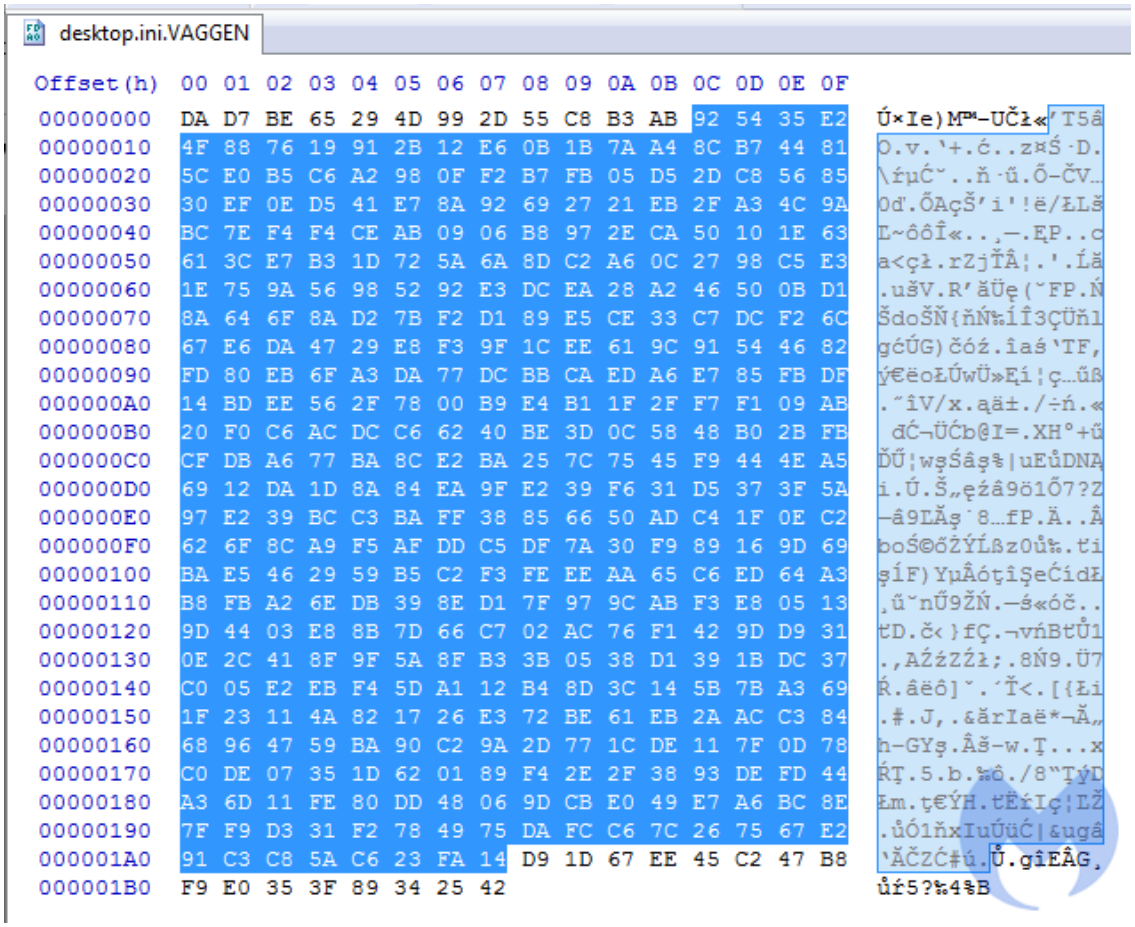
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

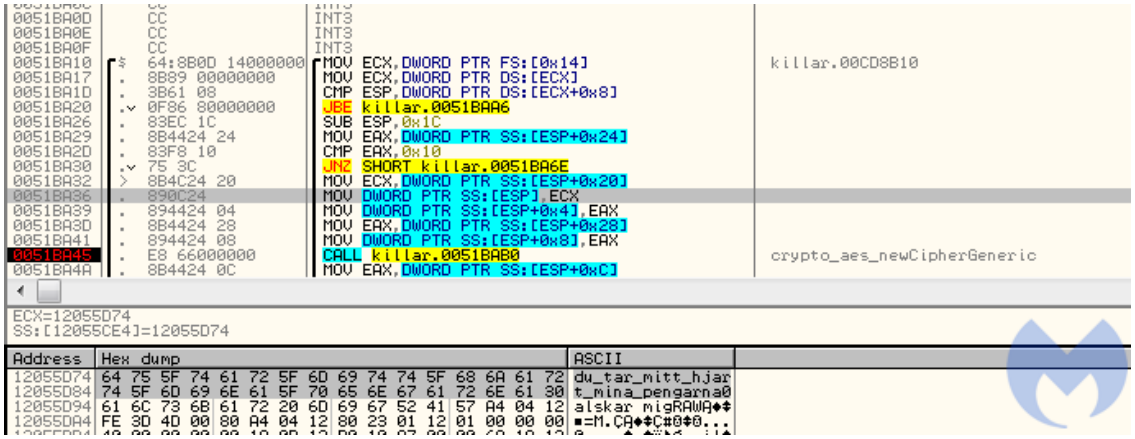
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

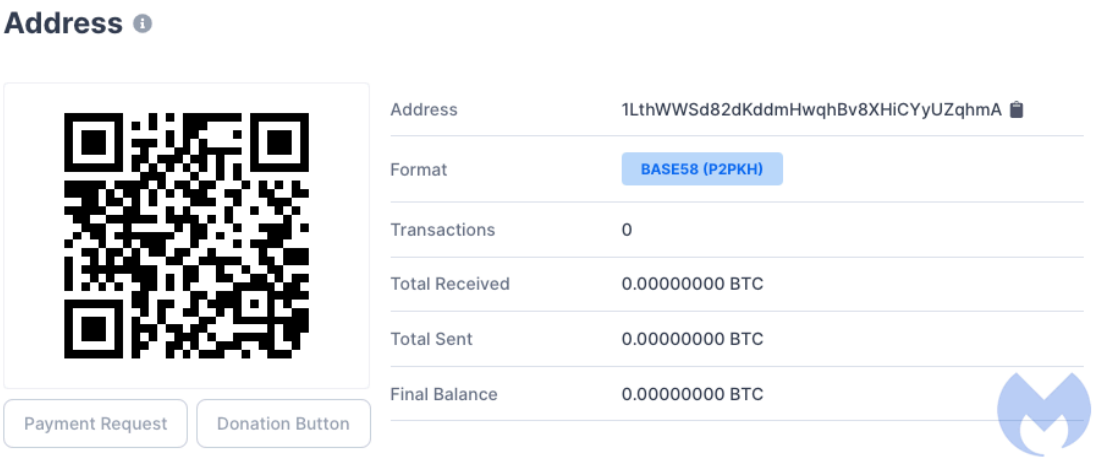
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

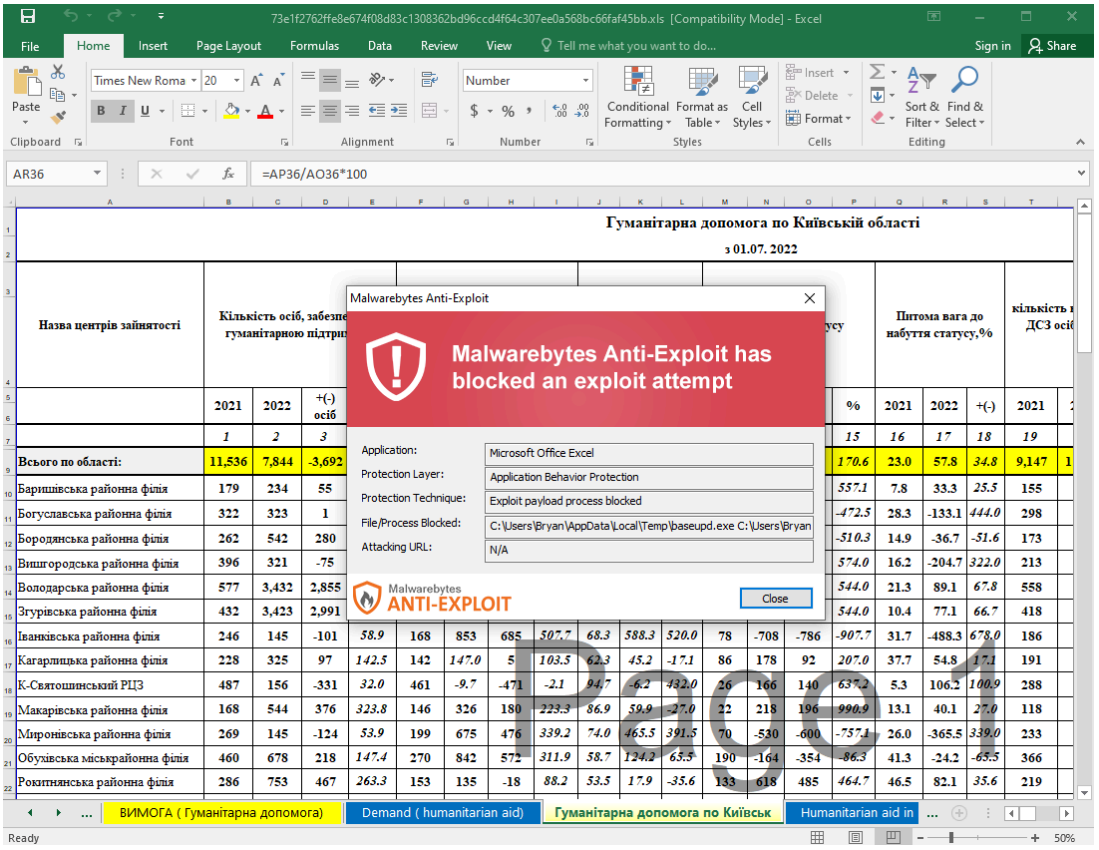
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

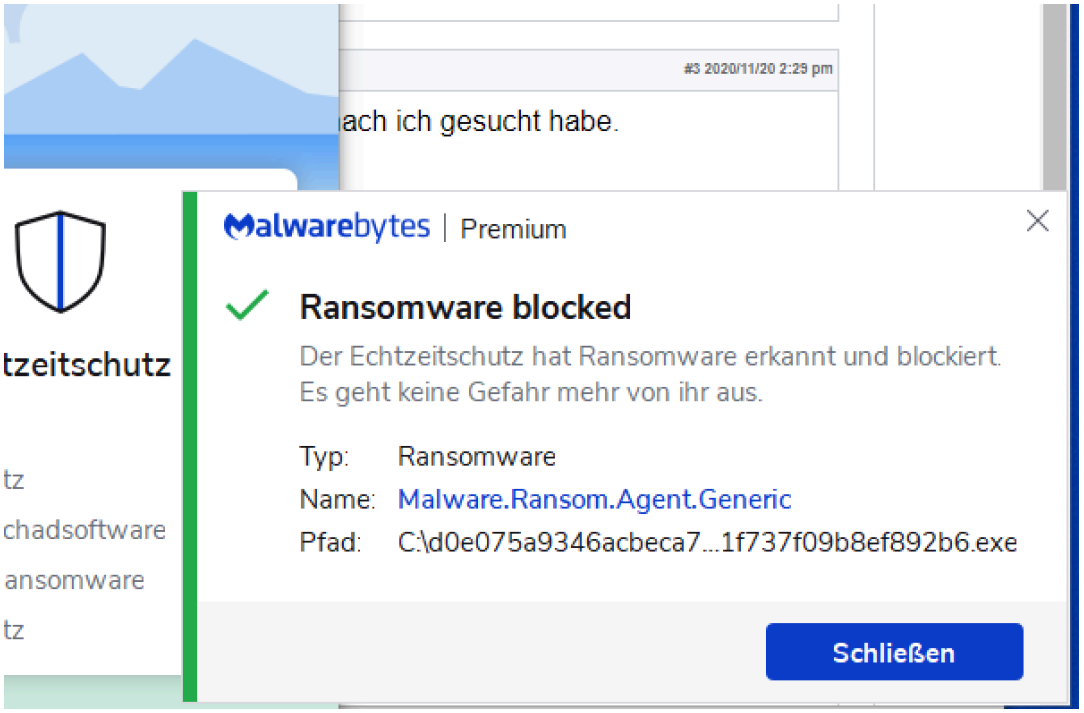
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

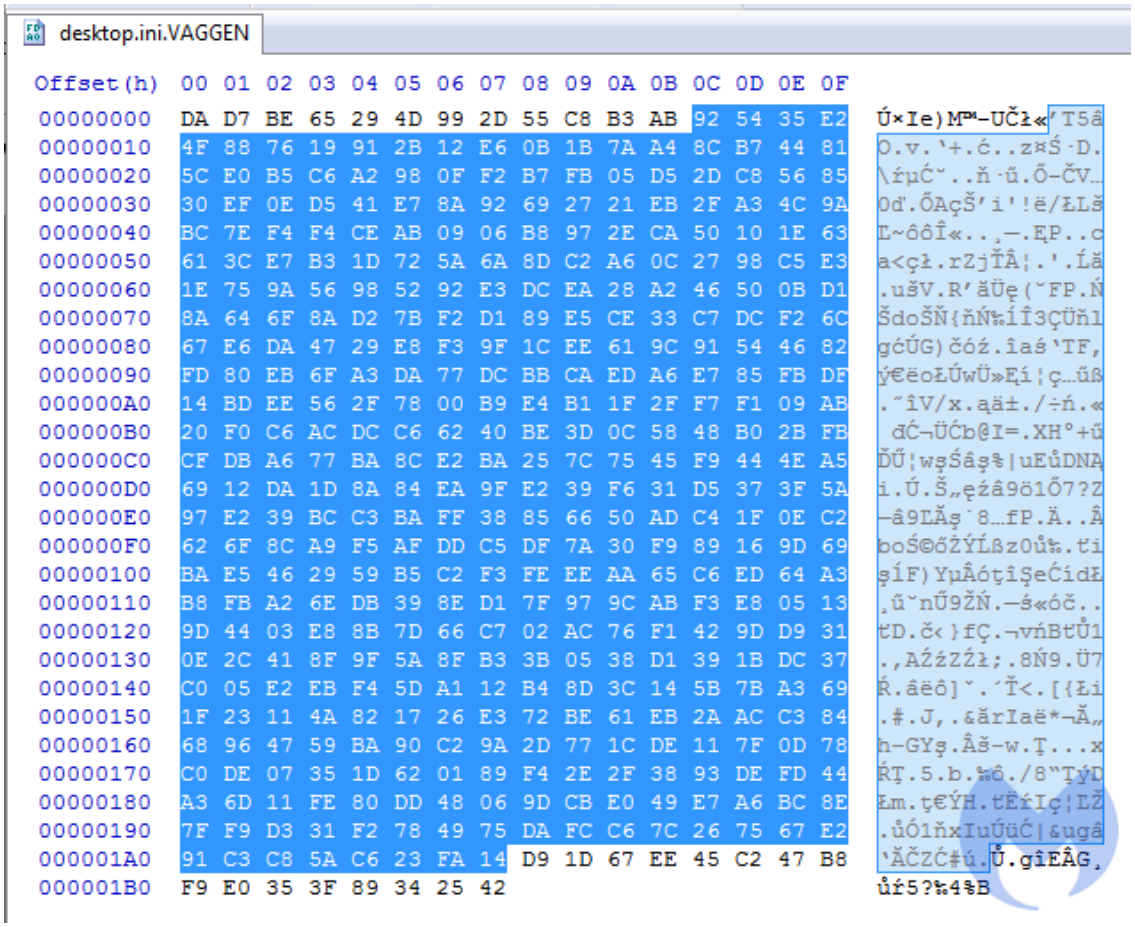
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

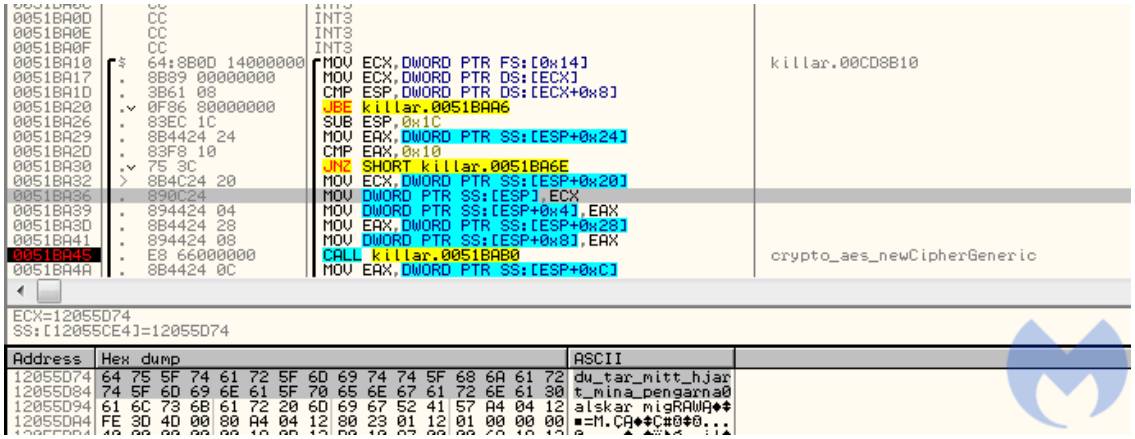


The content of the output file (with .VAGGEN extension) contains:

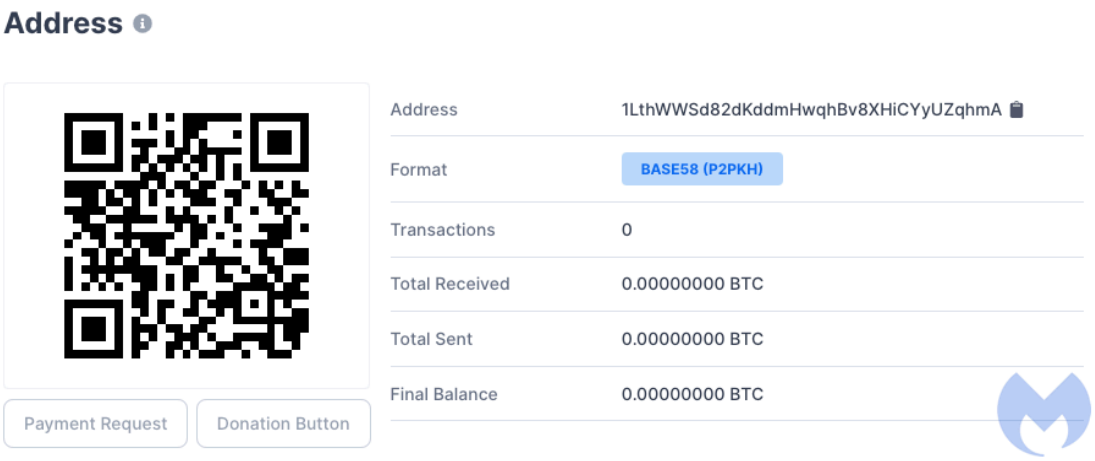
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

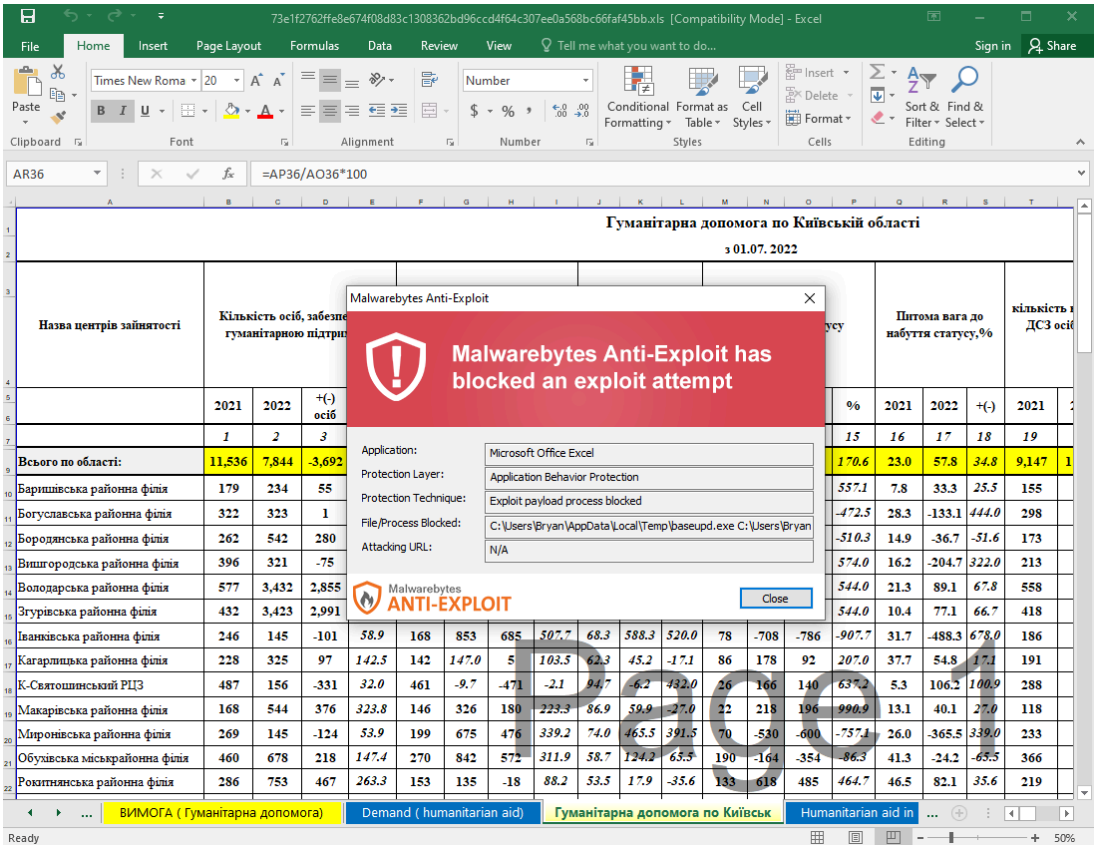
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

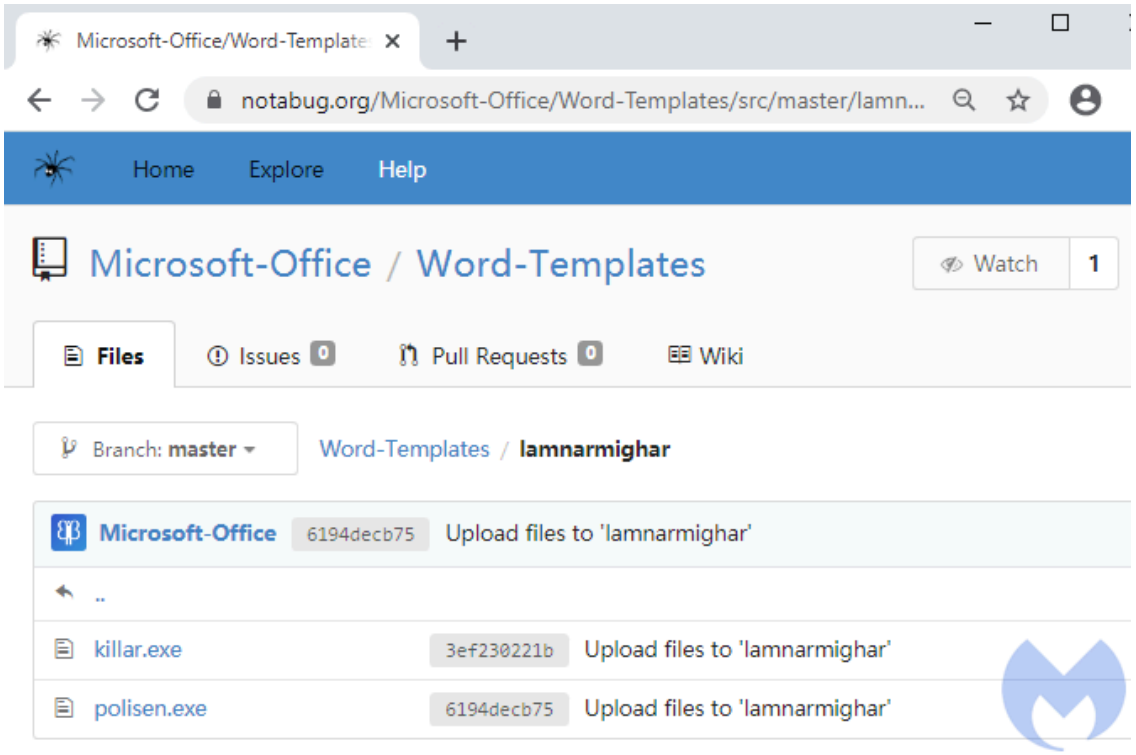
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



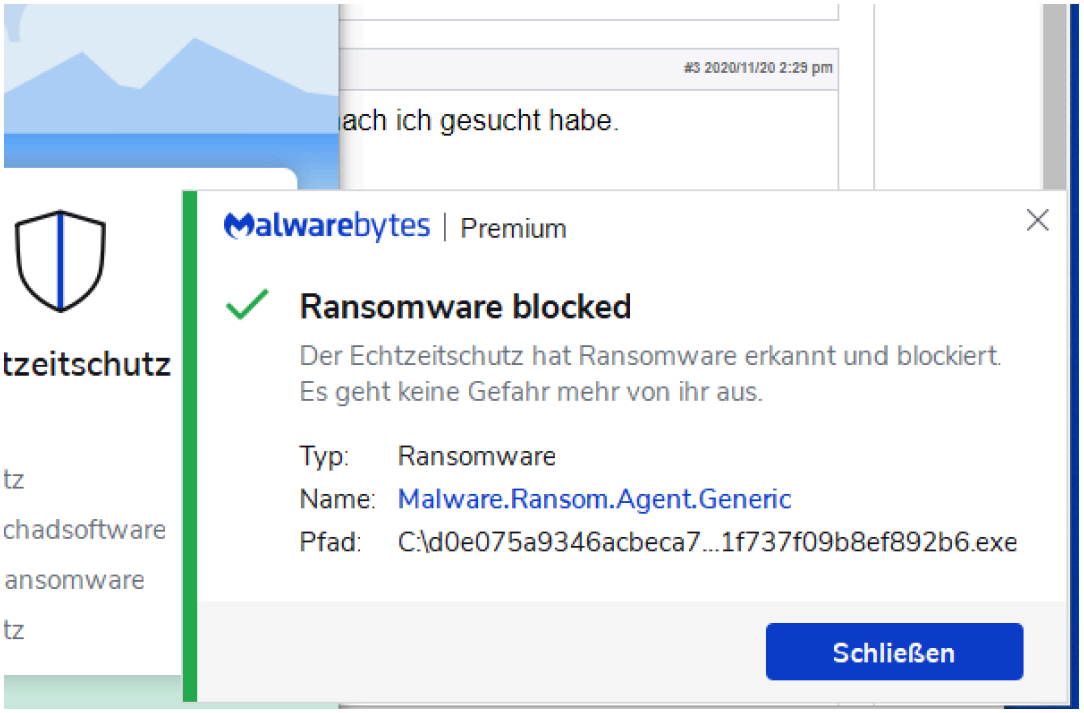
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

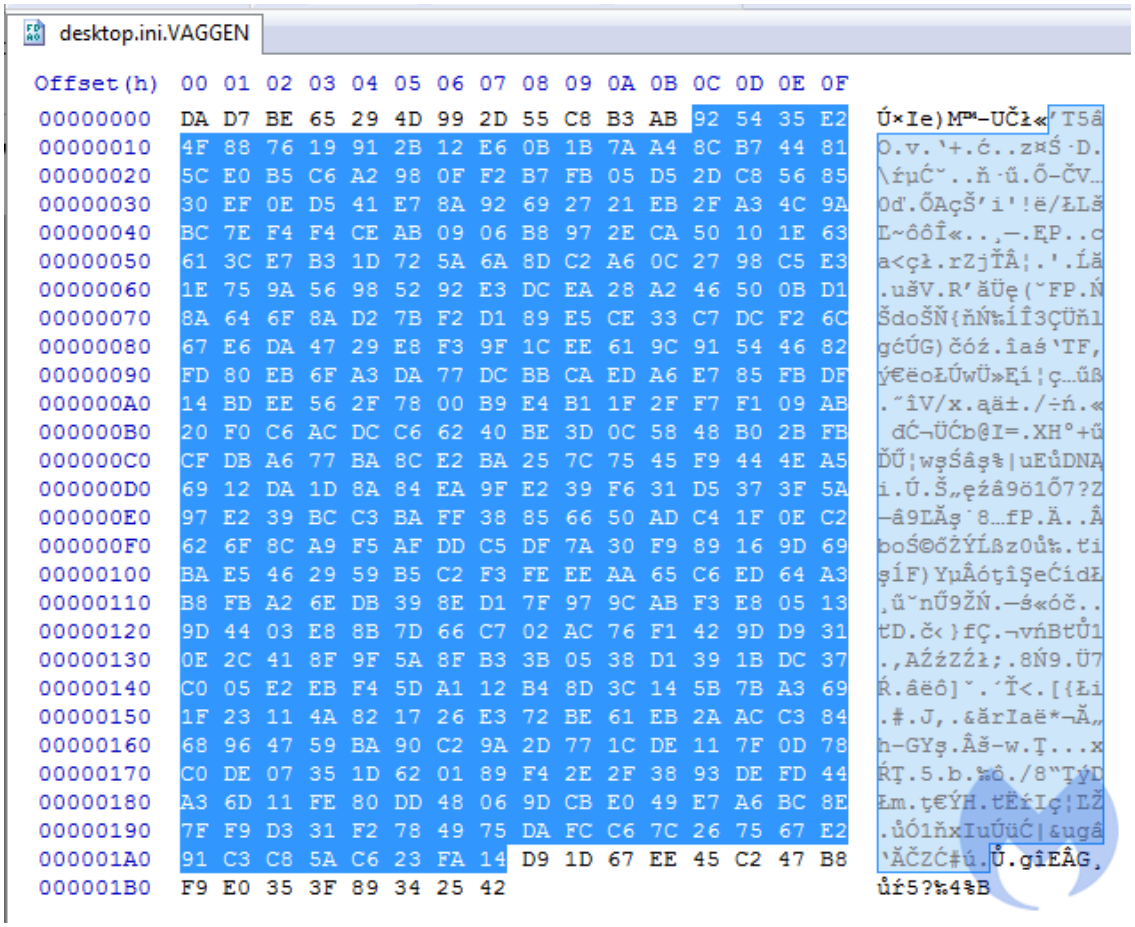
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

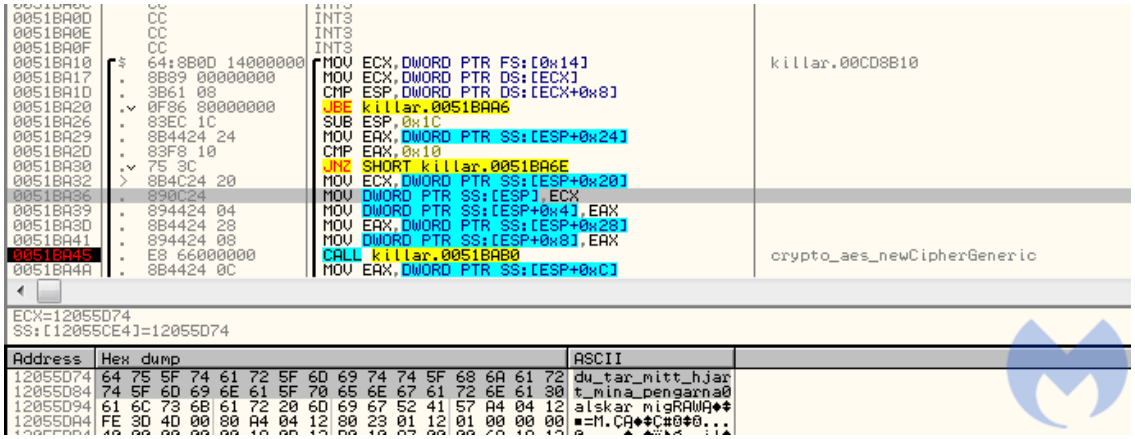


The content of the output file (with .VAGGEN extension) contains:

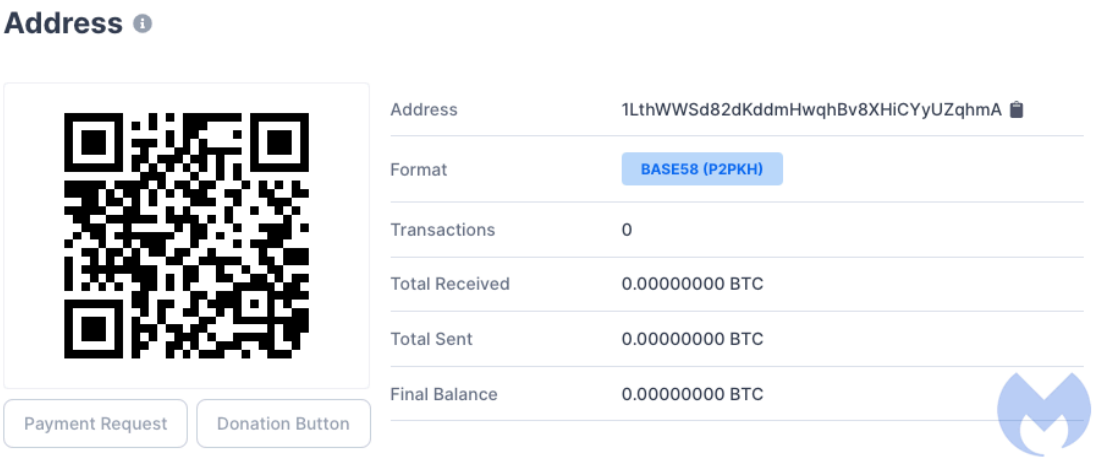
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

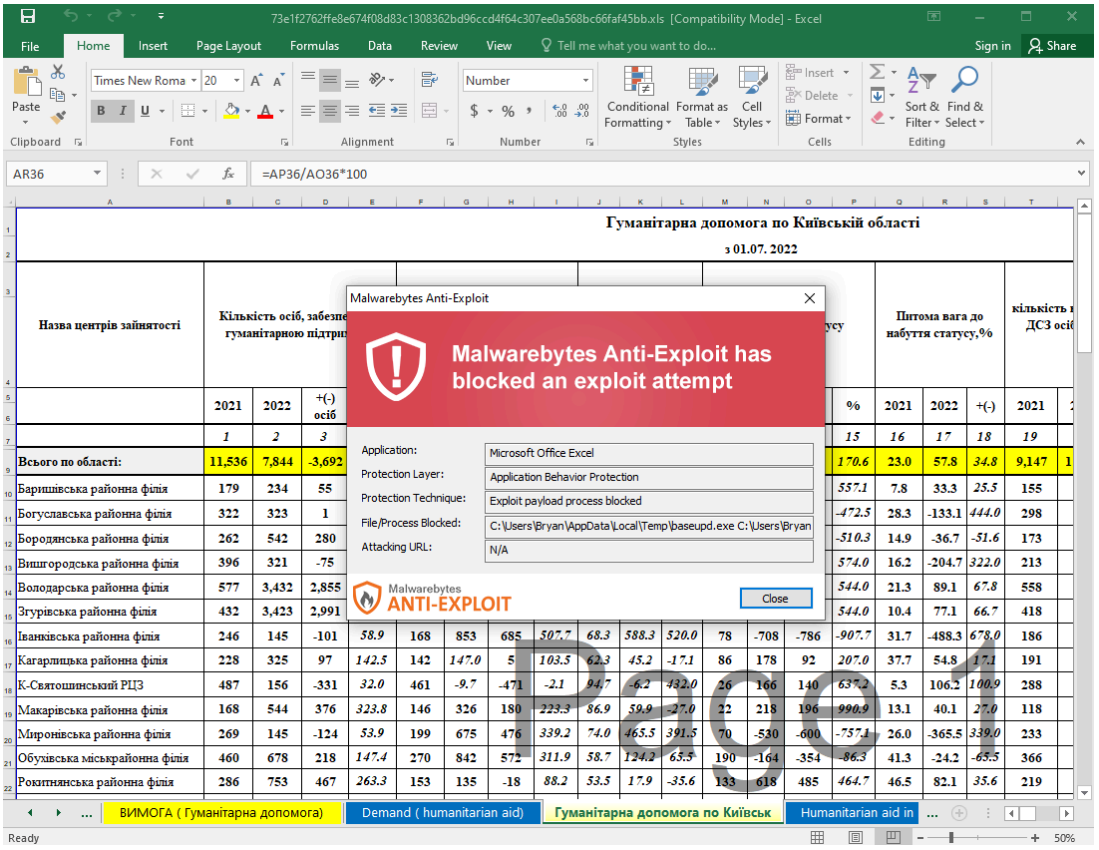
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

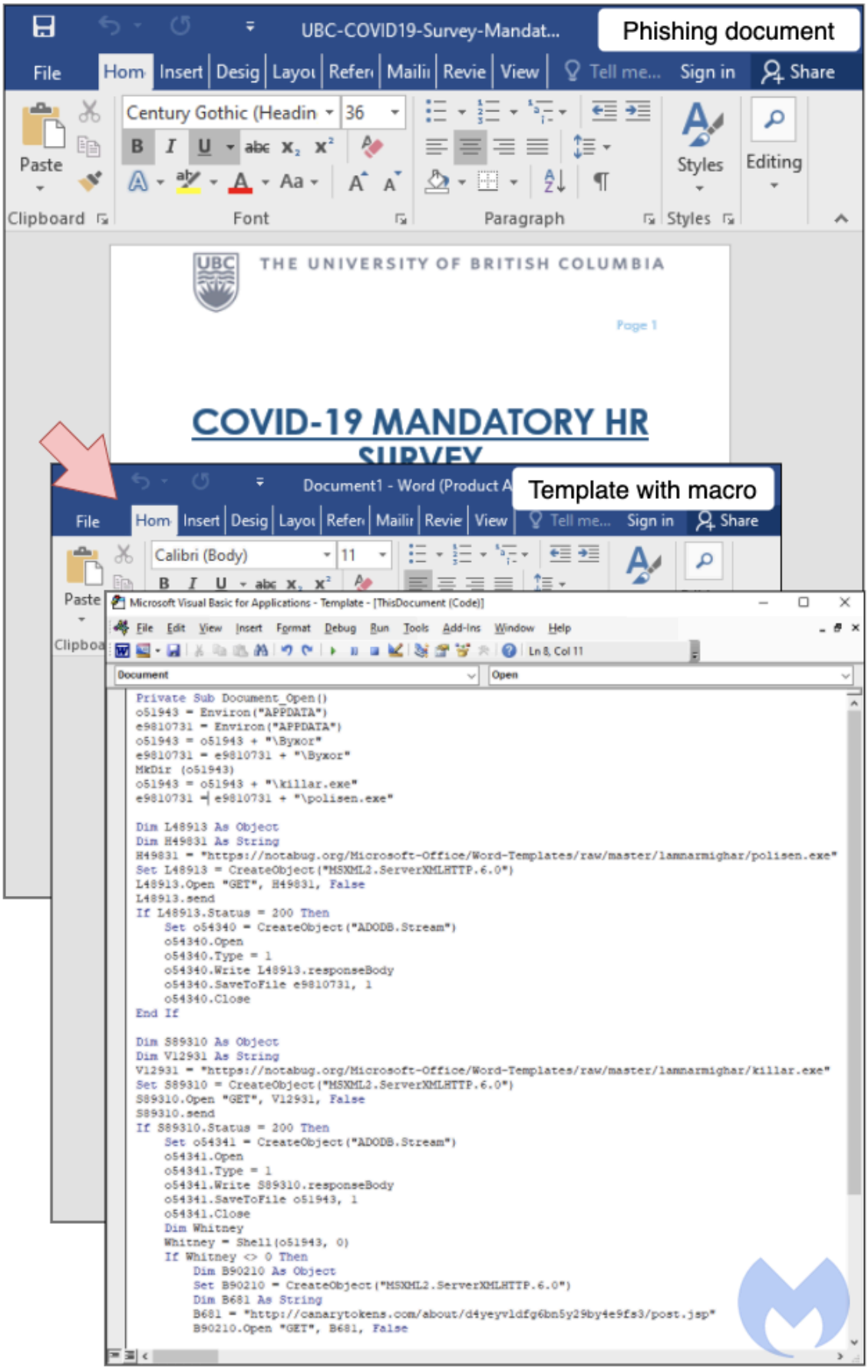
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

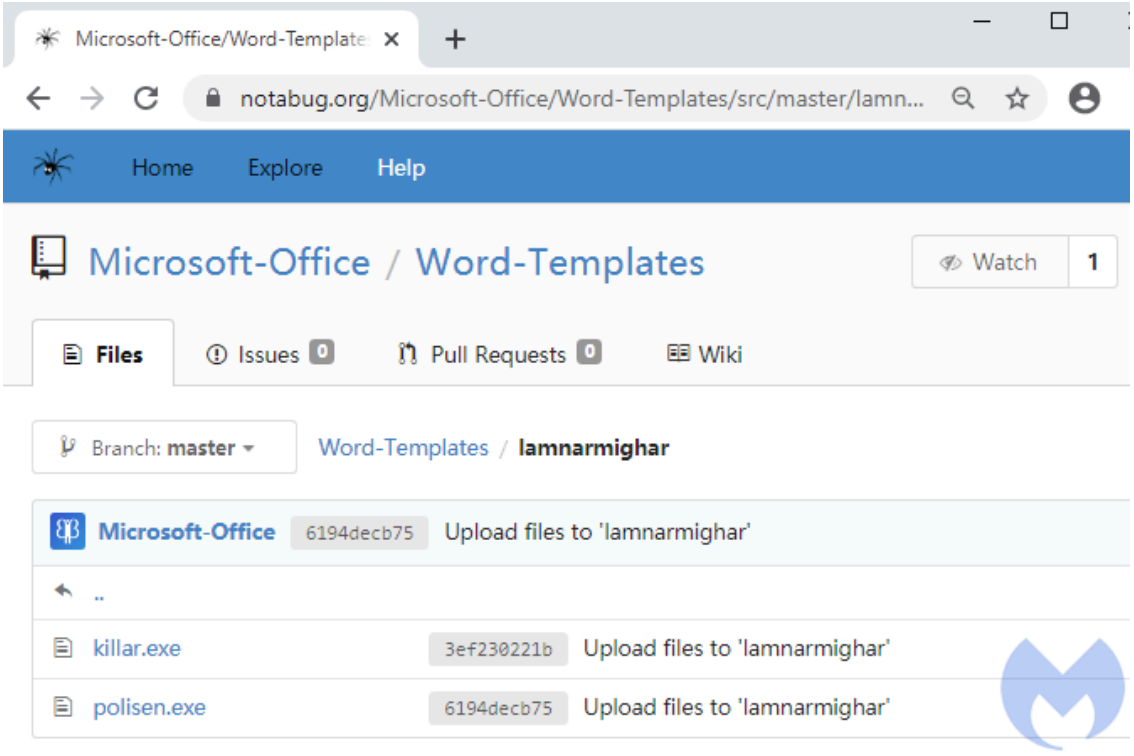
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



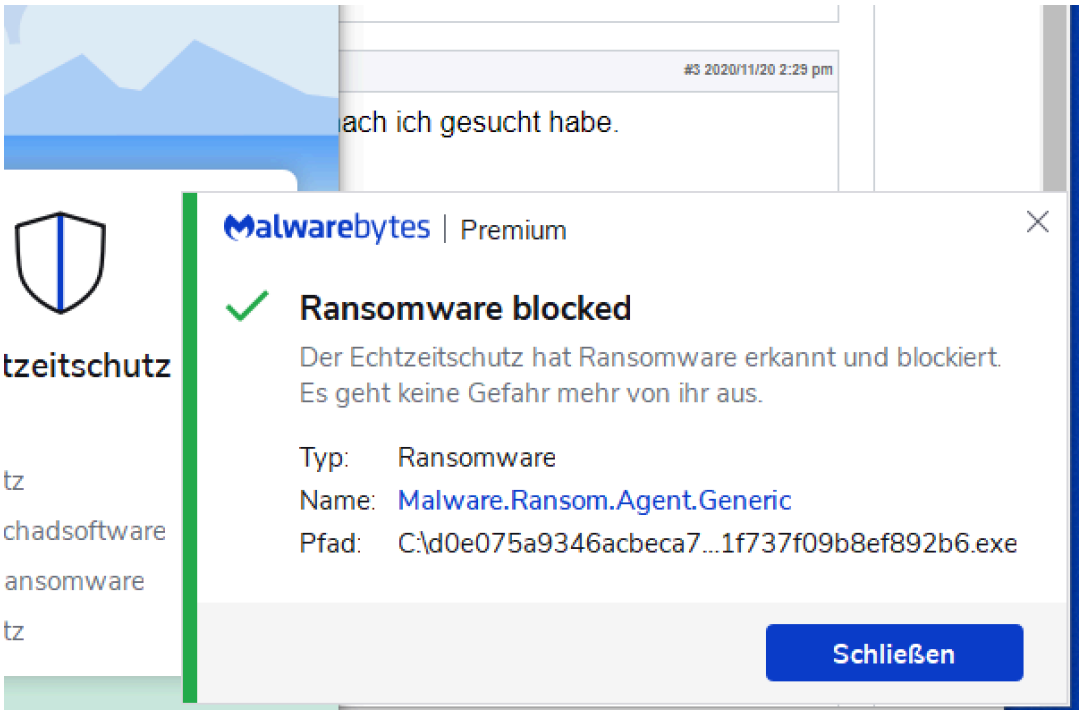
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

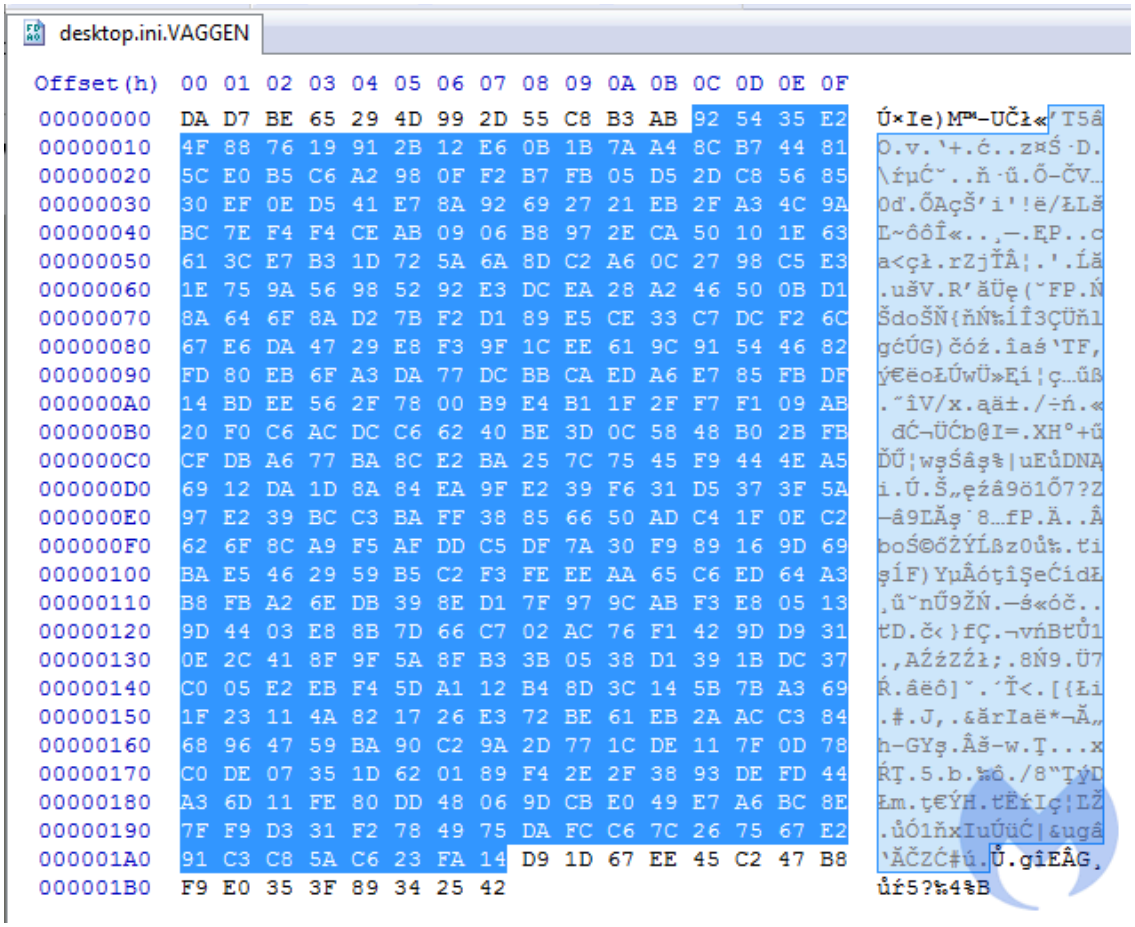
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



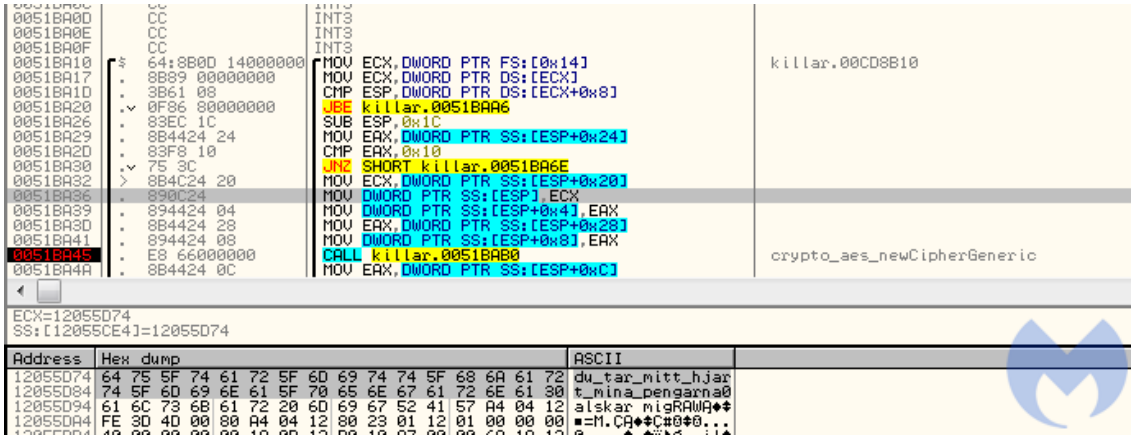
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

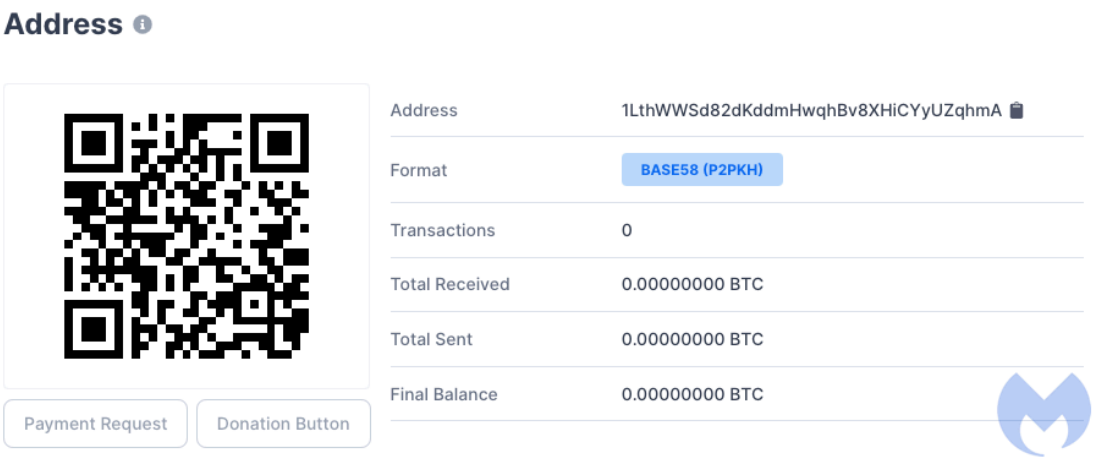
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

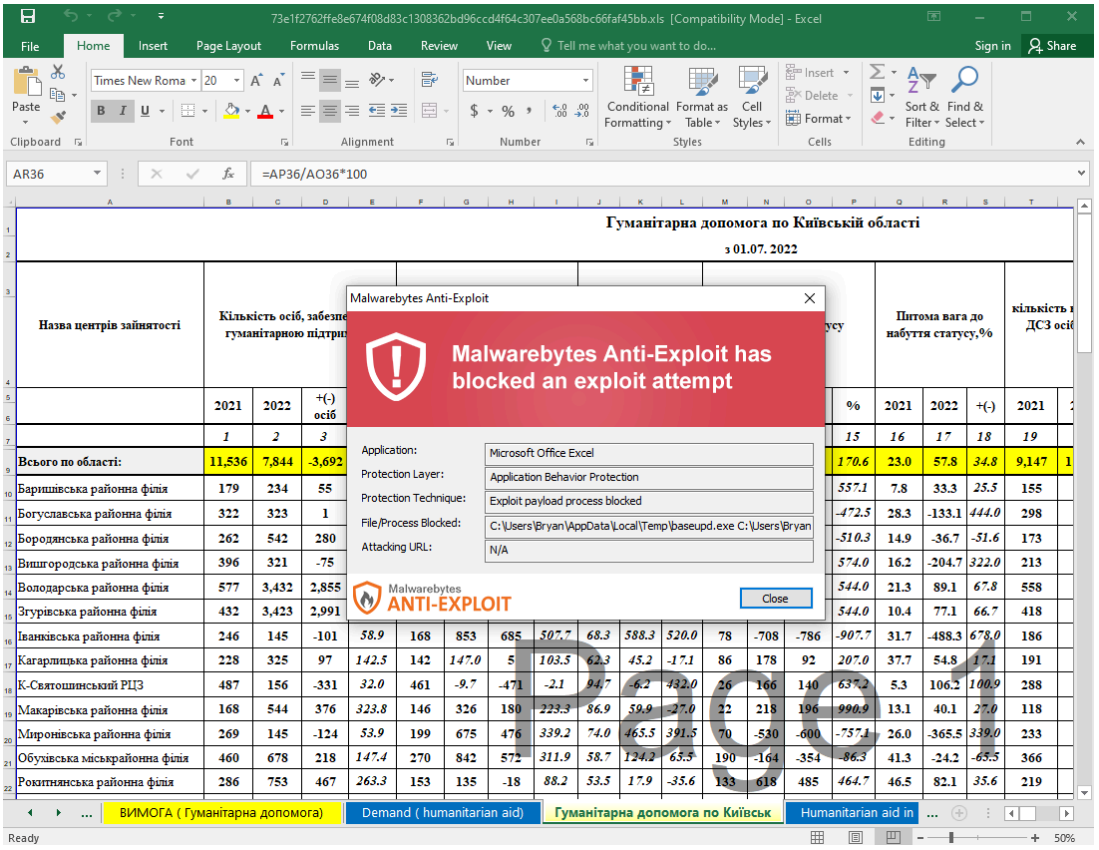
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

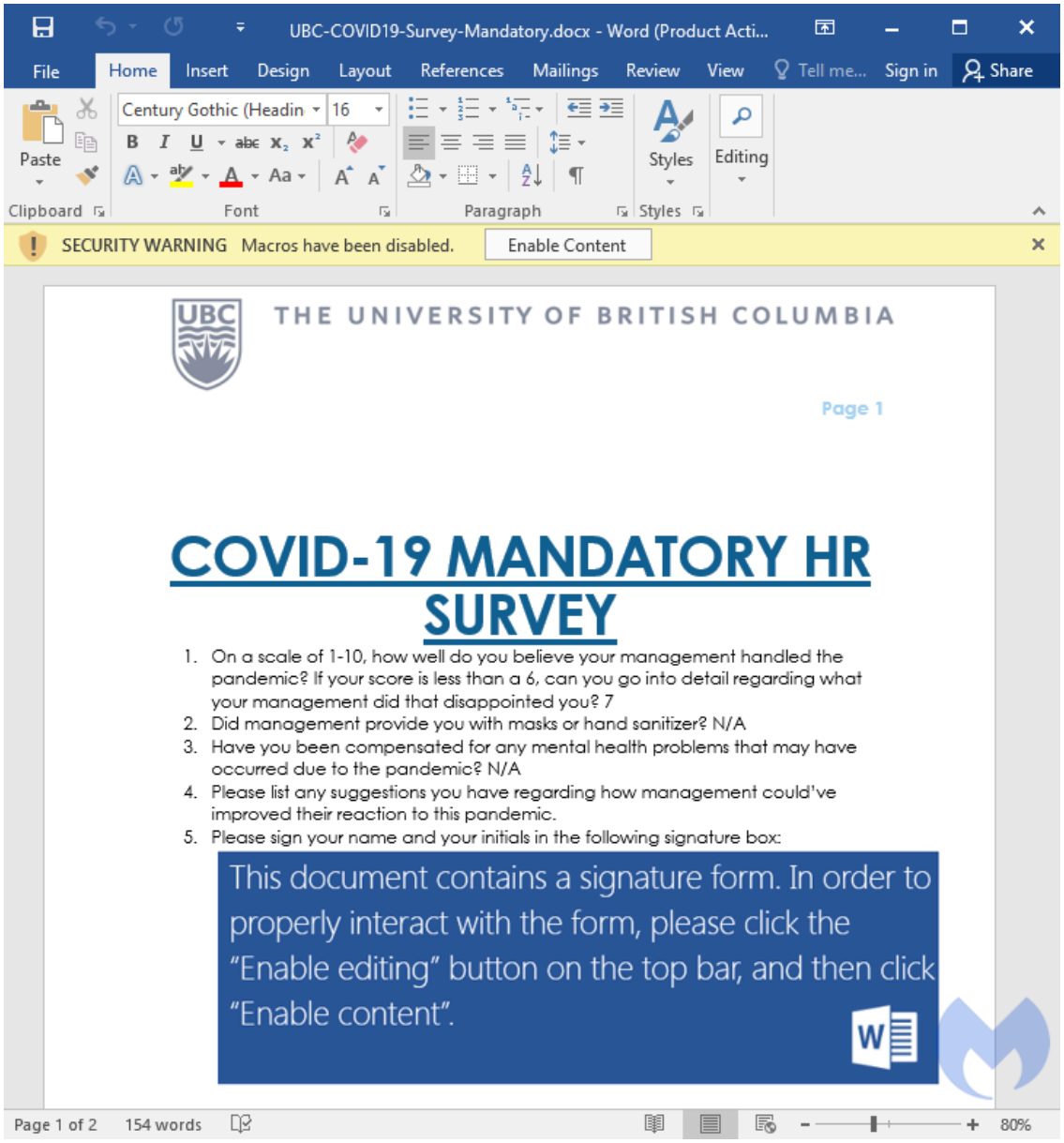
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

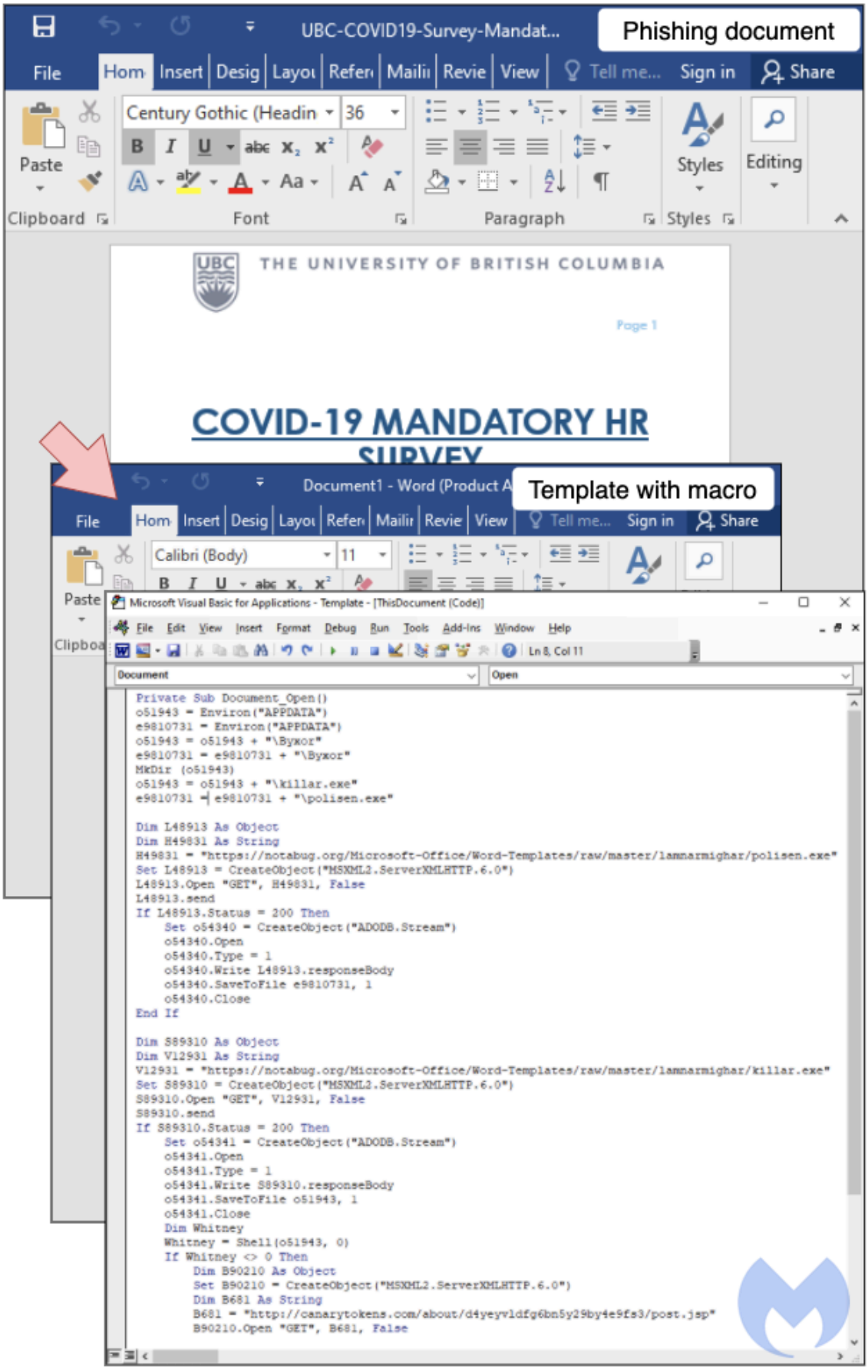
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

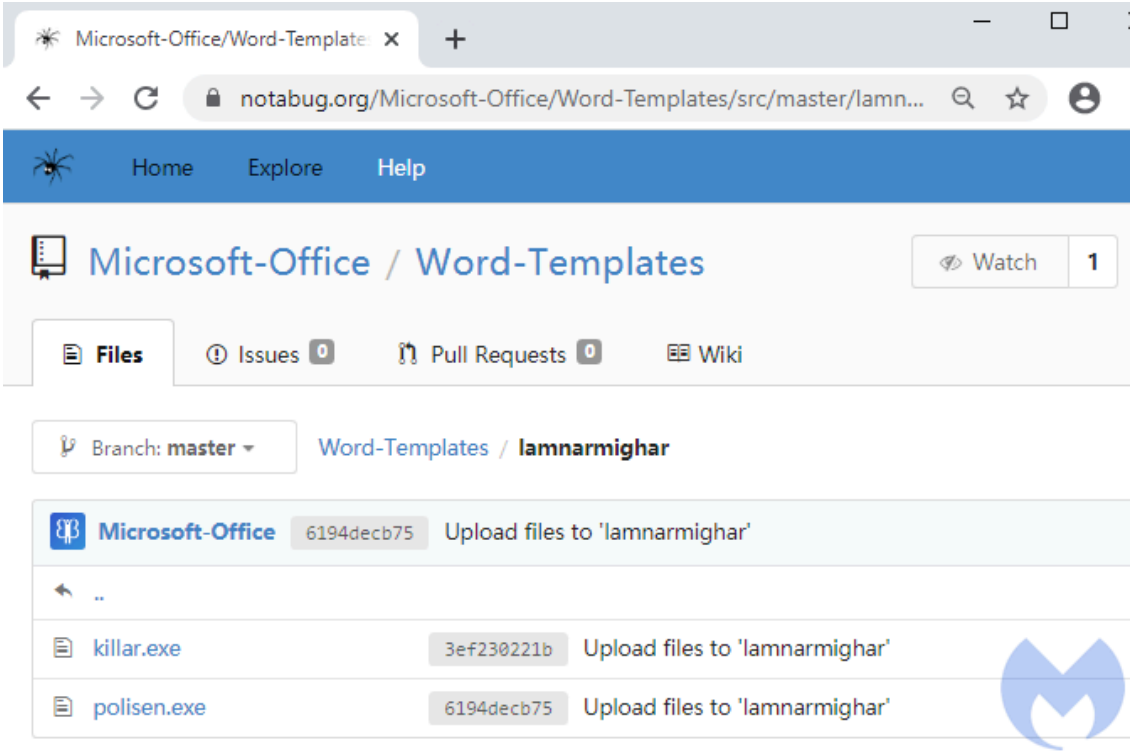
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



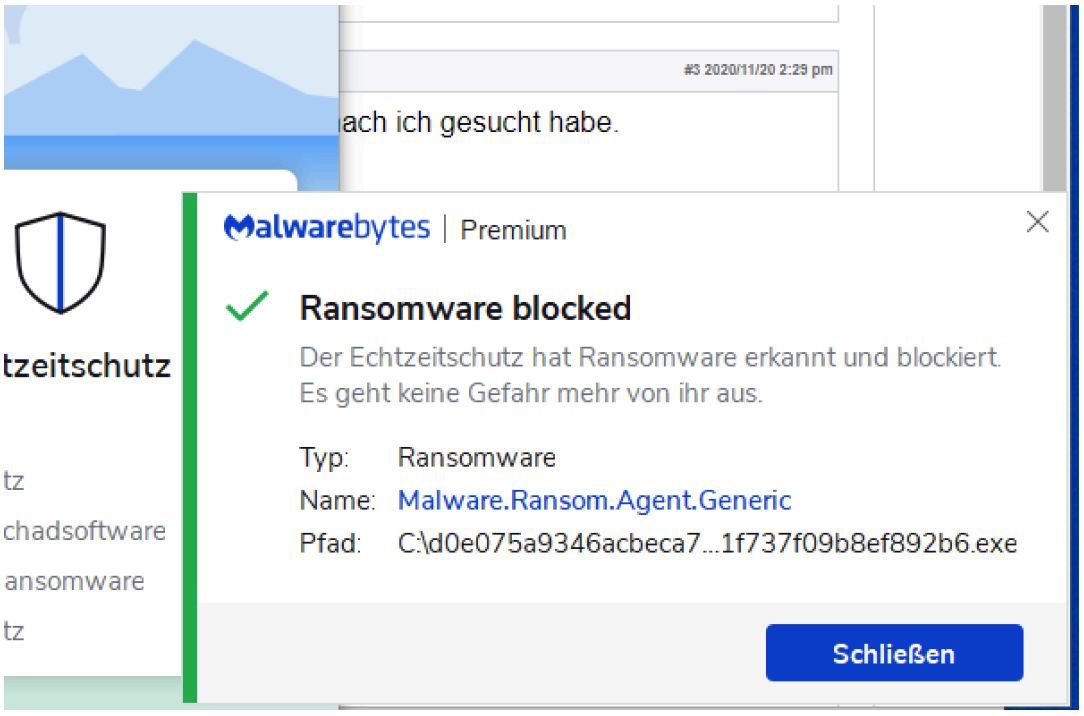
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

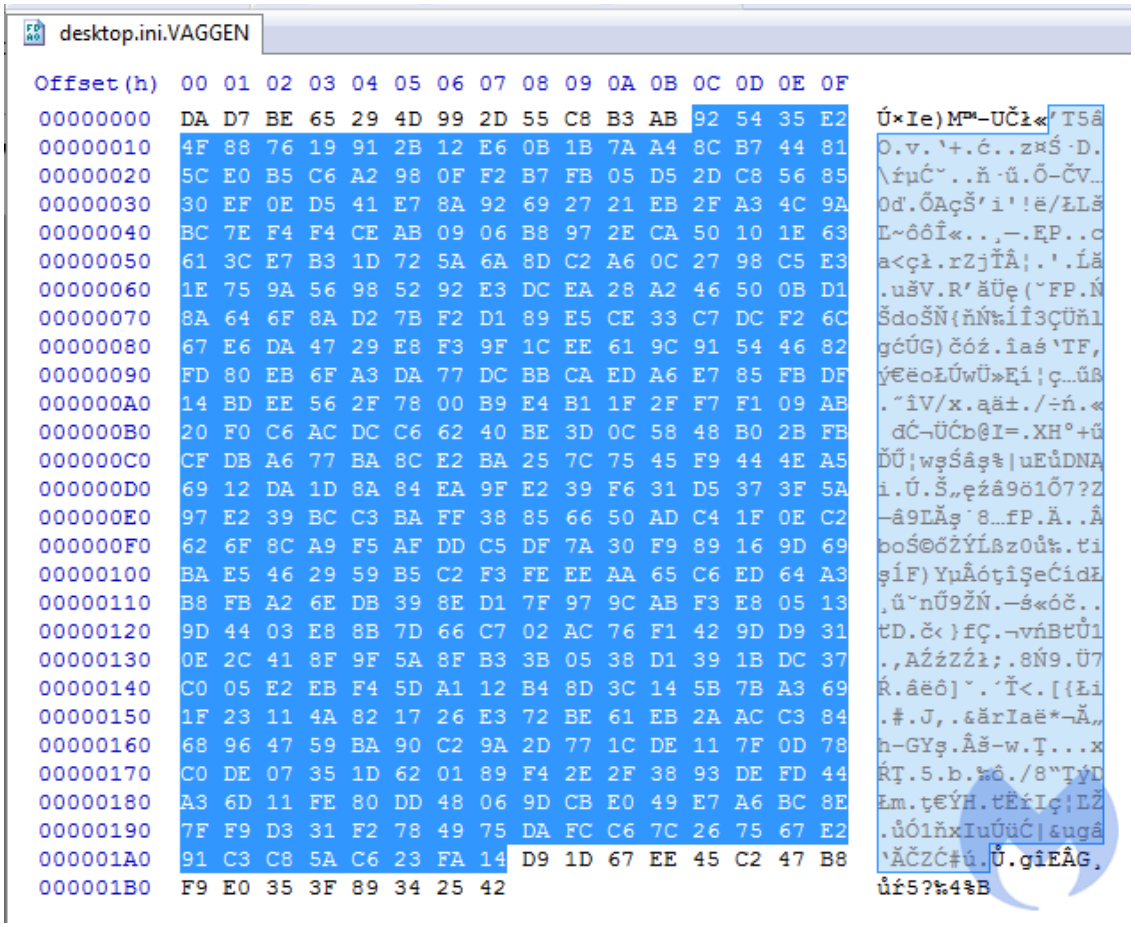
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



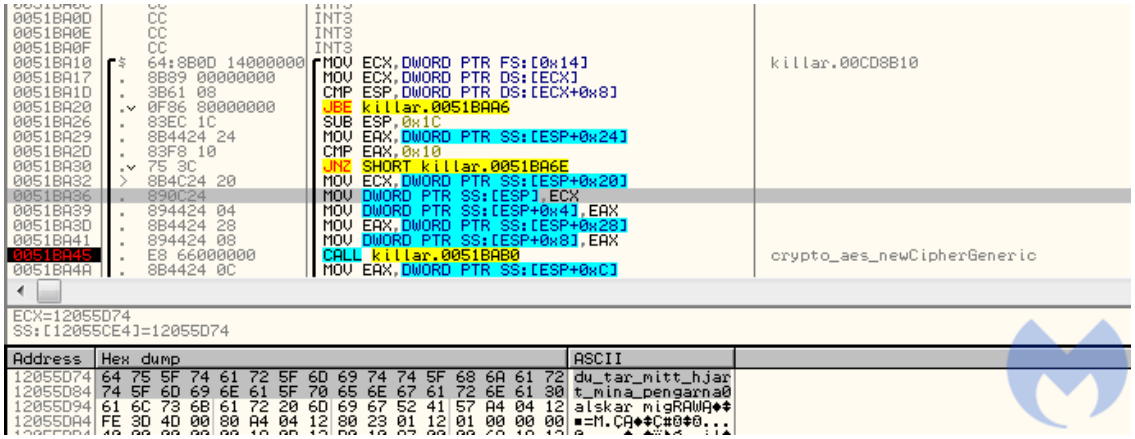
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

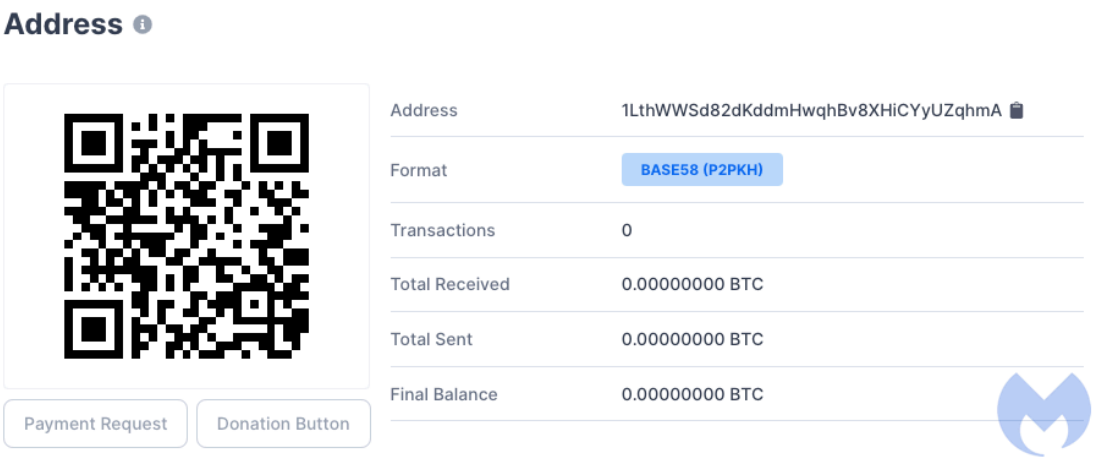
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

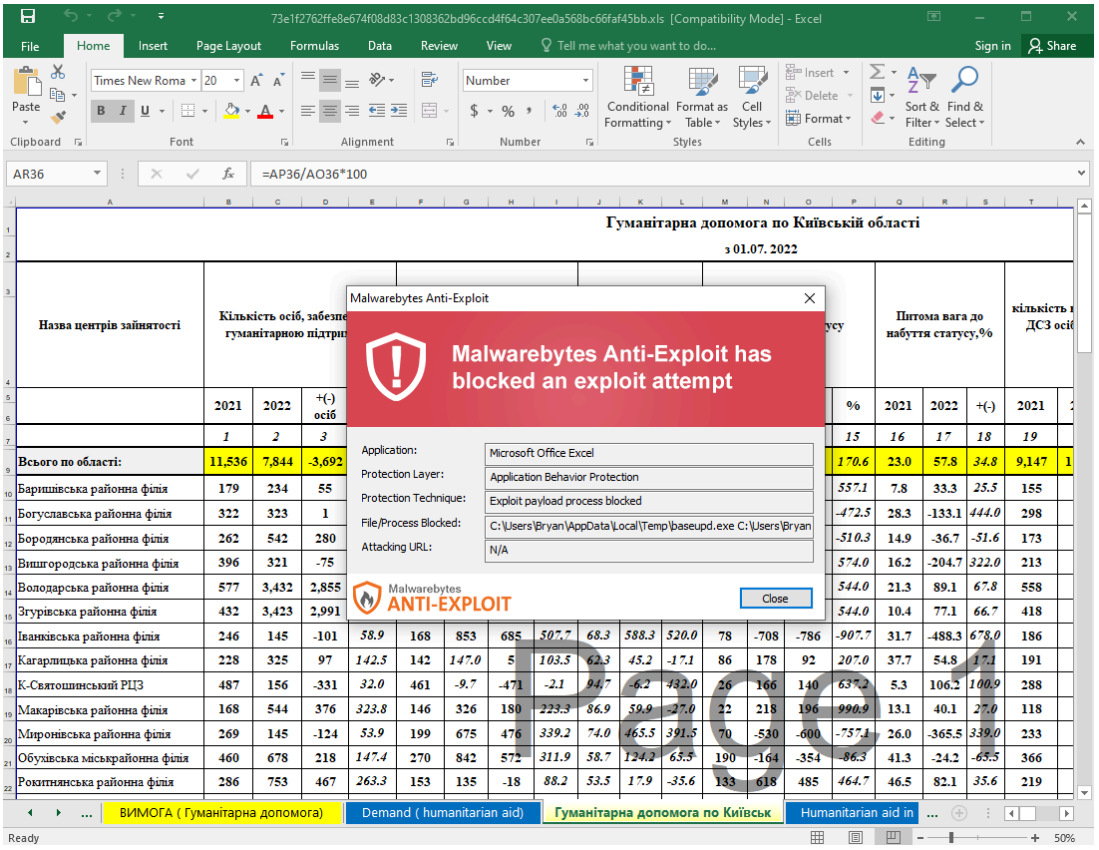
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

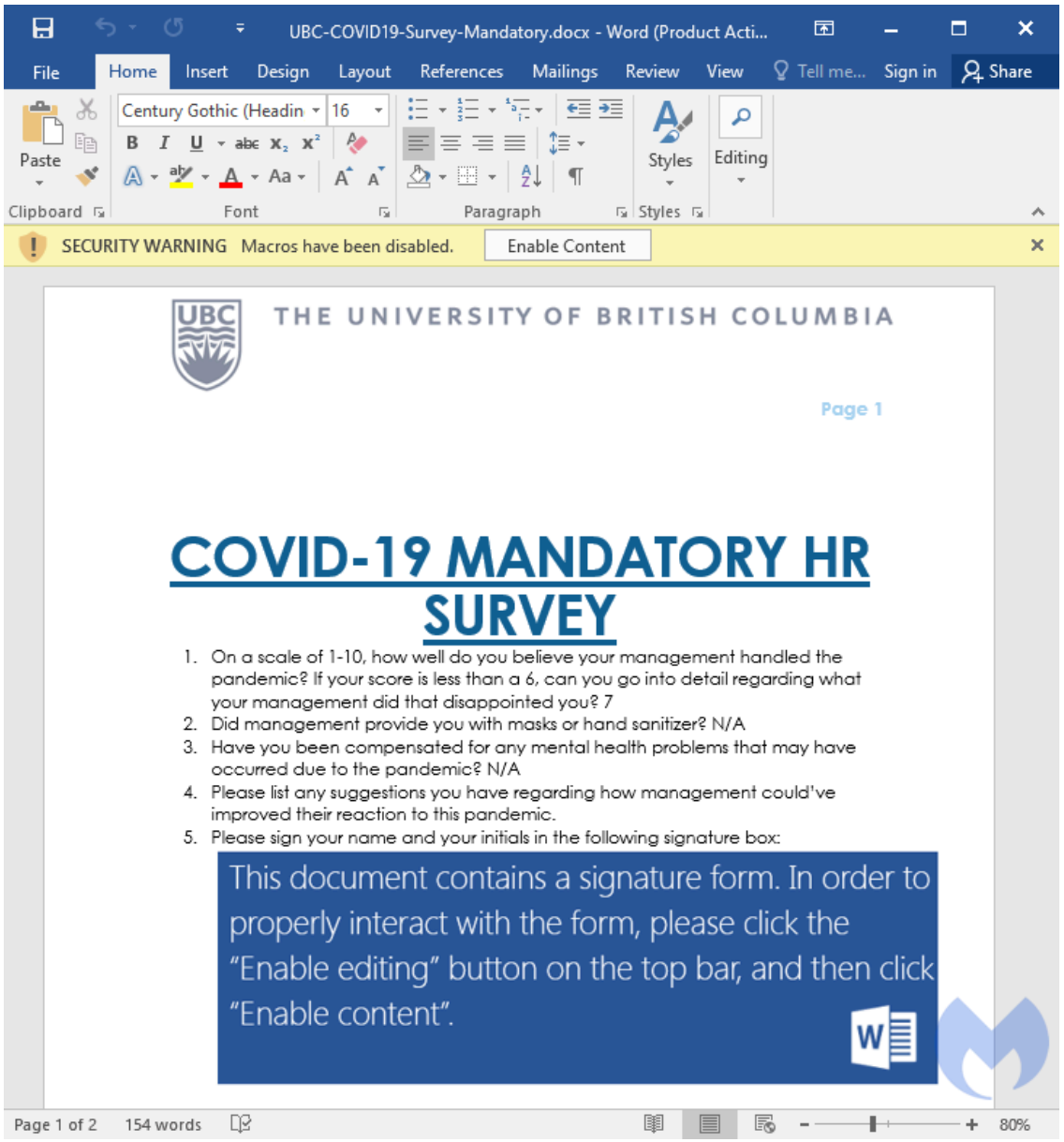
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

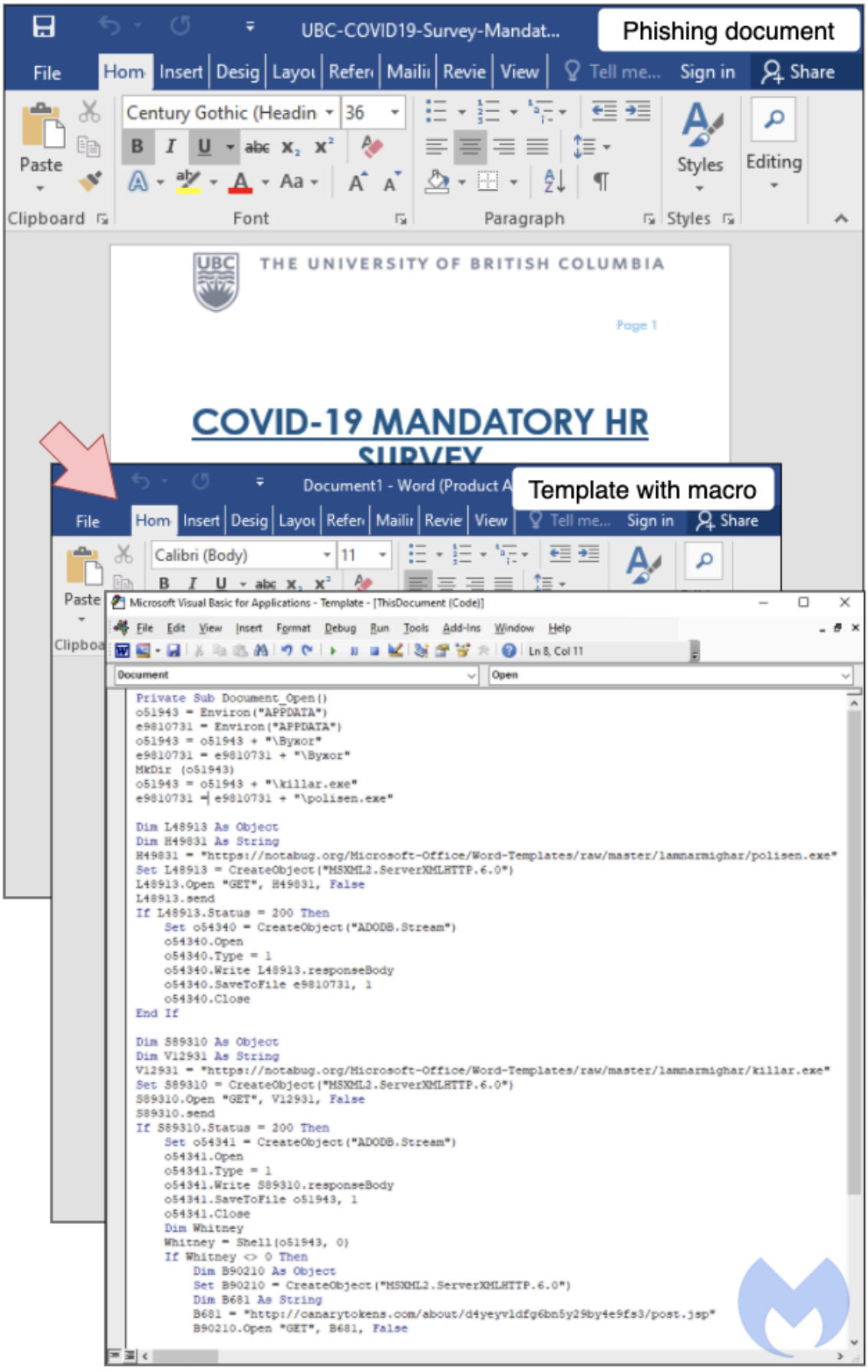
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

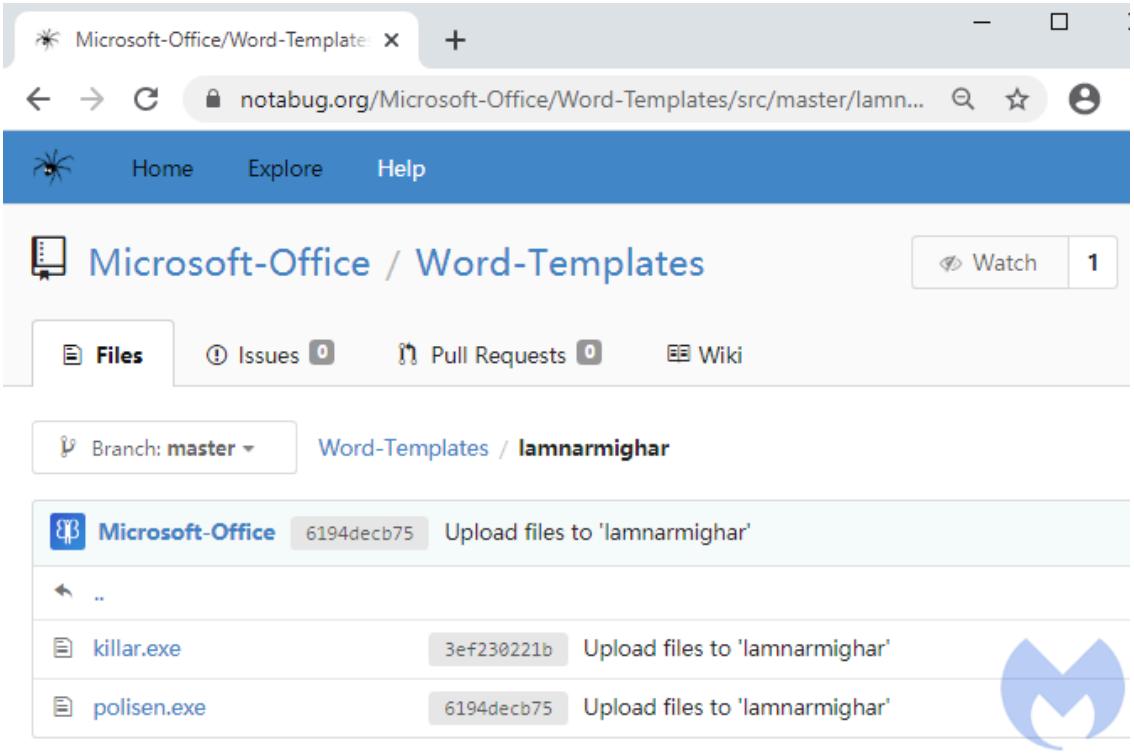
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



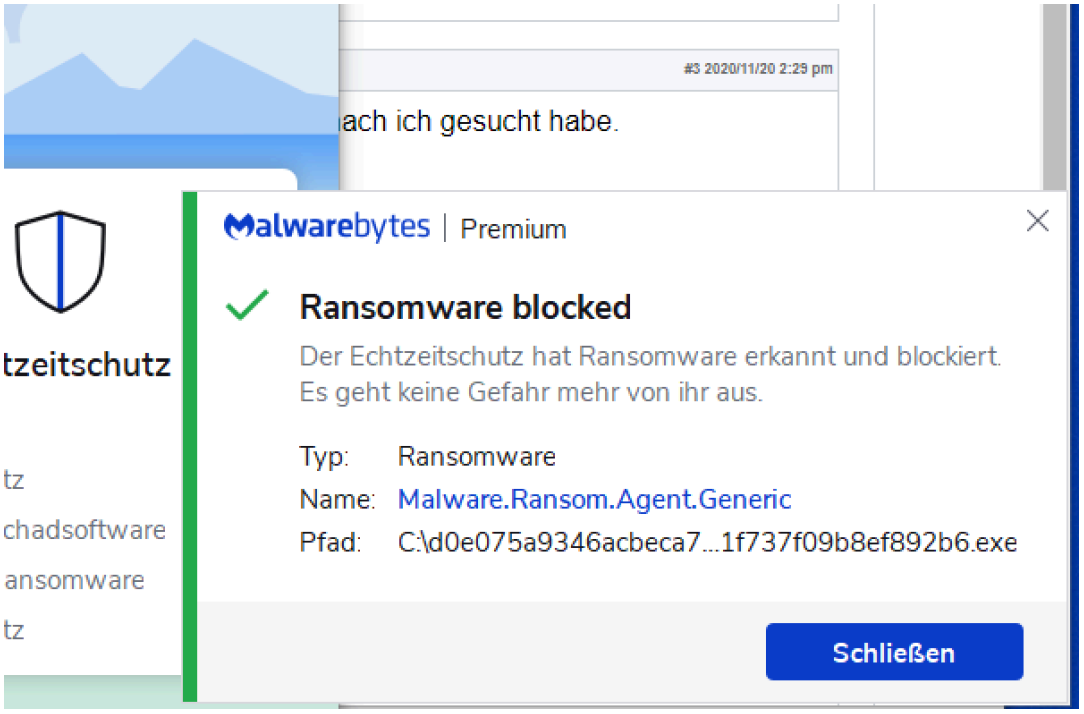
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

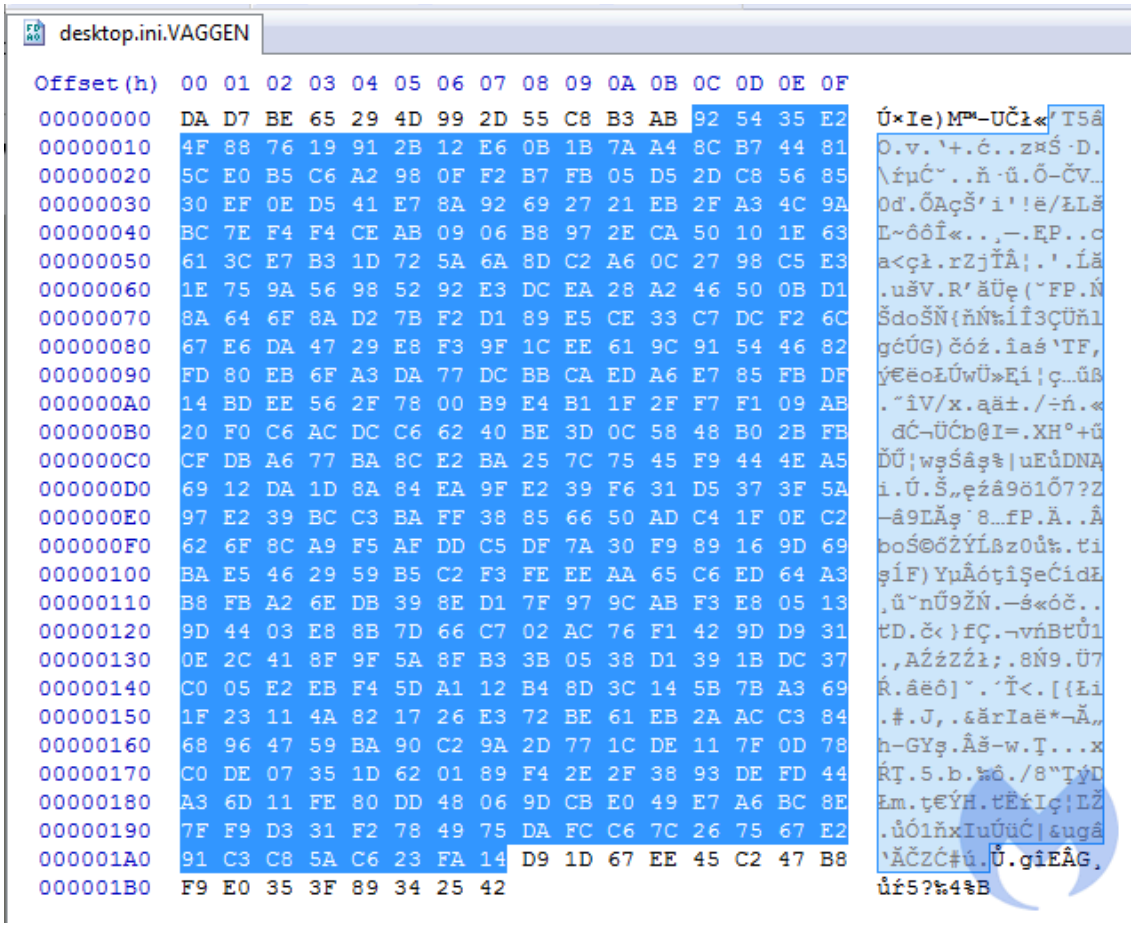
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

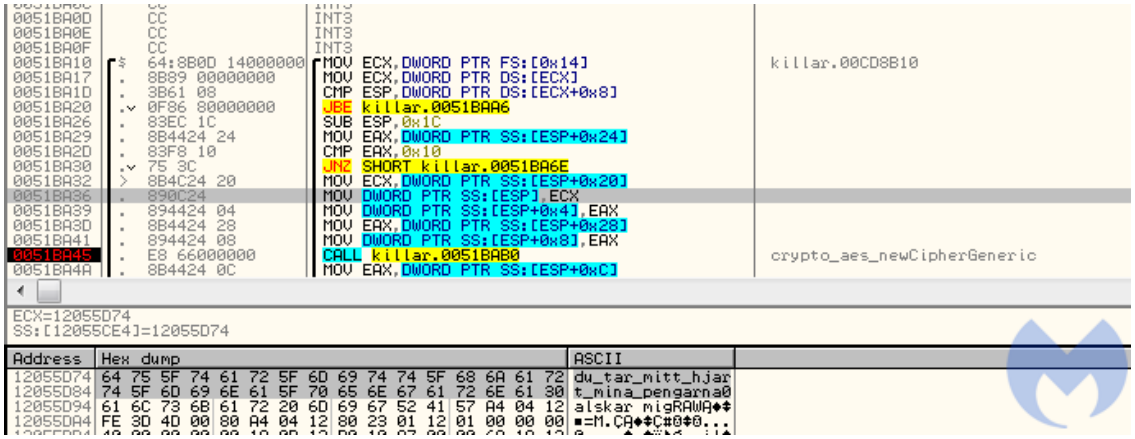


The content of the output file (with .VAGGEN extension) contains:

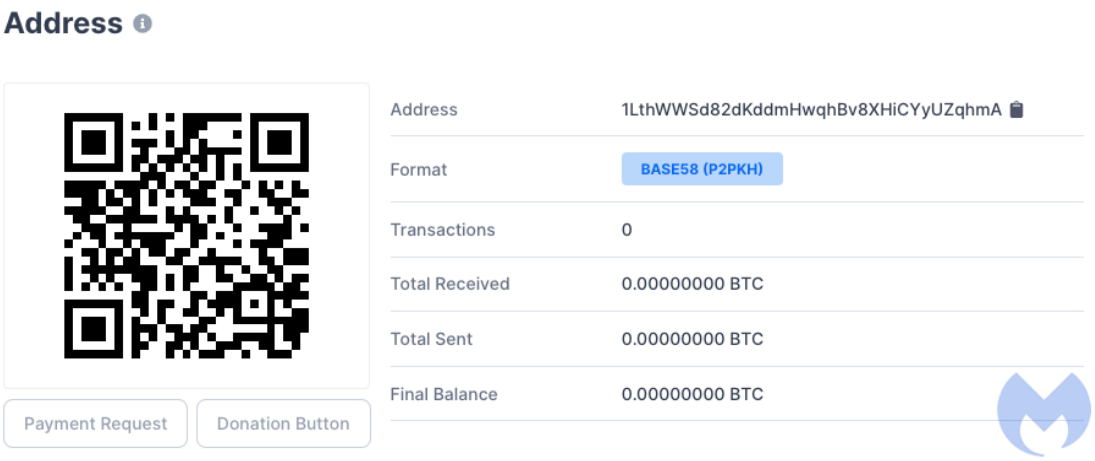
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

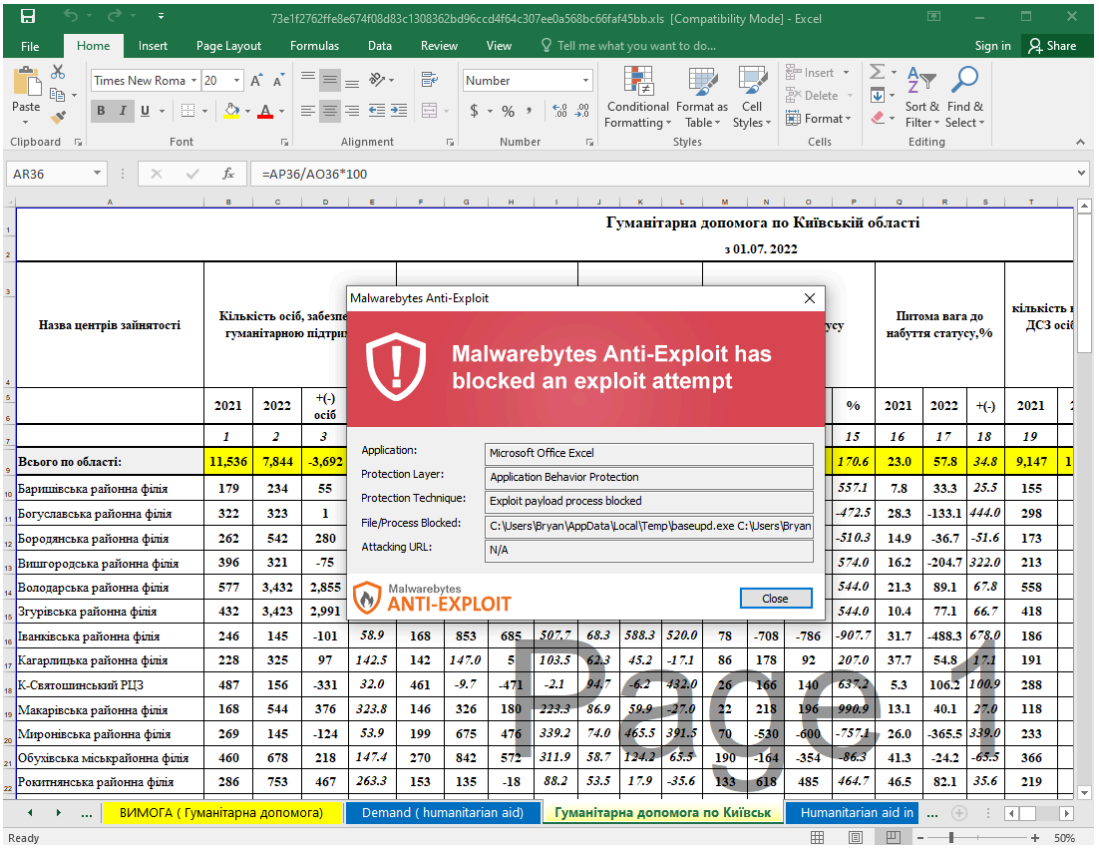
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

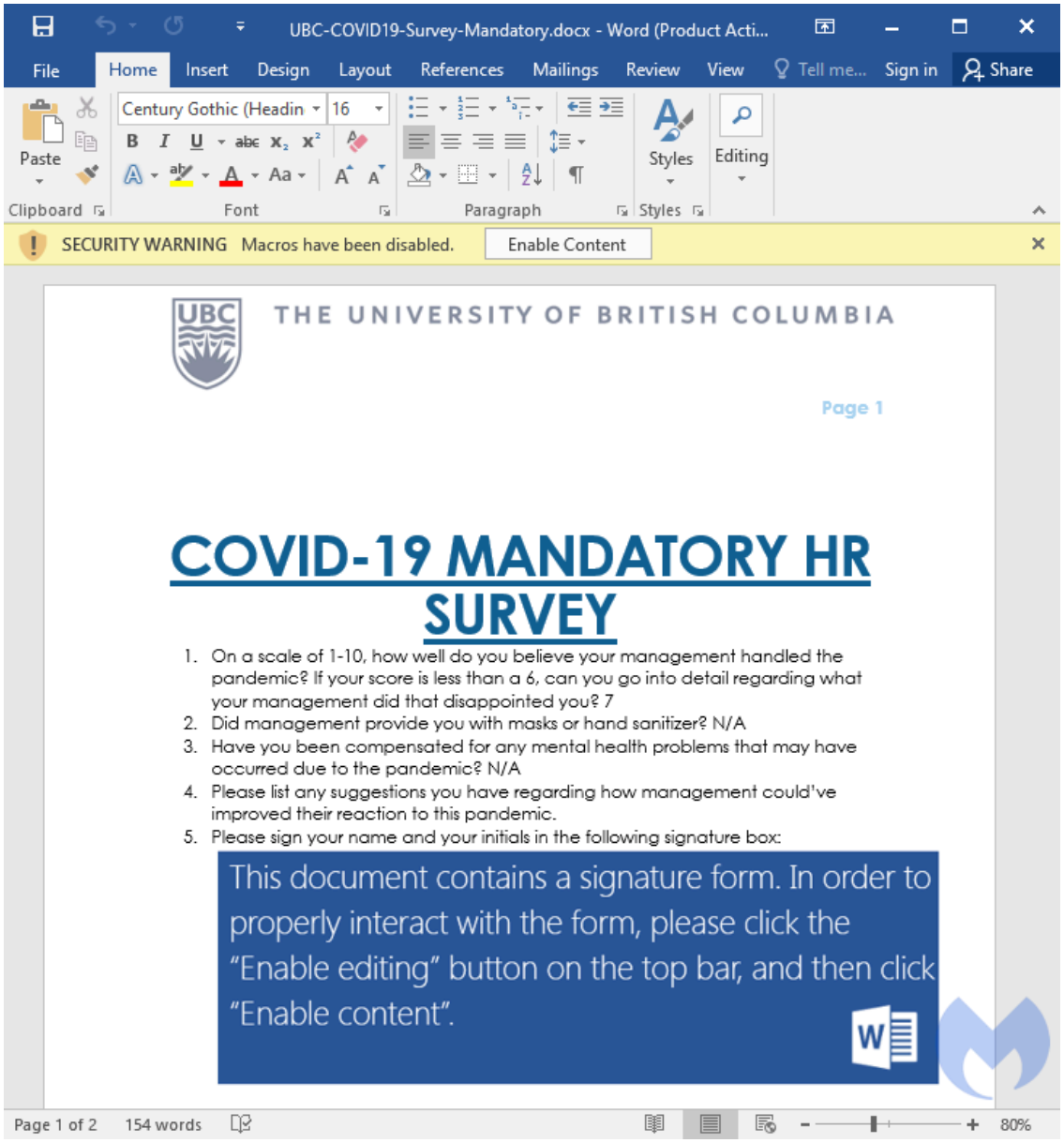
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

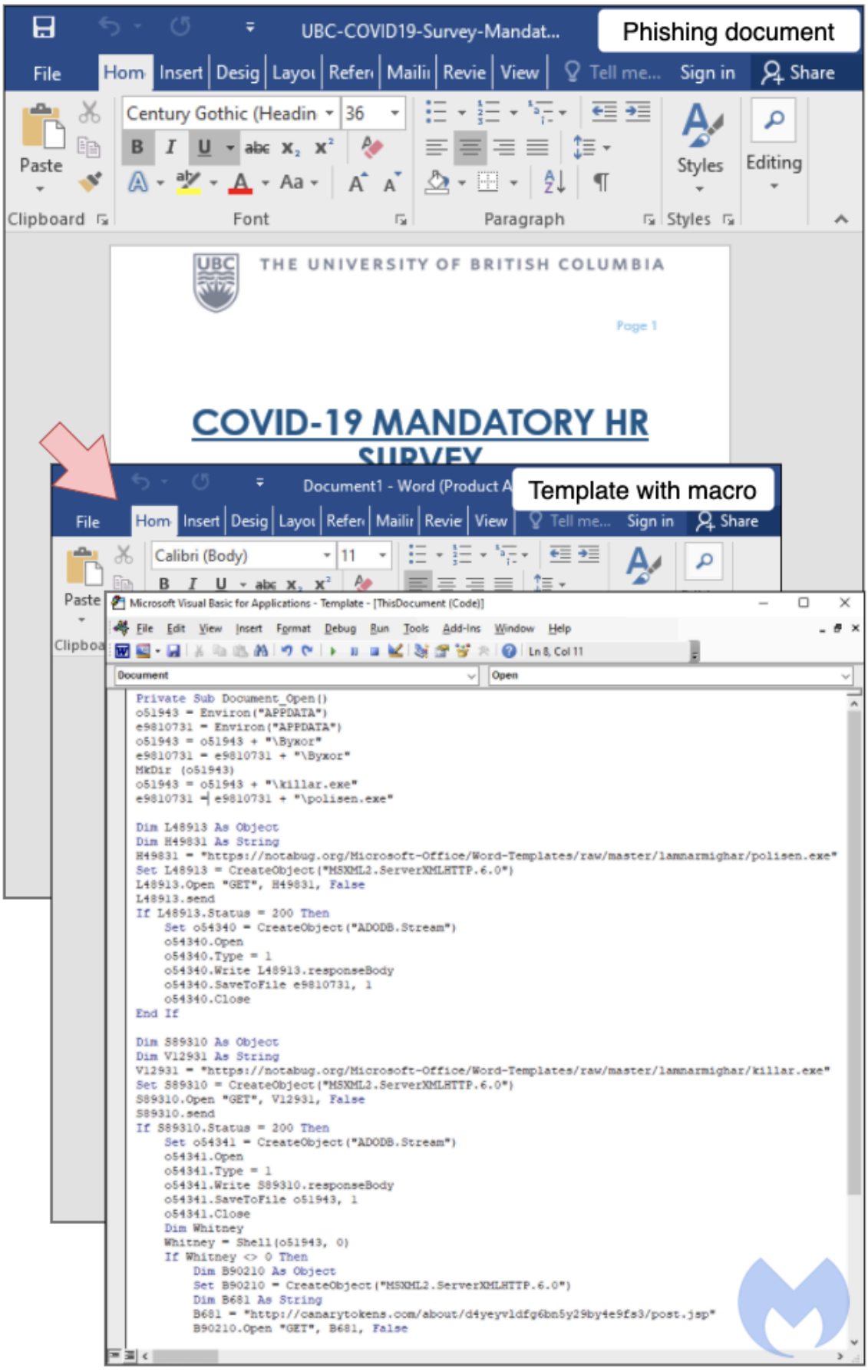
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

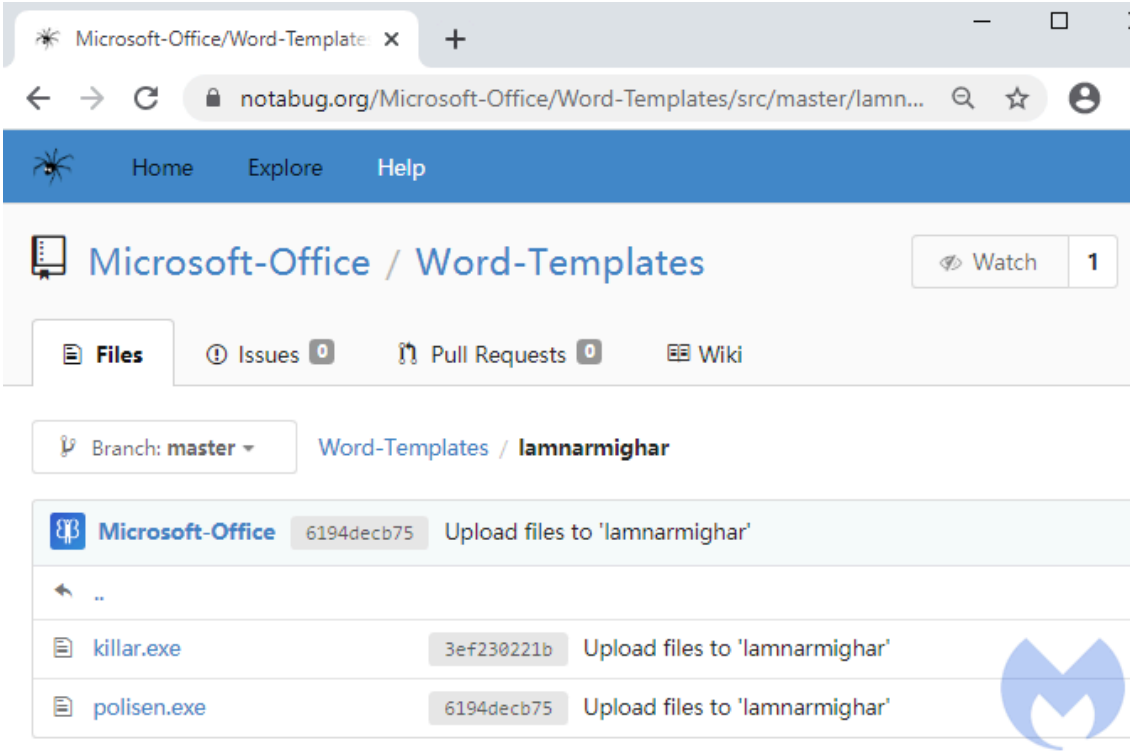
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



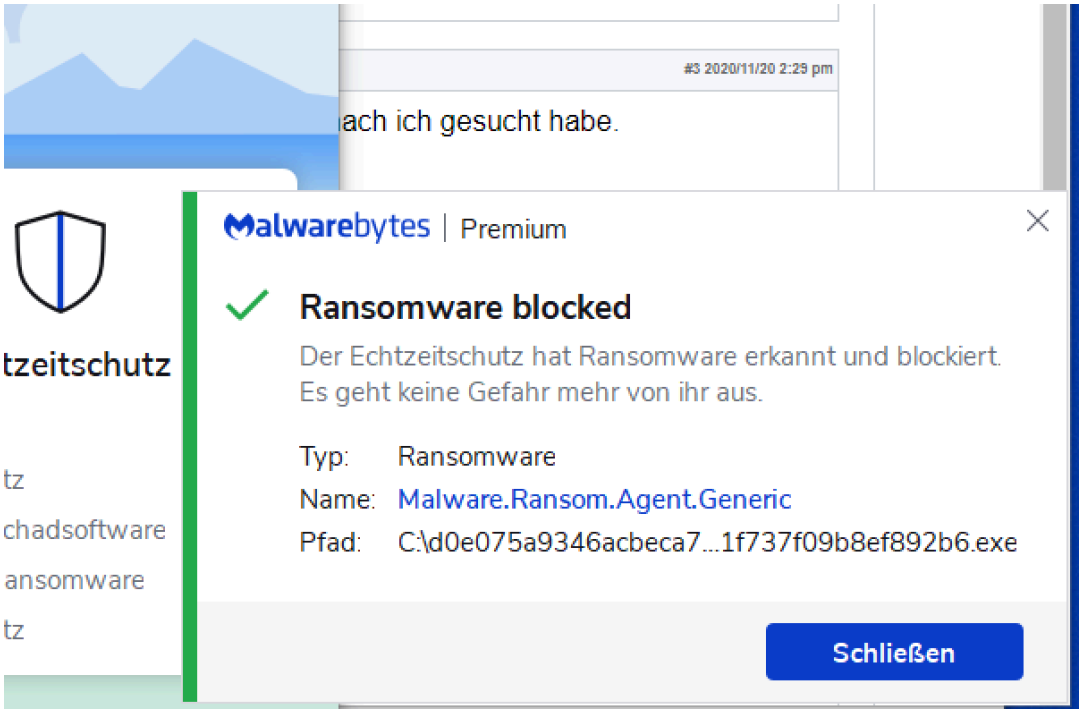
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

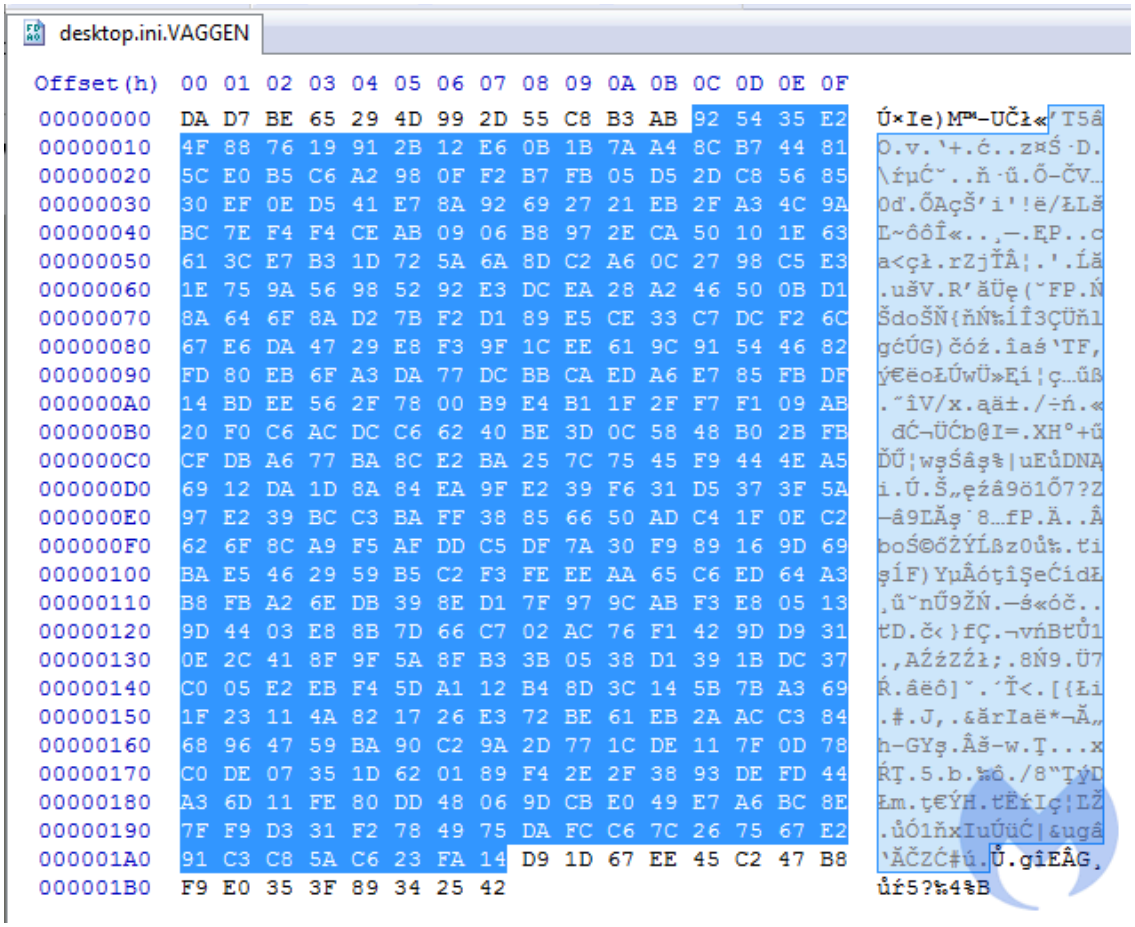
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

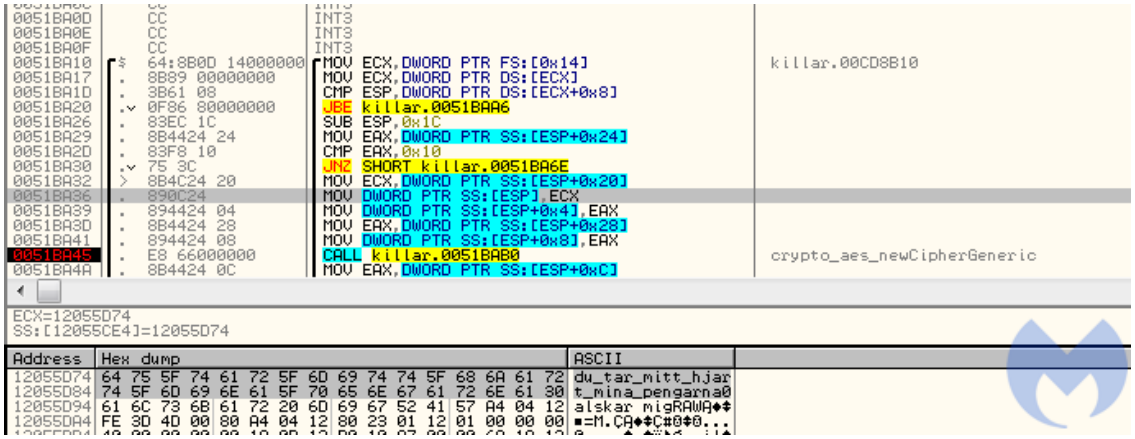


The content of the output file (with .VAGGEN extension) contains:

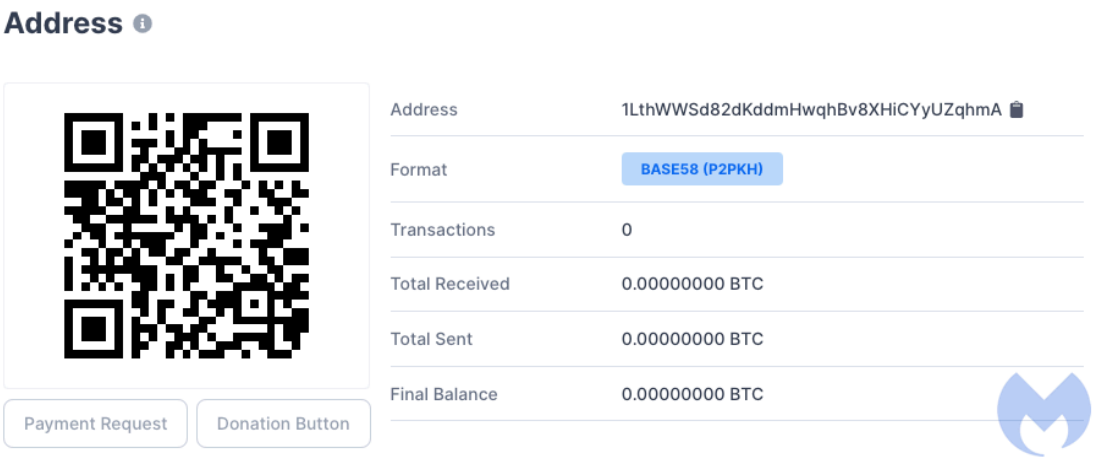
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

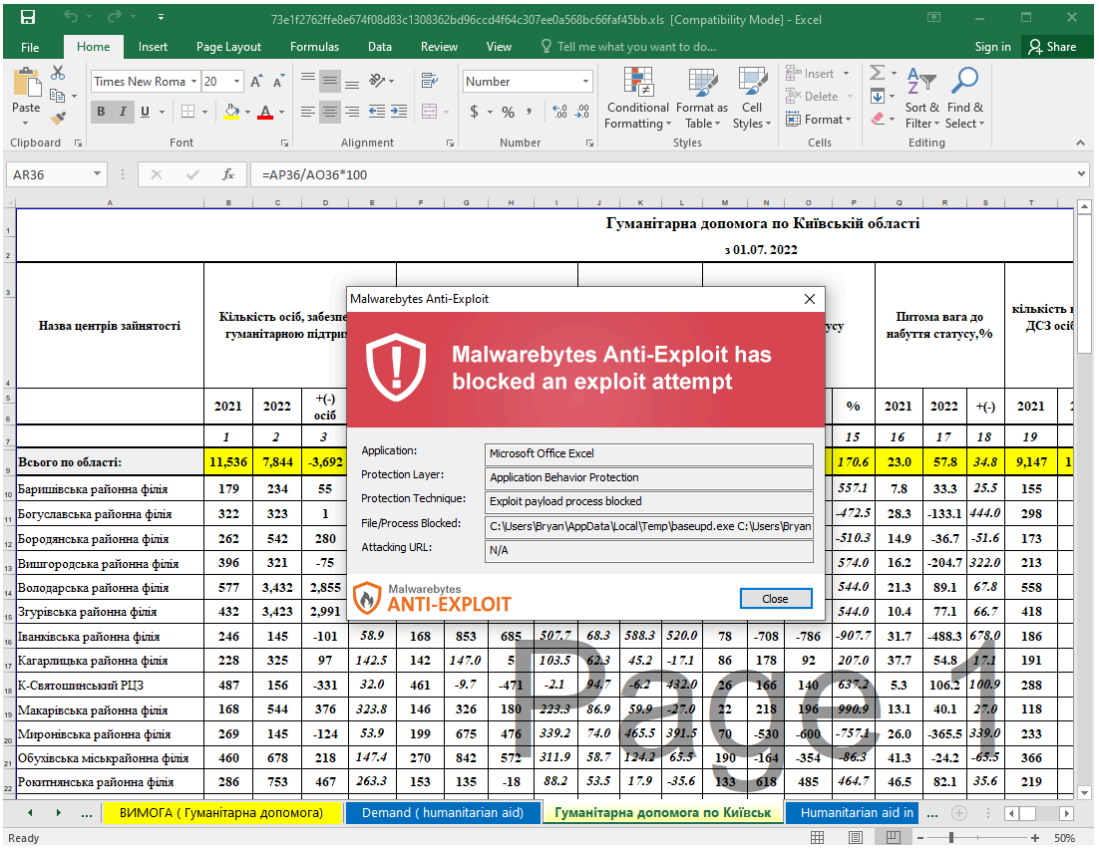
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

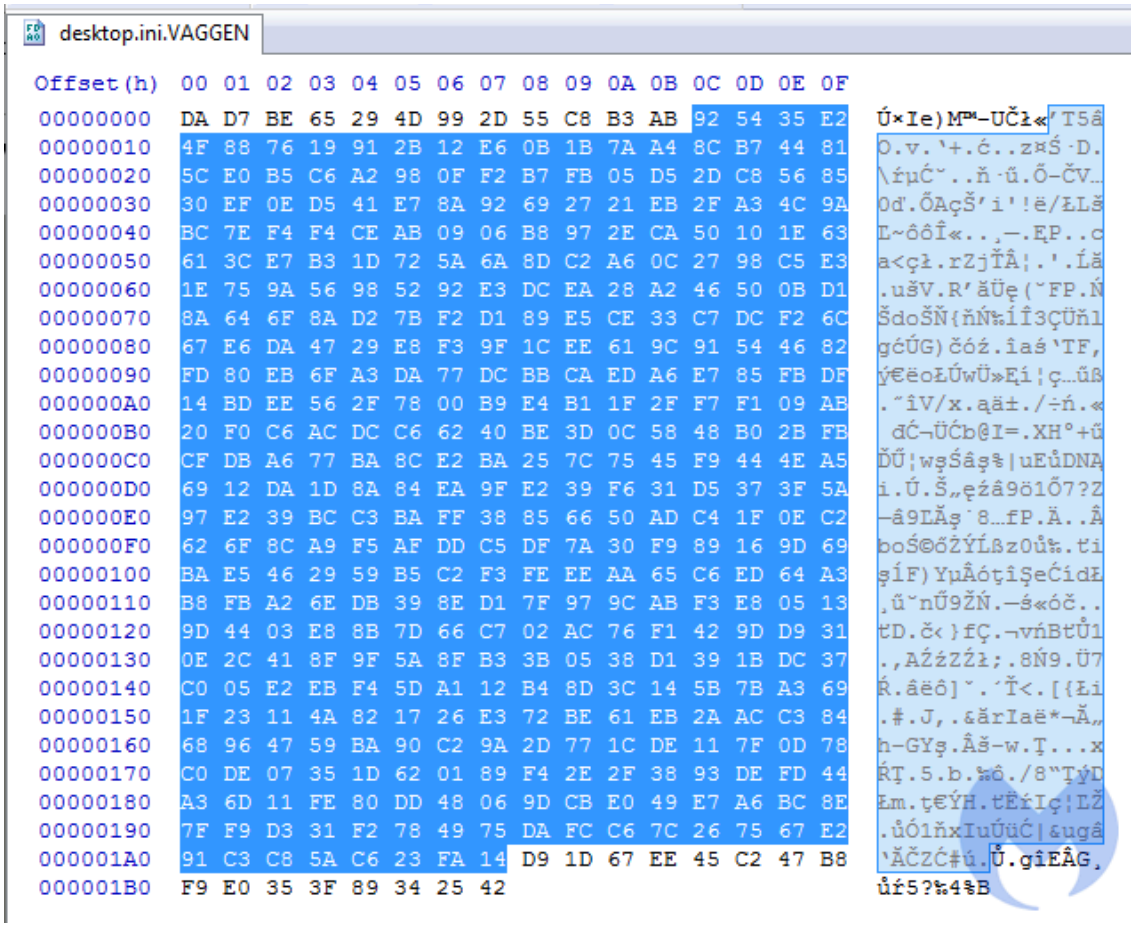
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

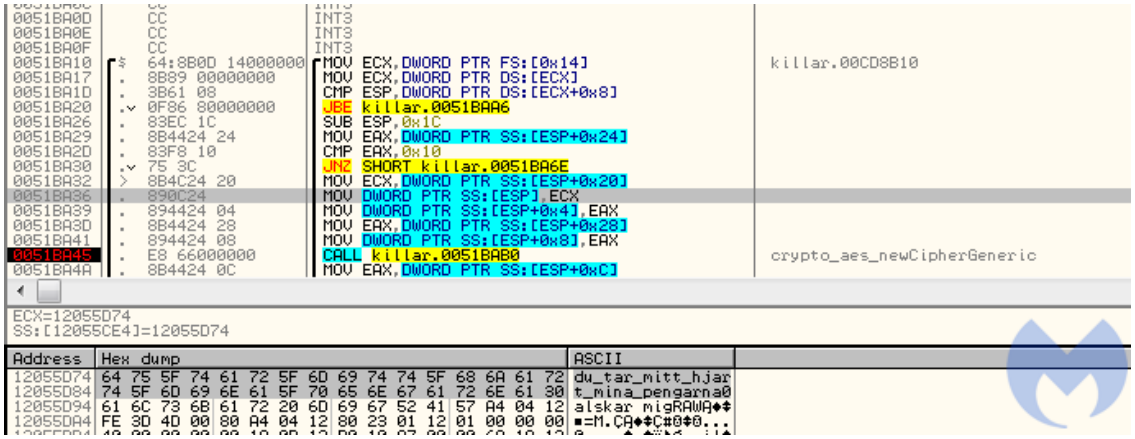


The content of the output file (with .VAGGEN extension) contains:

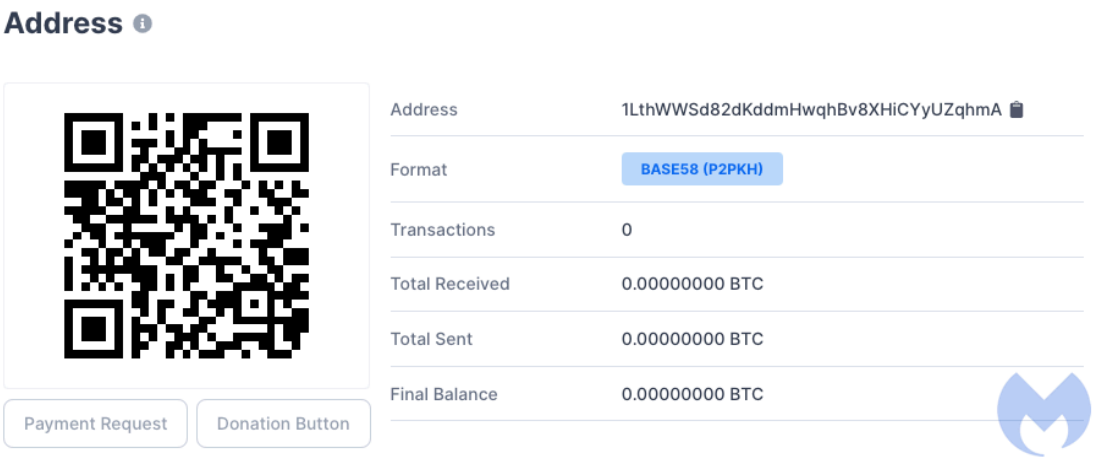
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

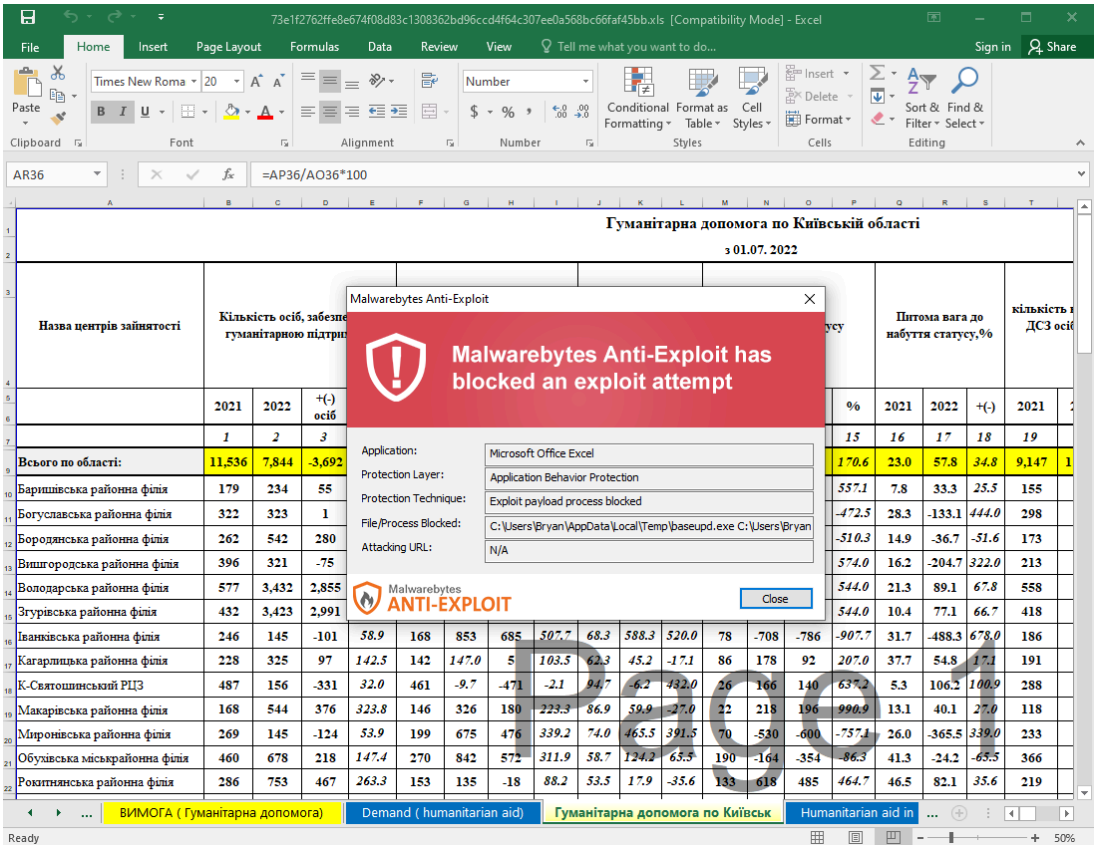
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

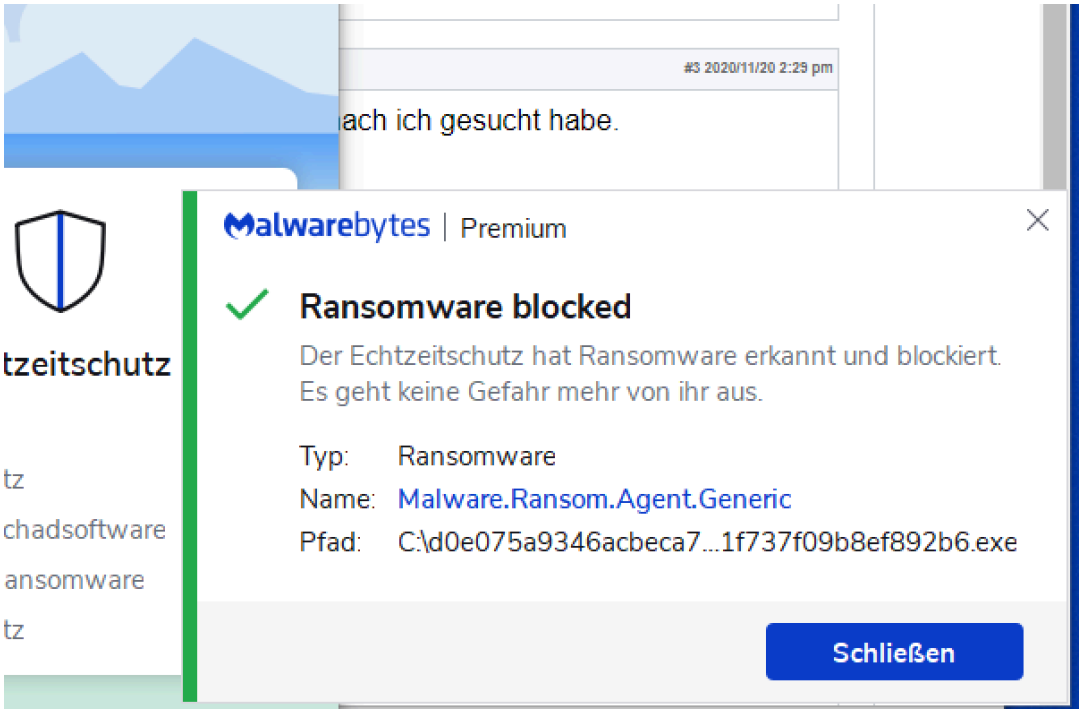
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

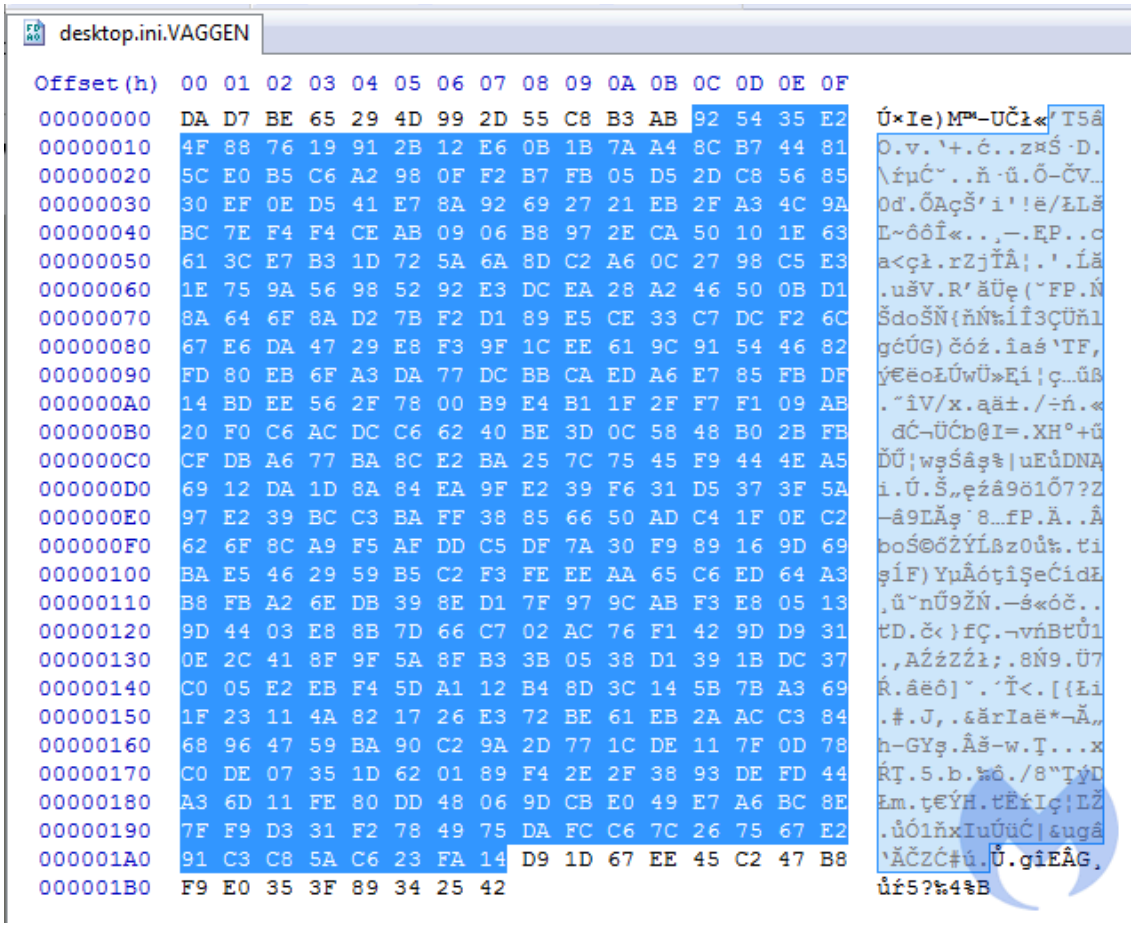
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



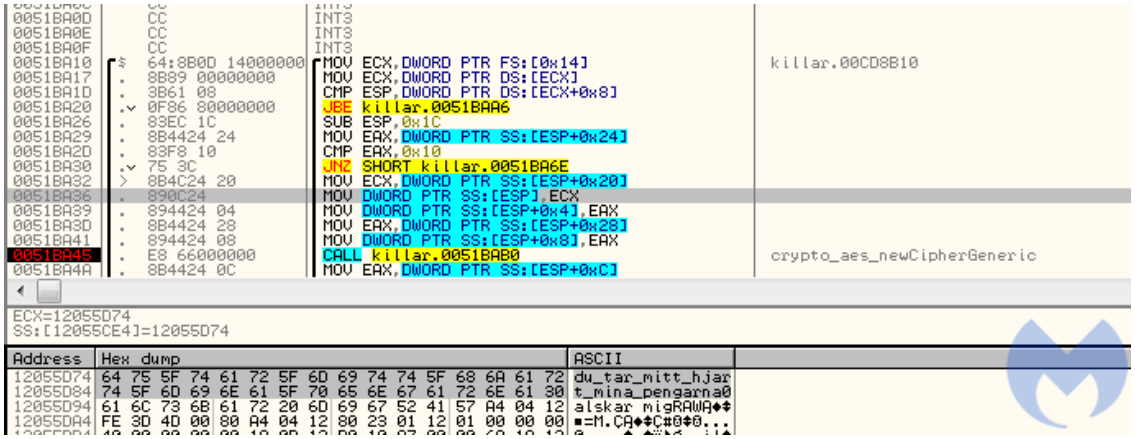
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

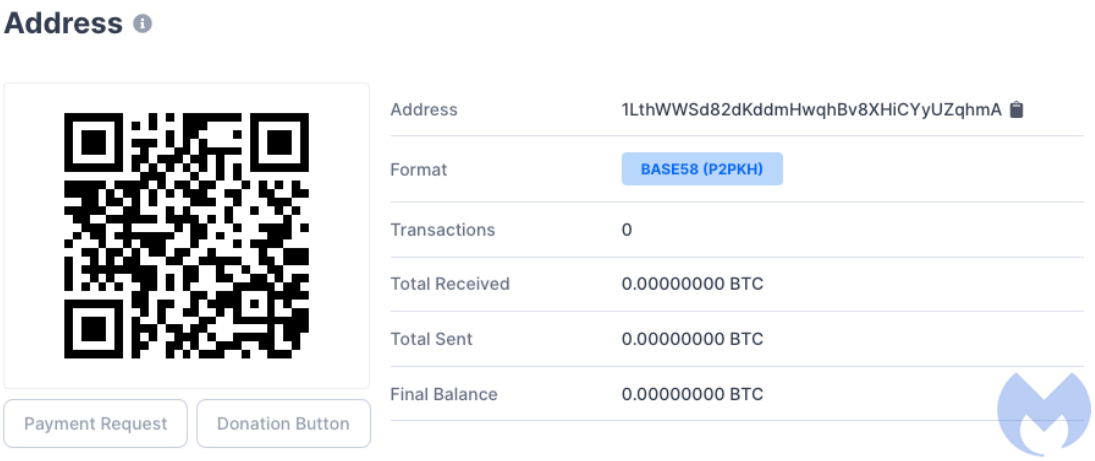
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

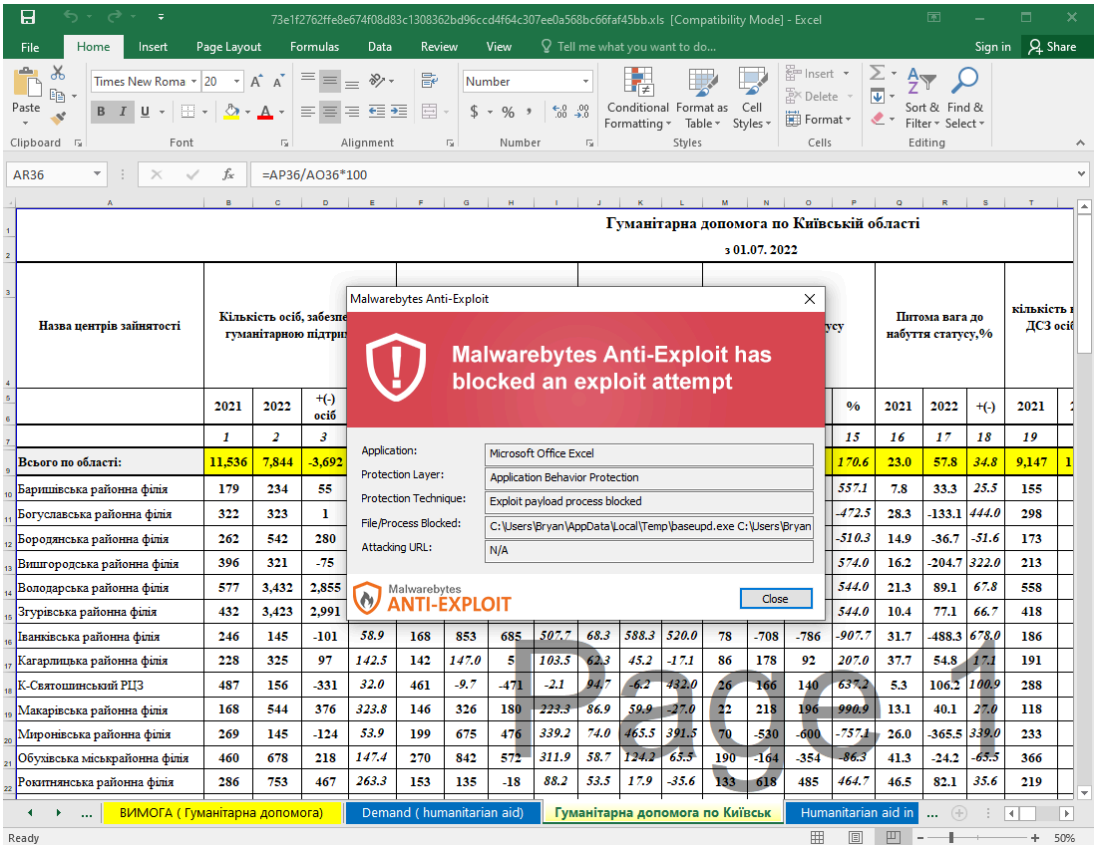
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

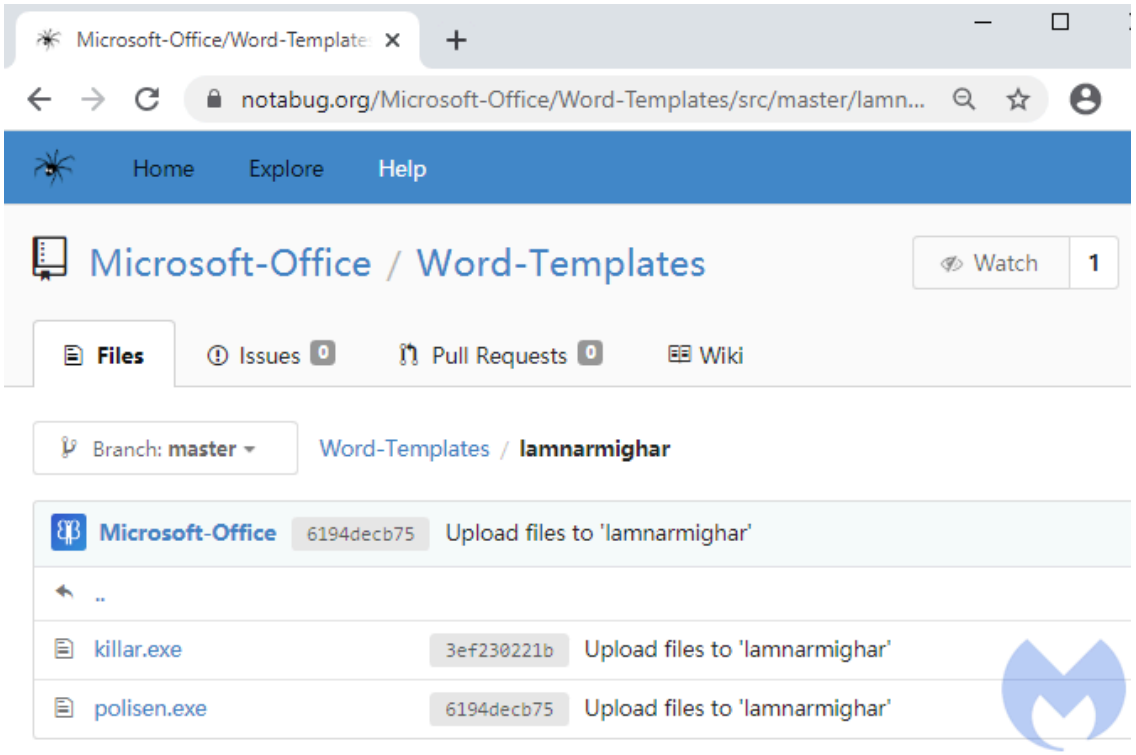
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



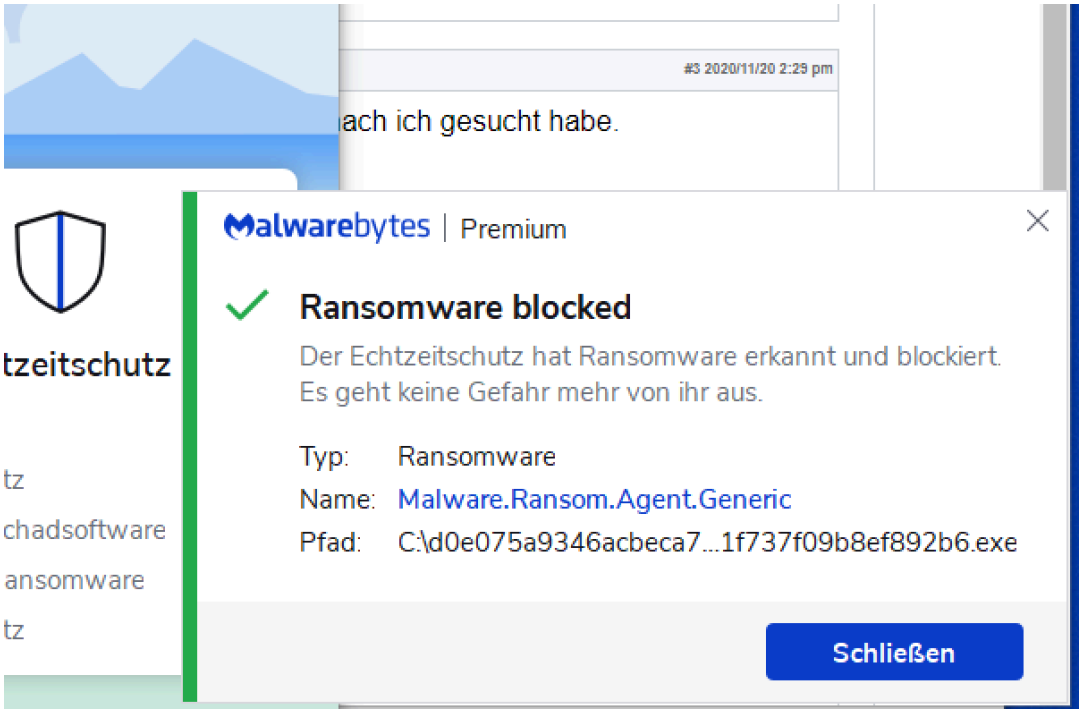
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

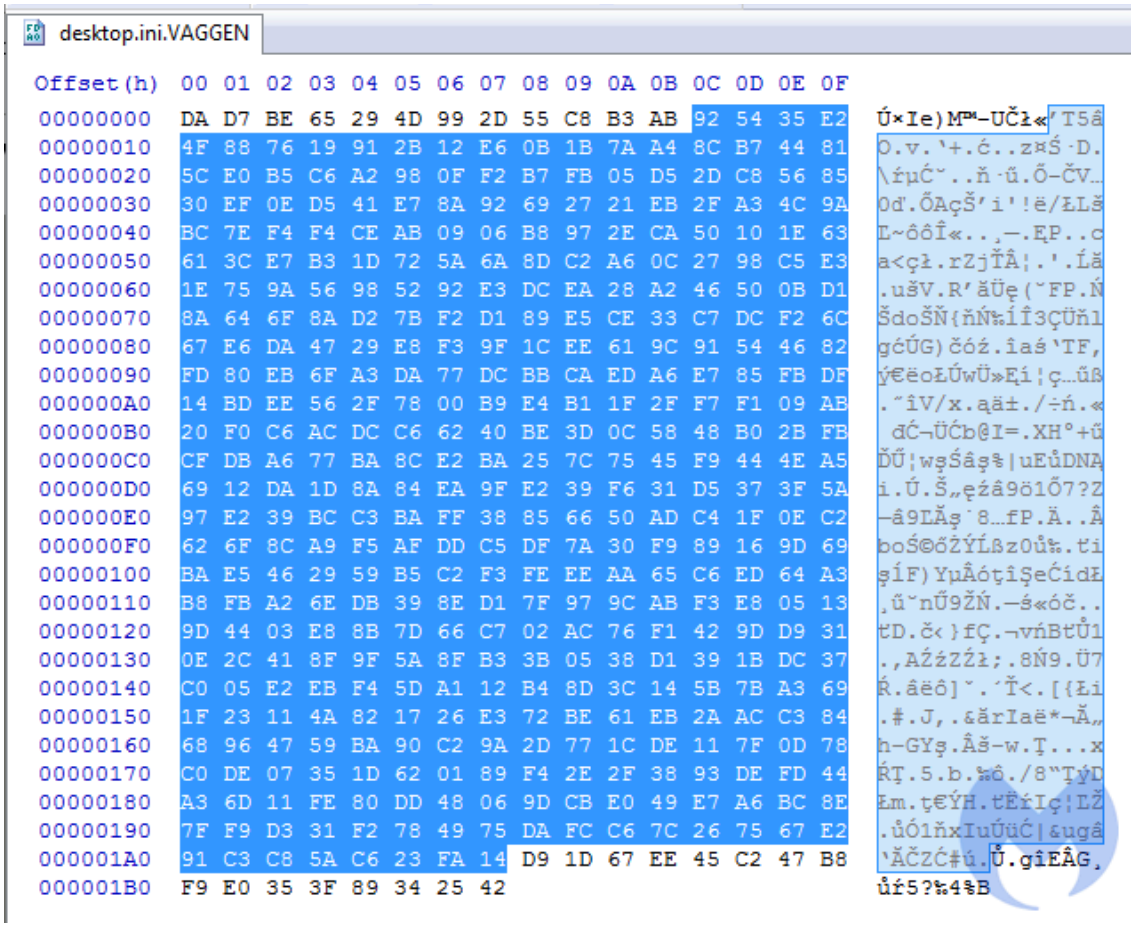
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

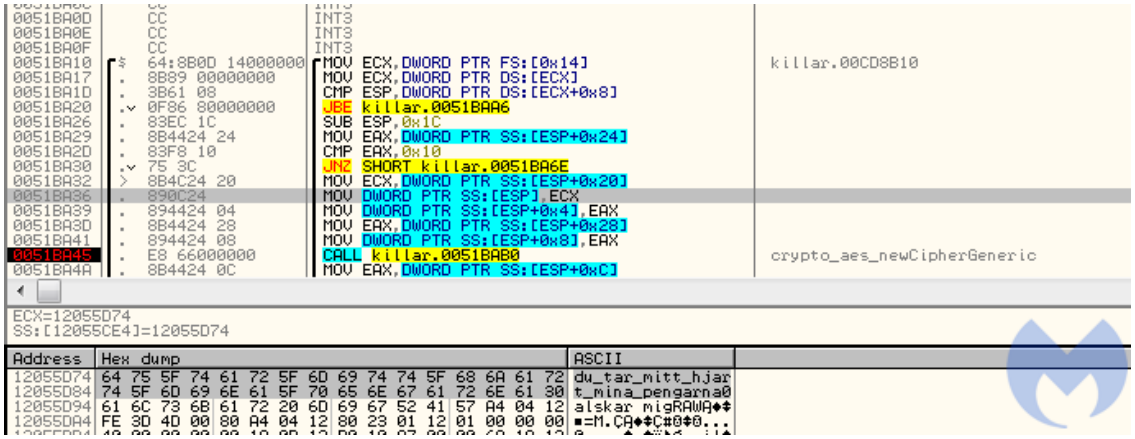


The content of the output file (with .VAGGEN extension) contains:

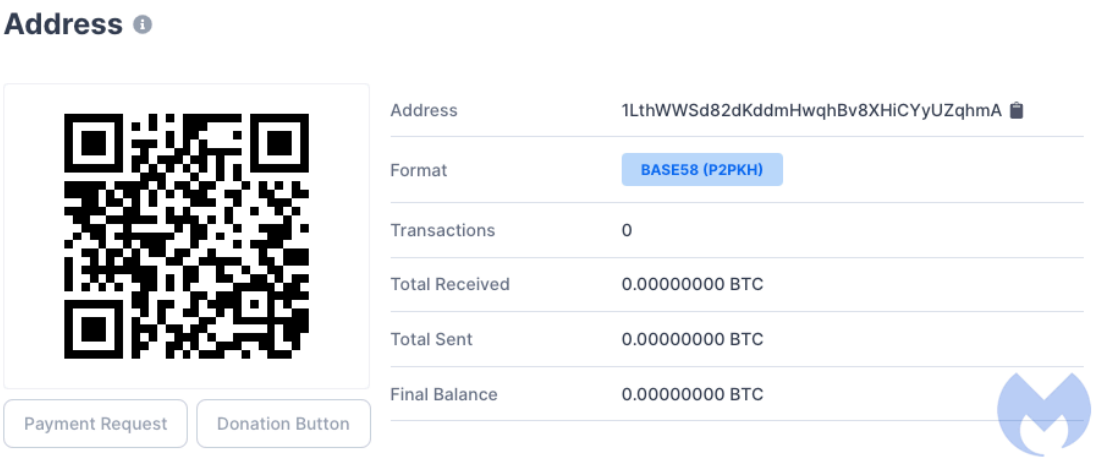
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

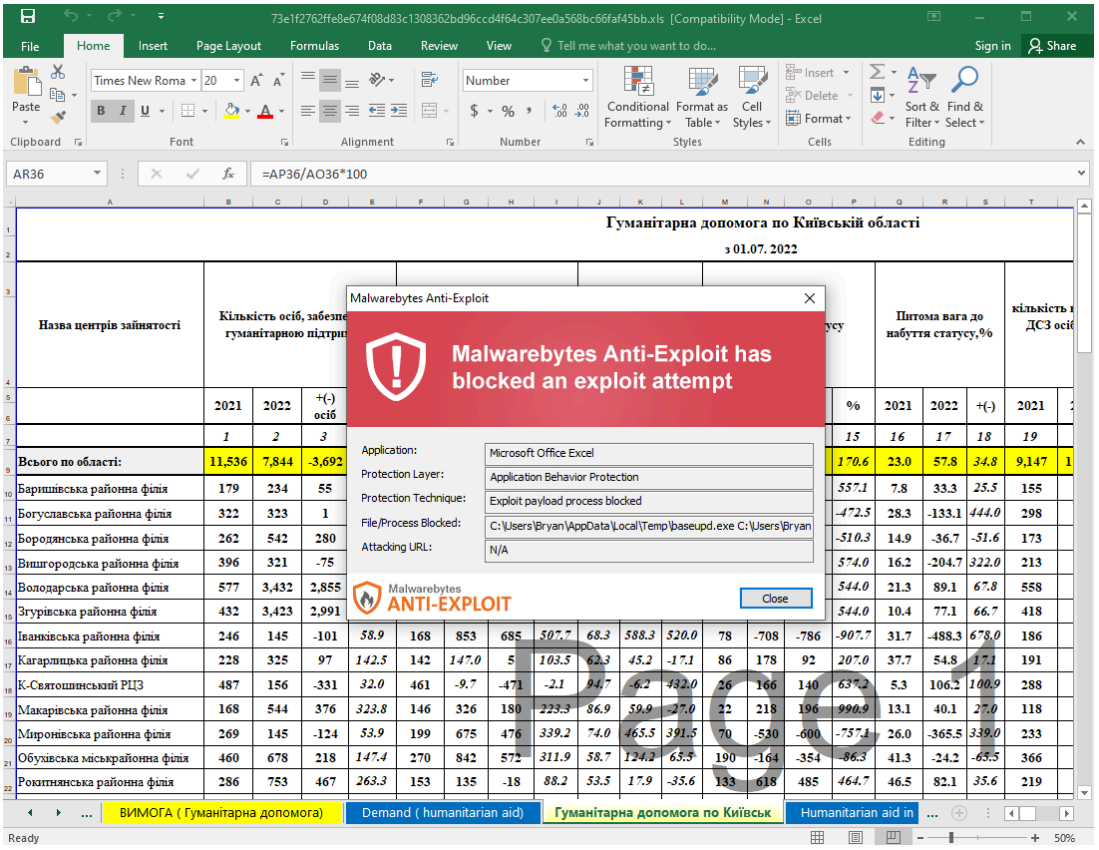
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

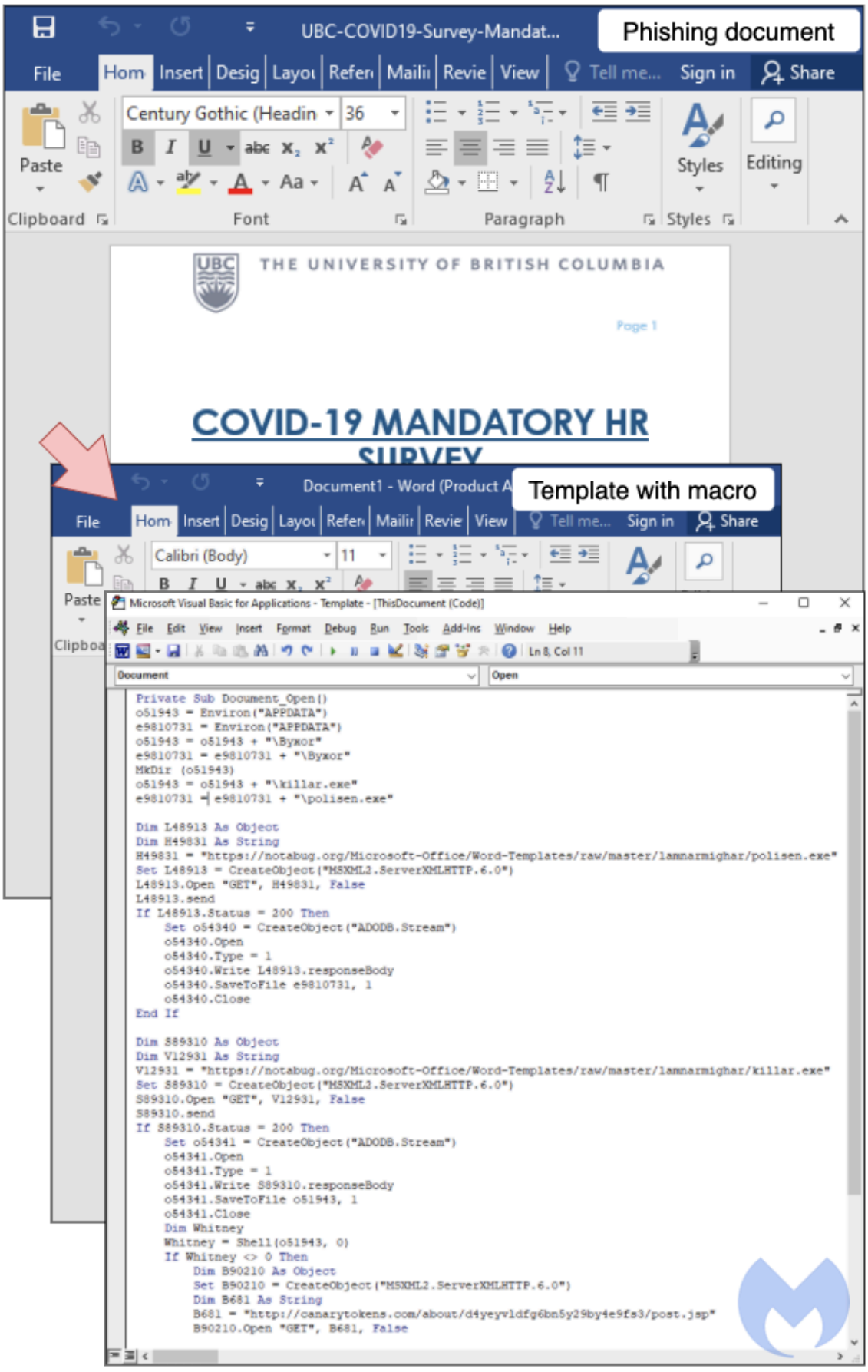
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

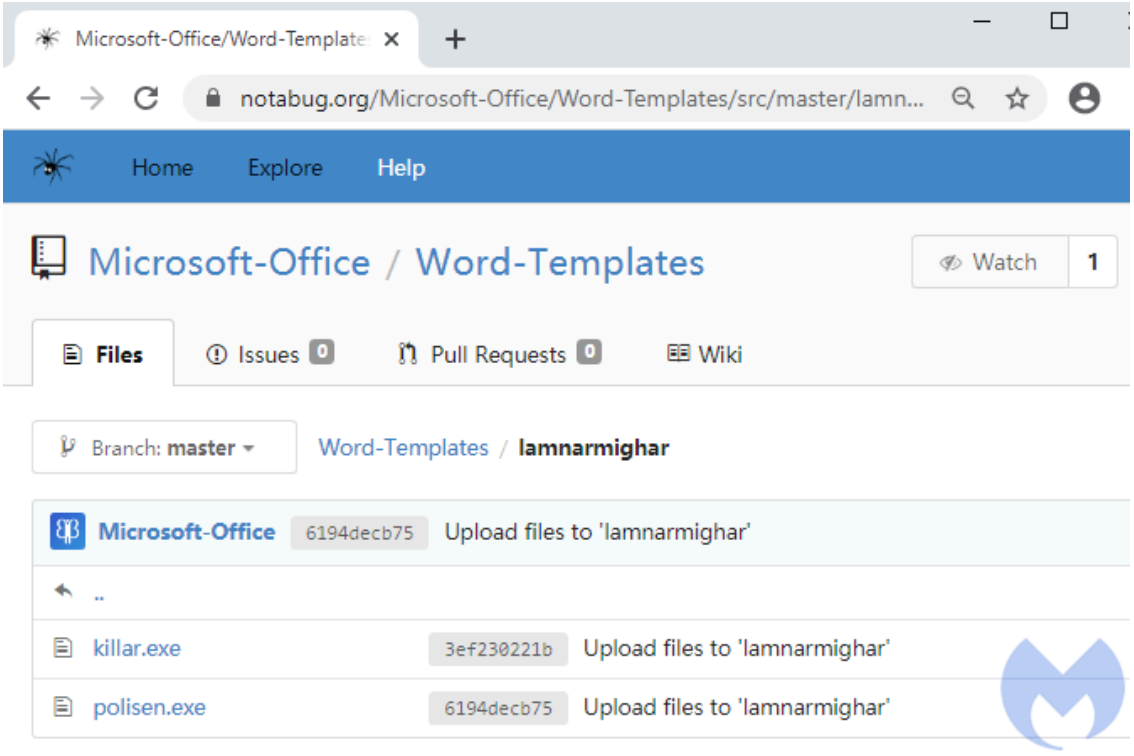
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



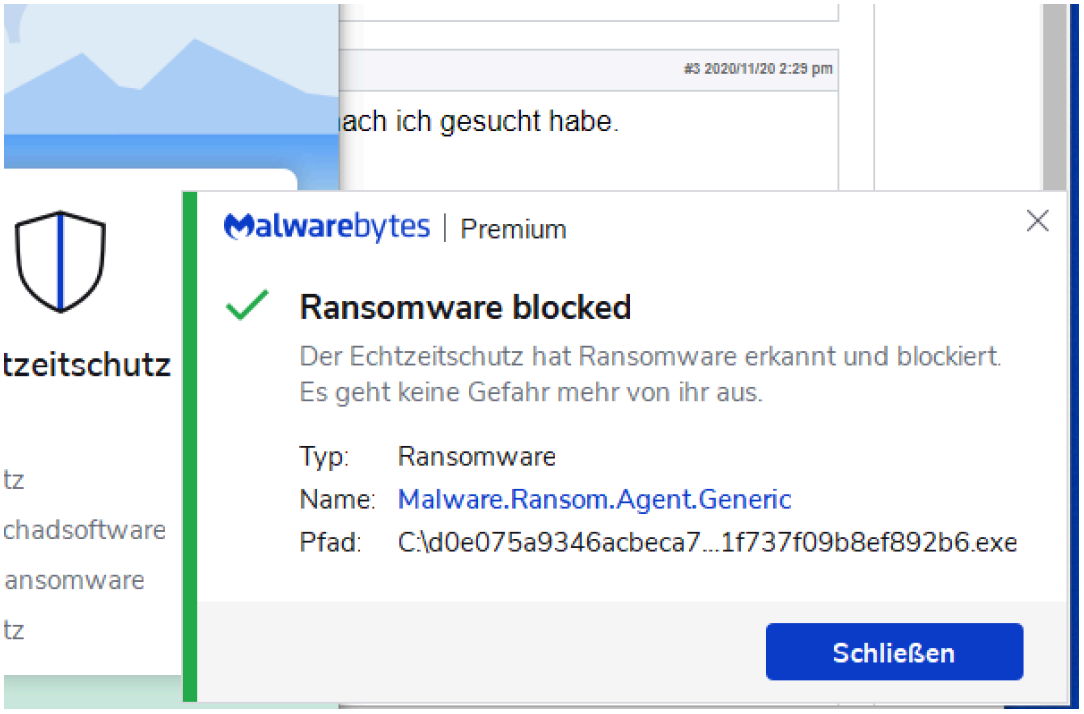
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

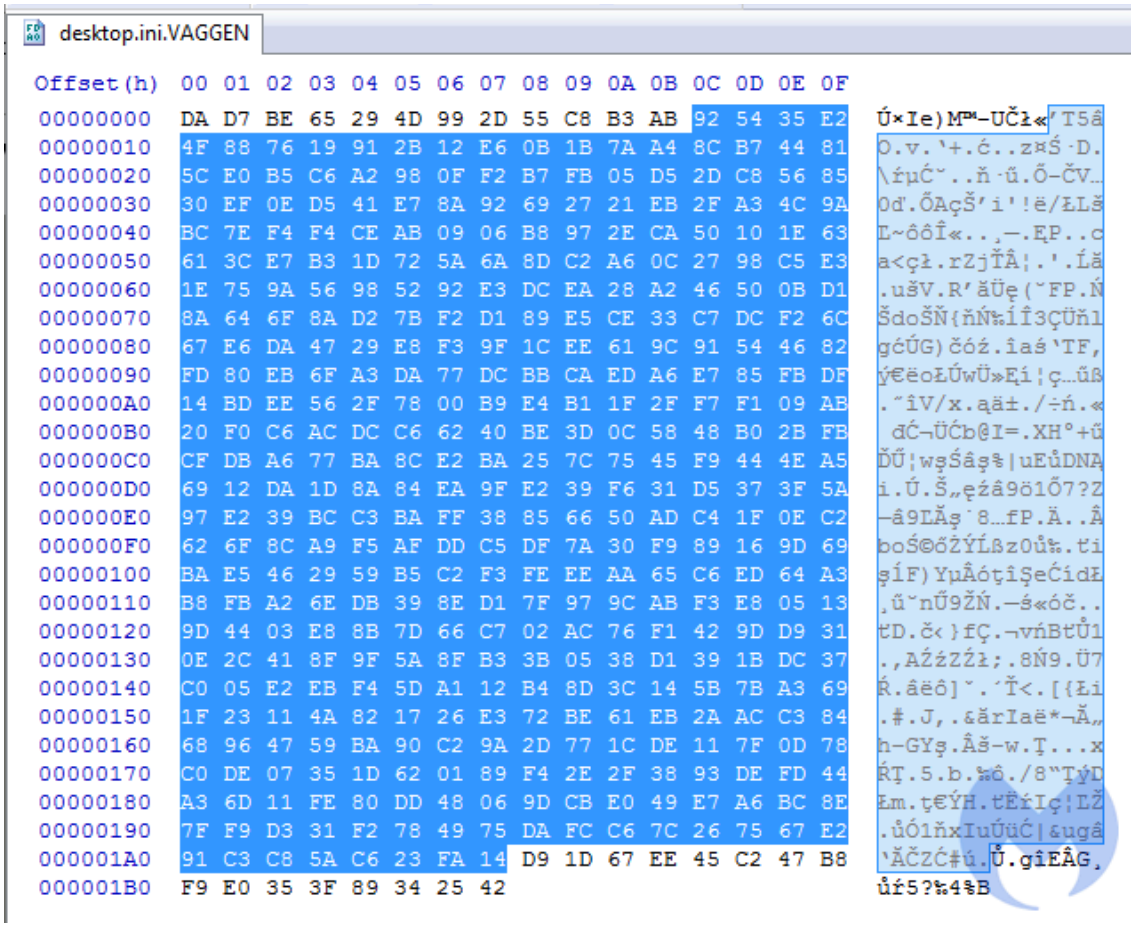
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

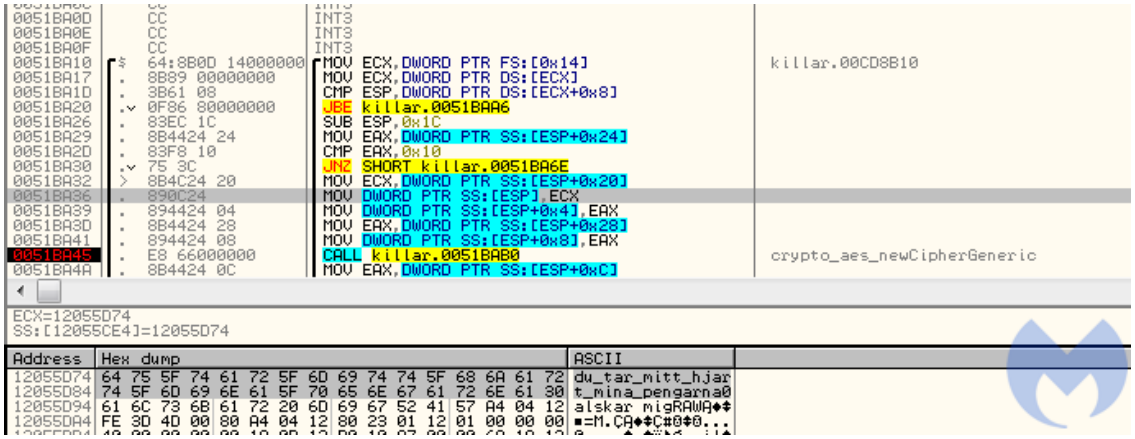


The content of the output file (with .VAGGEN extension) contains:

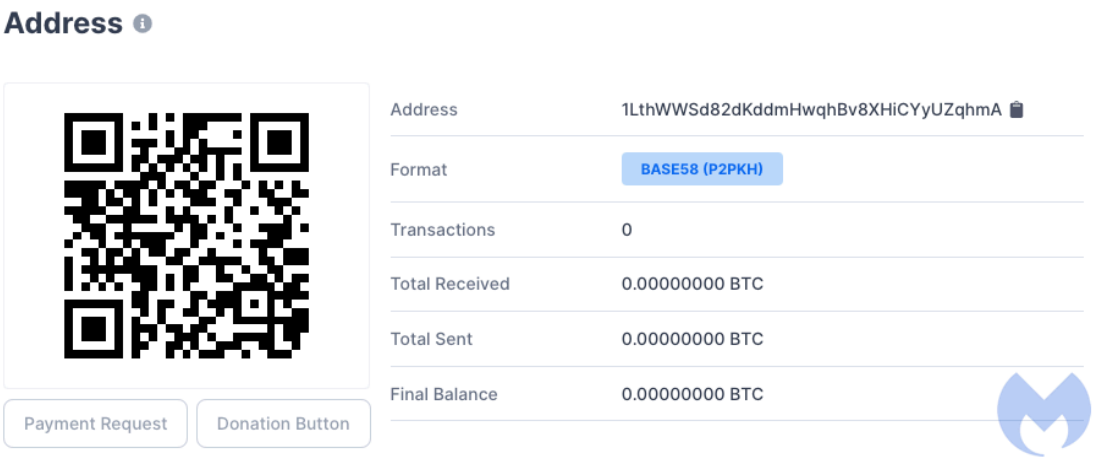
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

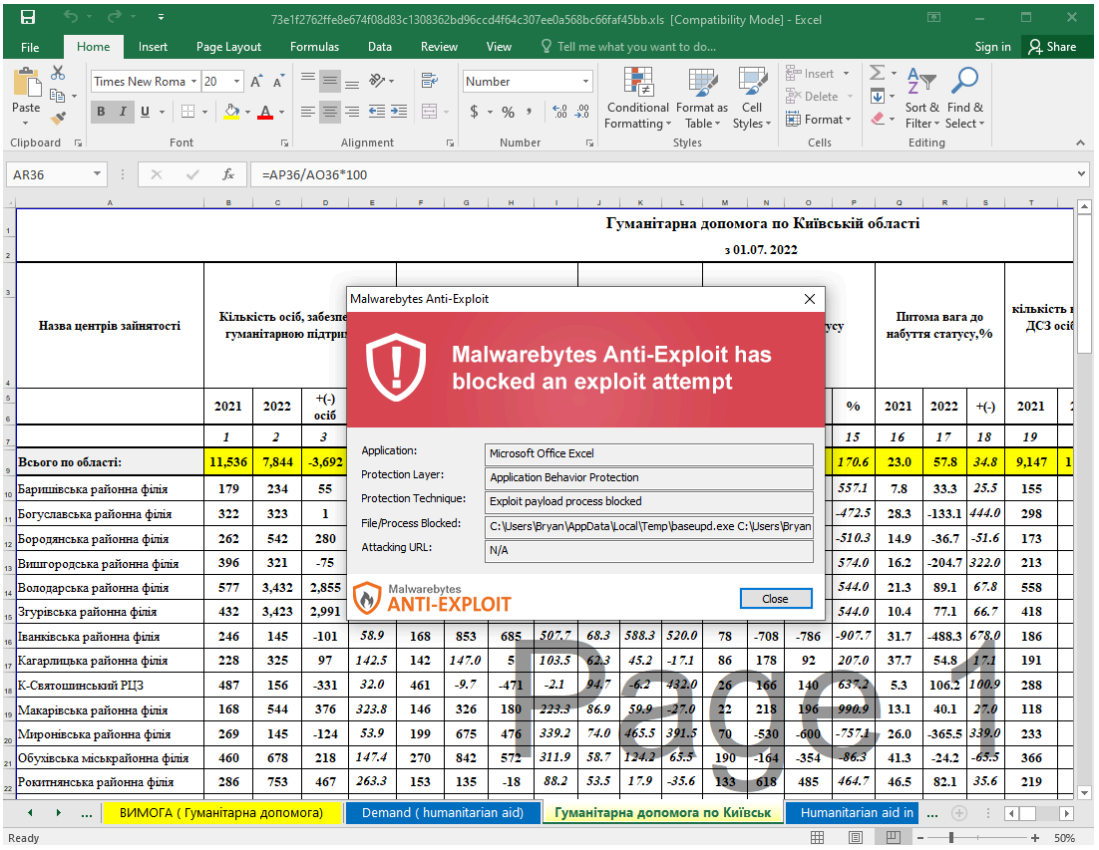
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

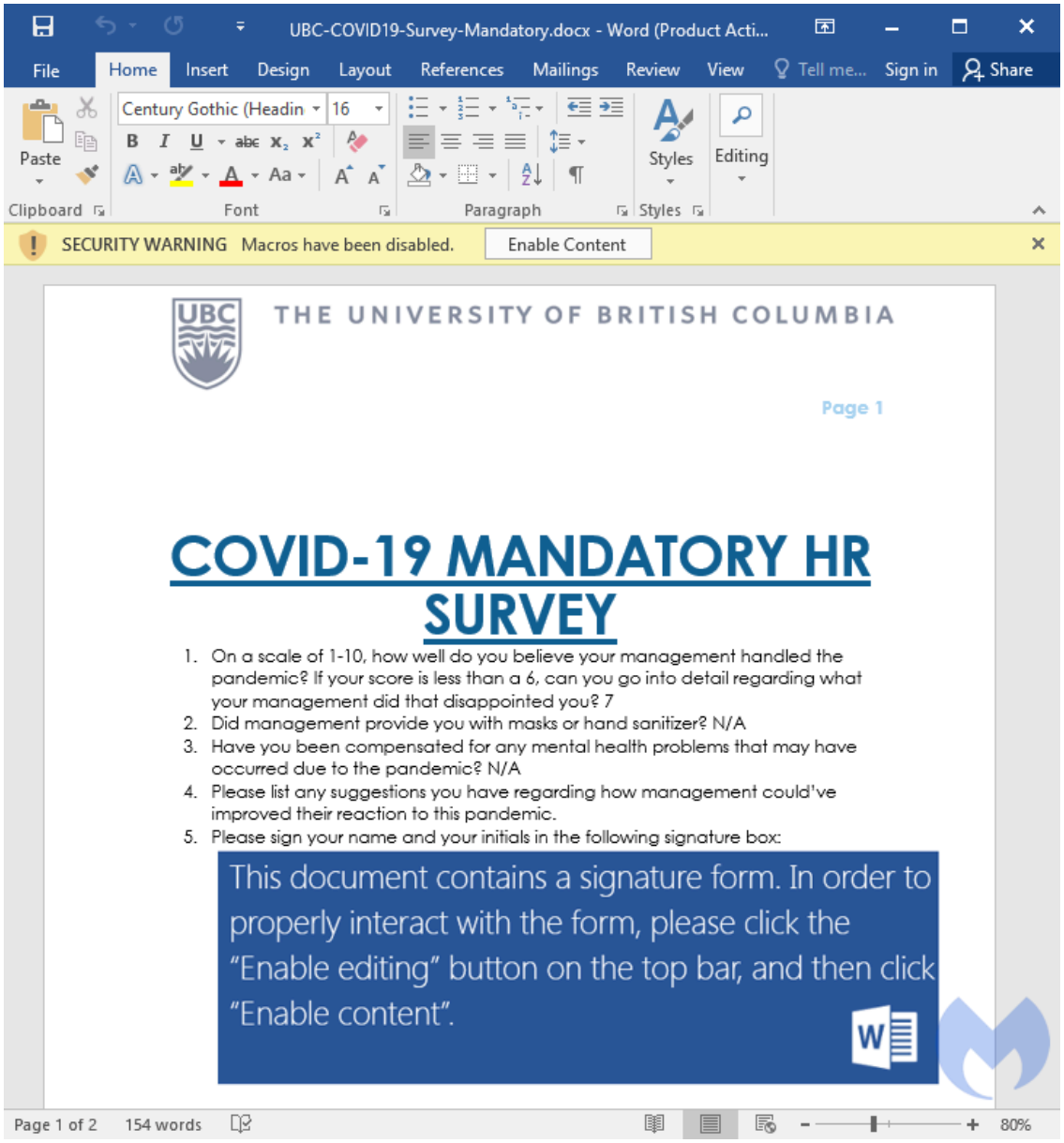
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

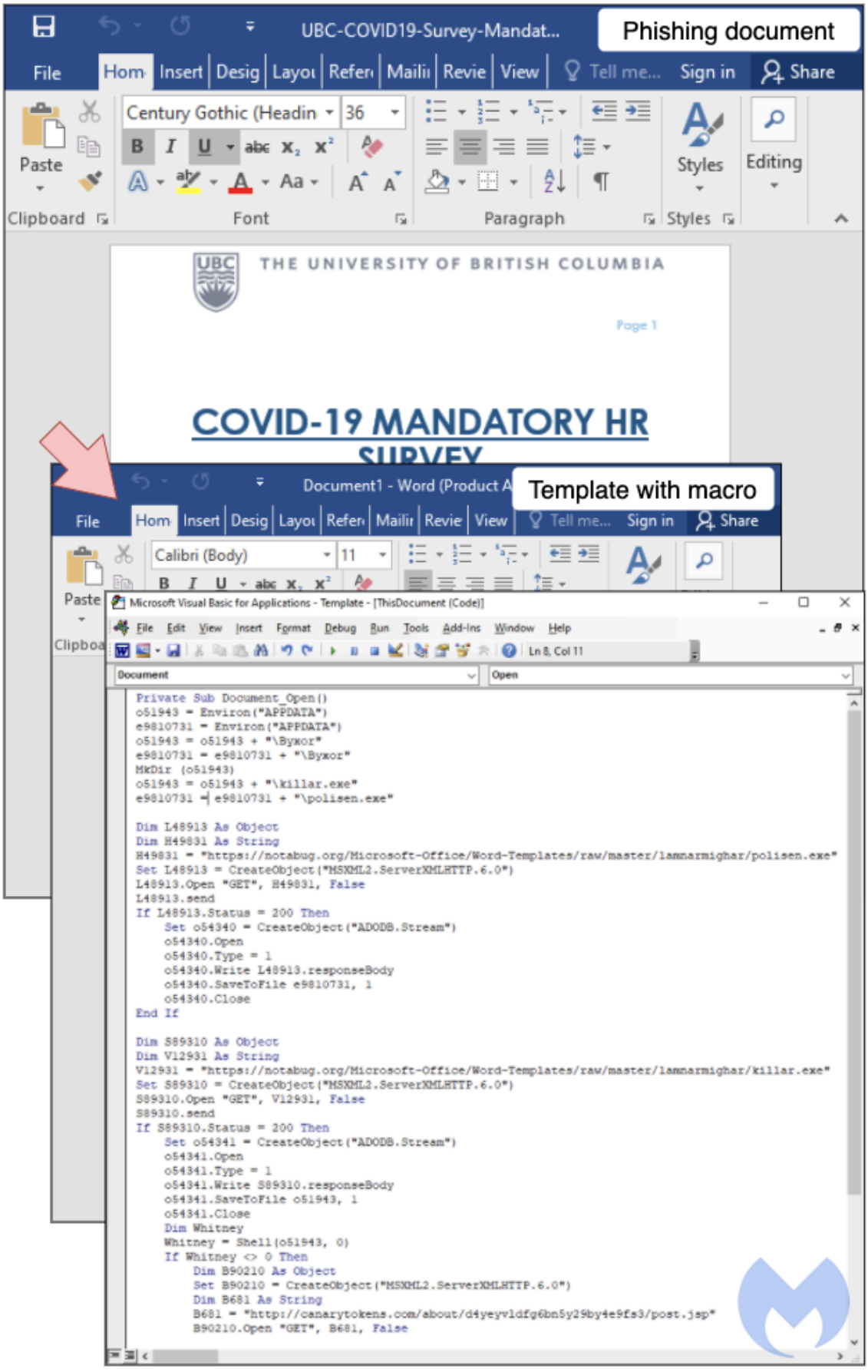
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

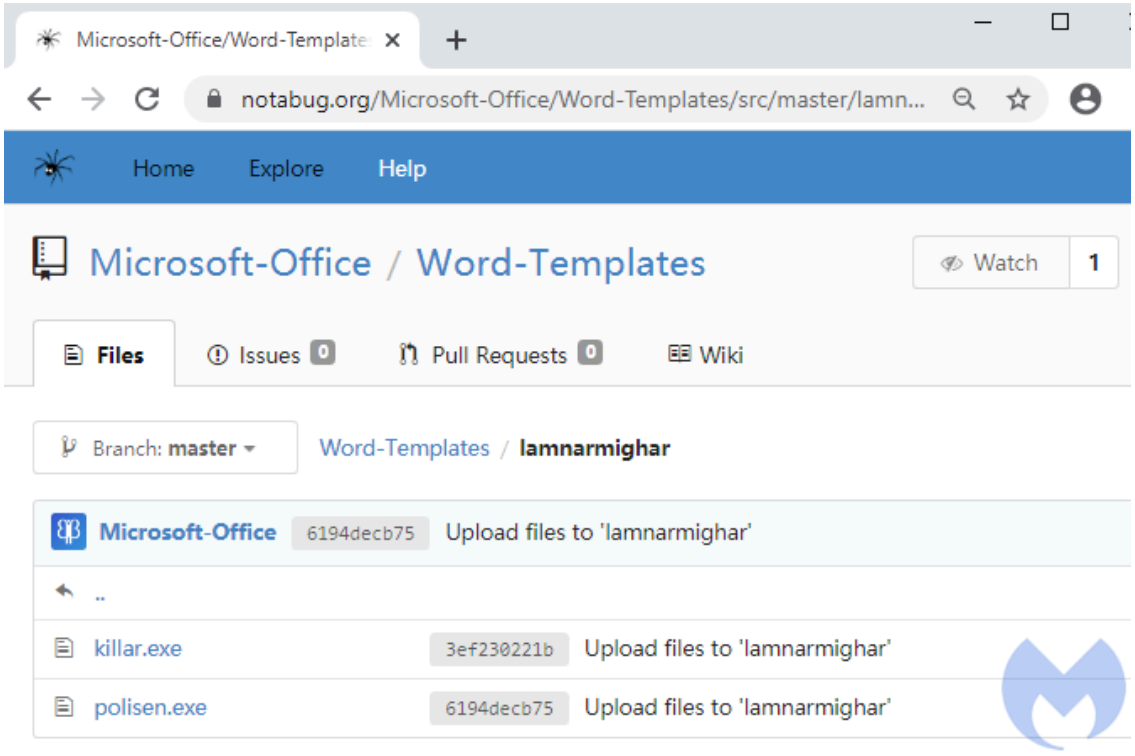
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



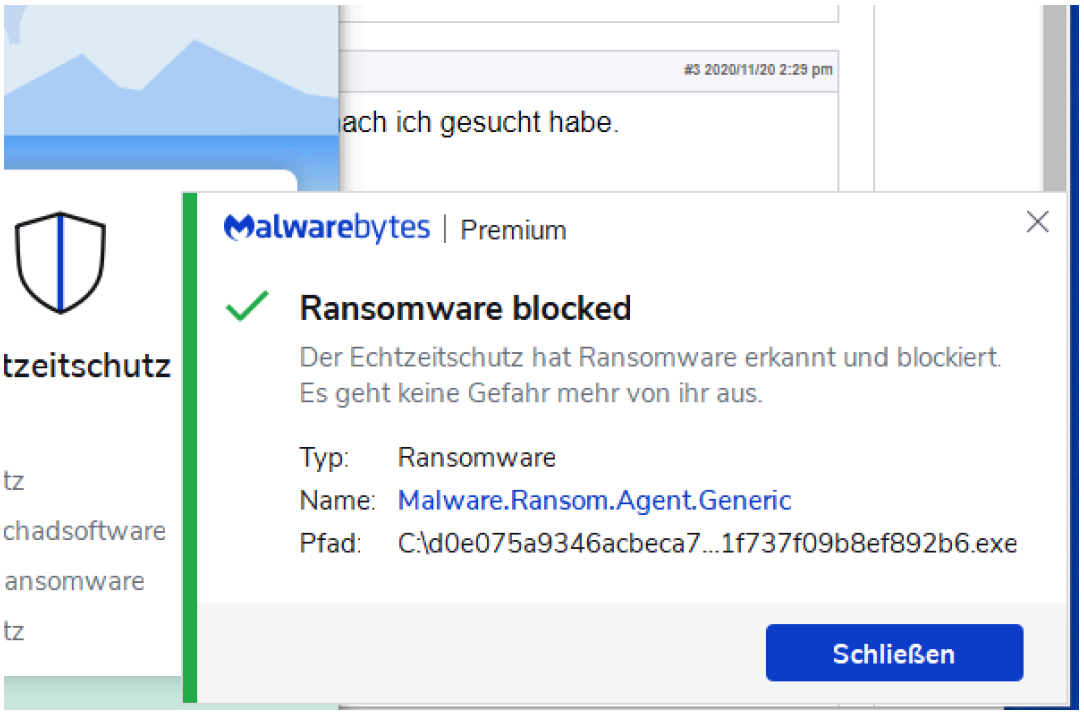
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

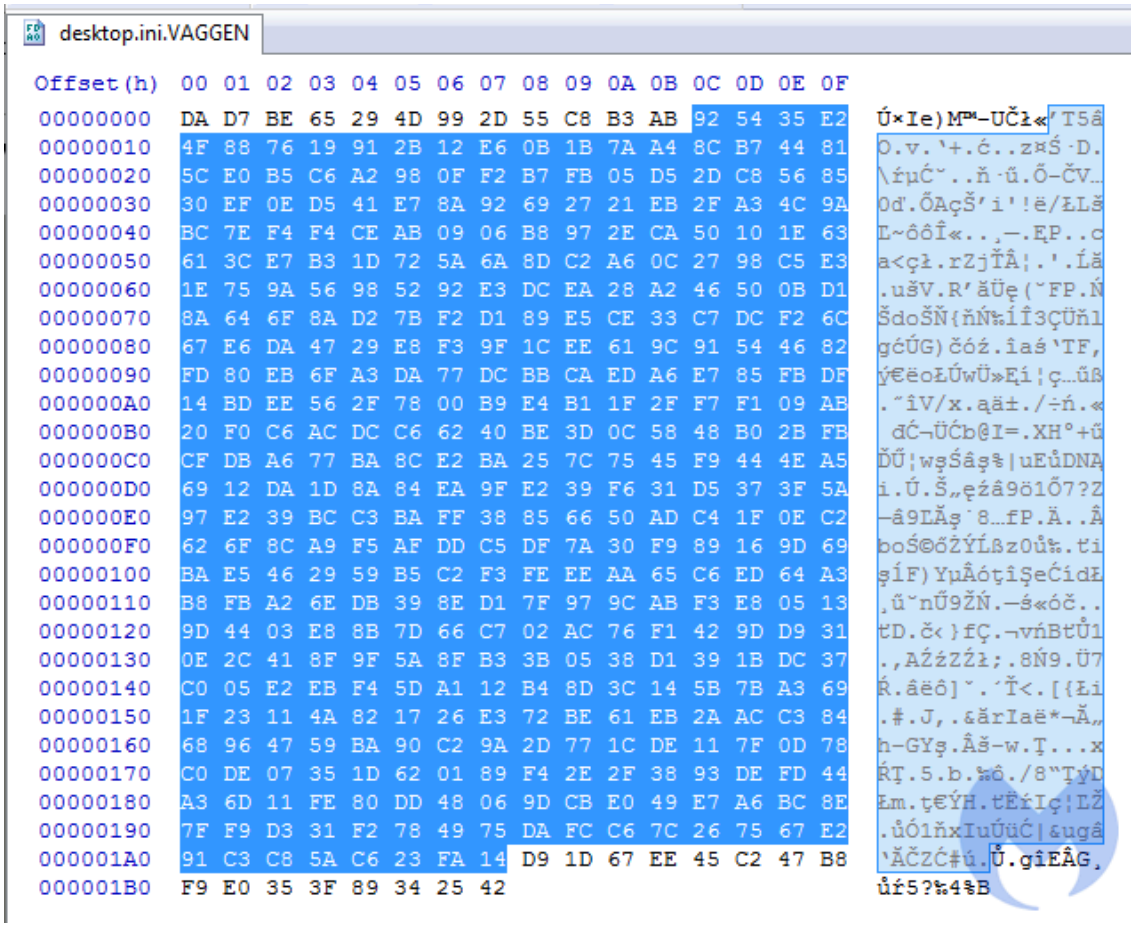
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

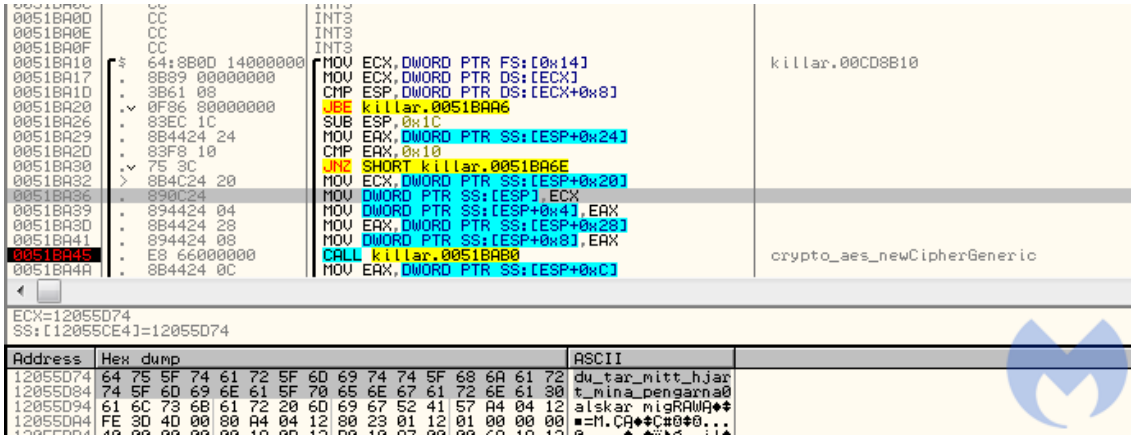


The content of the output file (with .VAGGEN extension) contains:

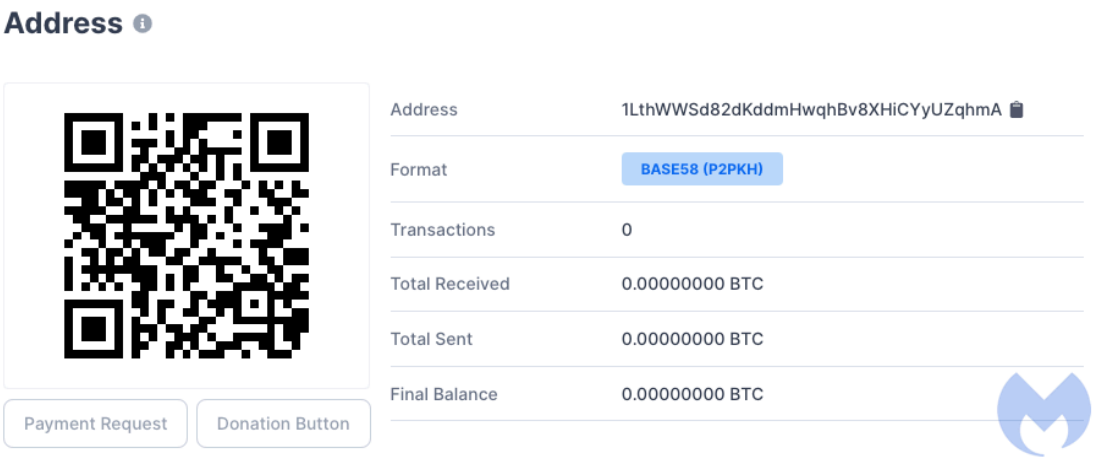
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

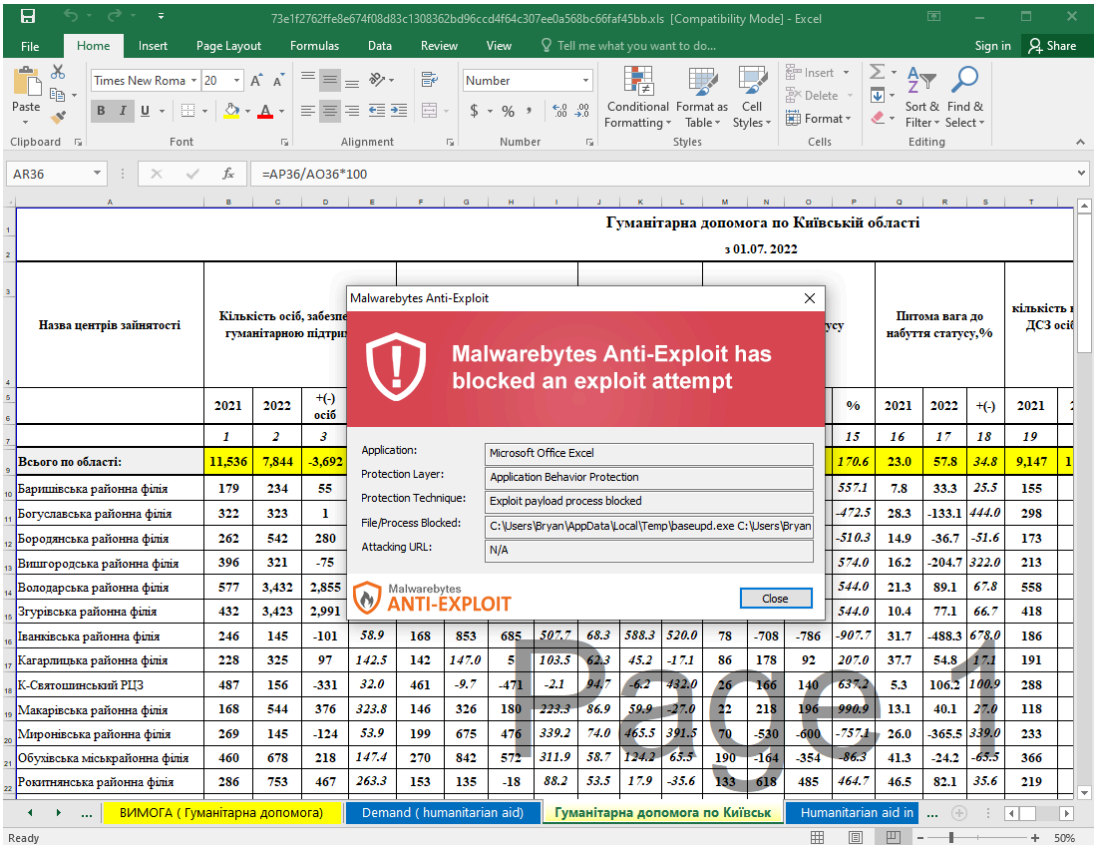
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

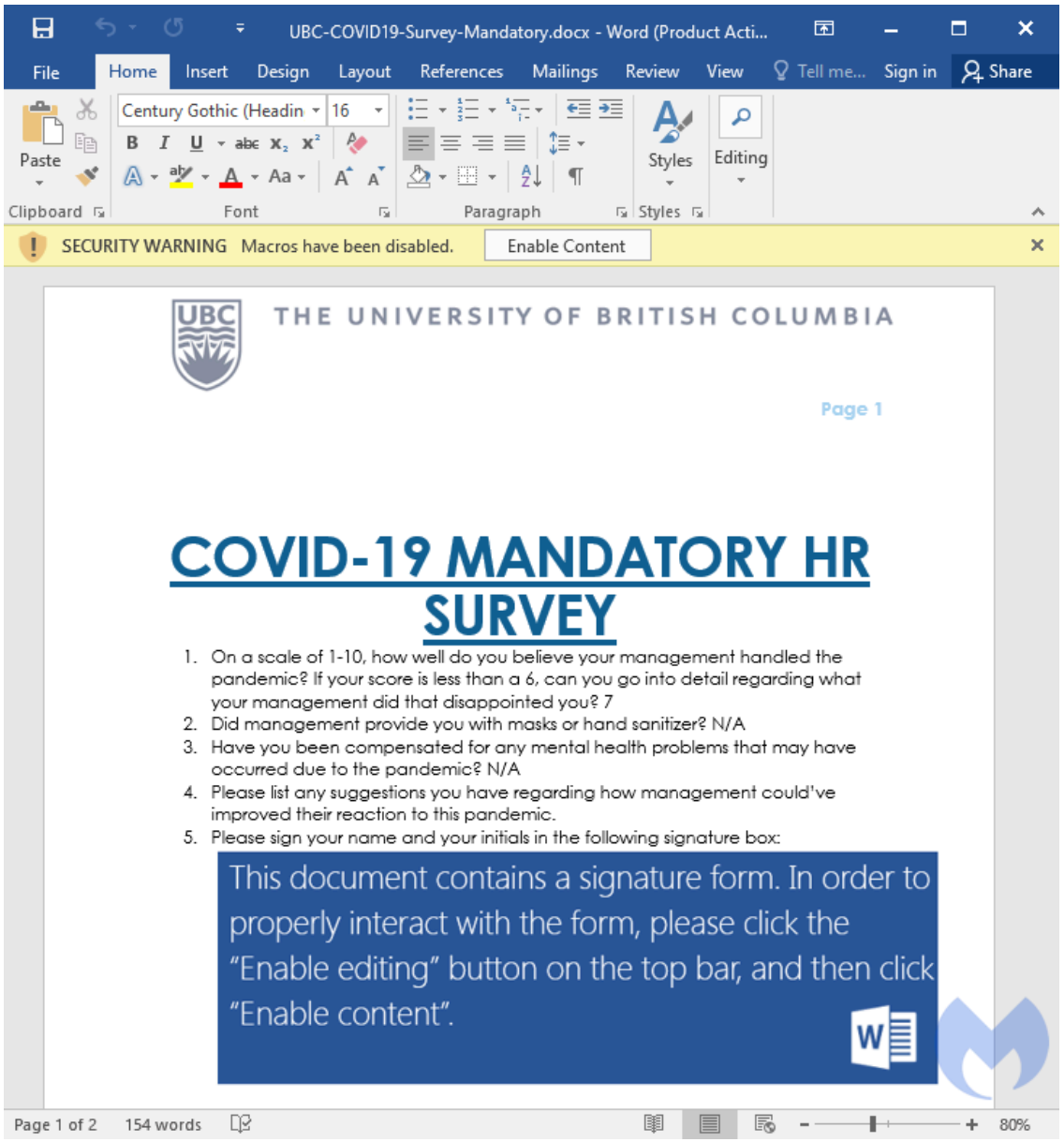
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

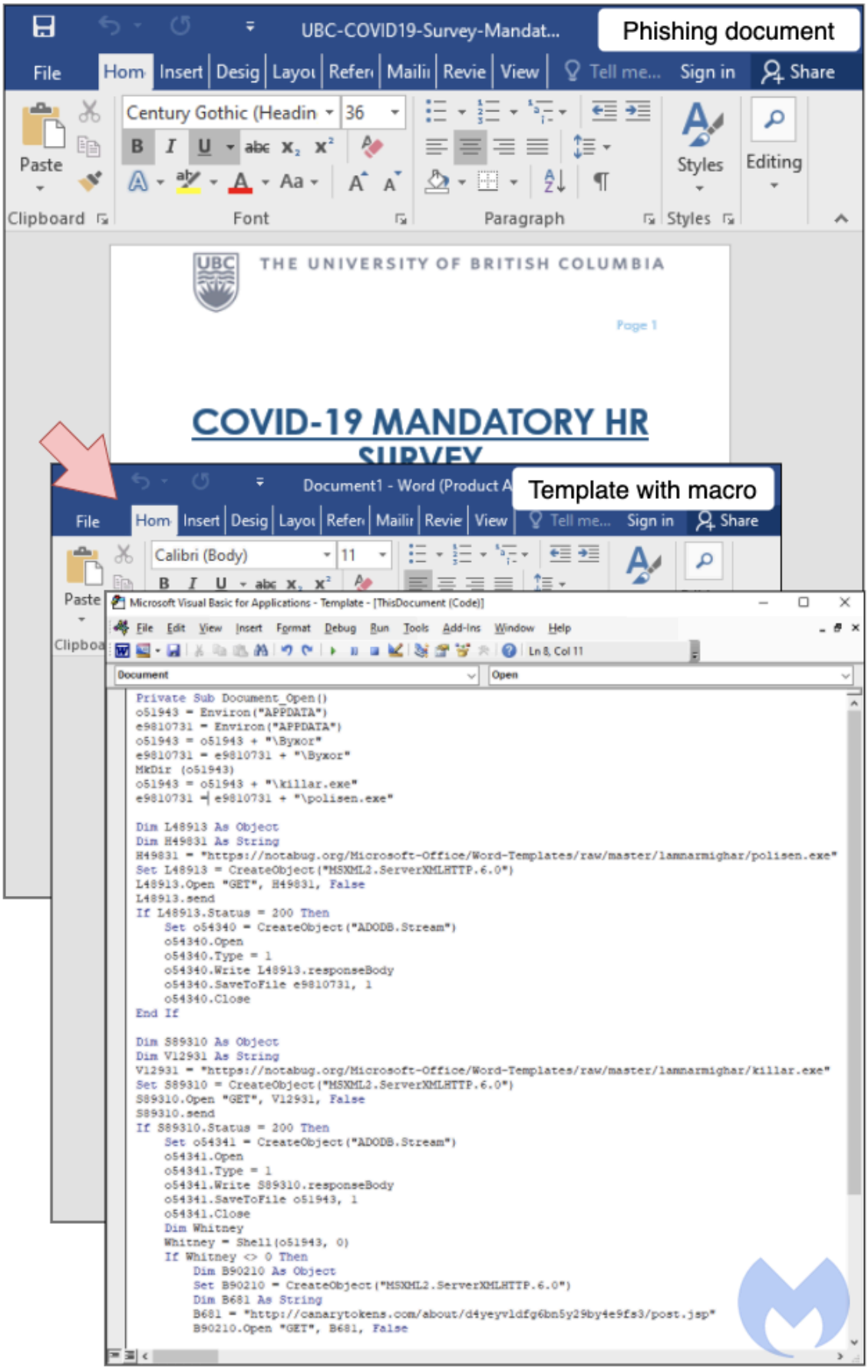
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

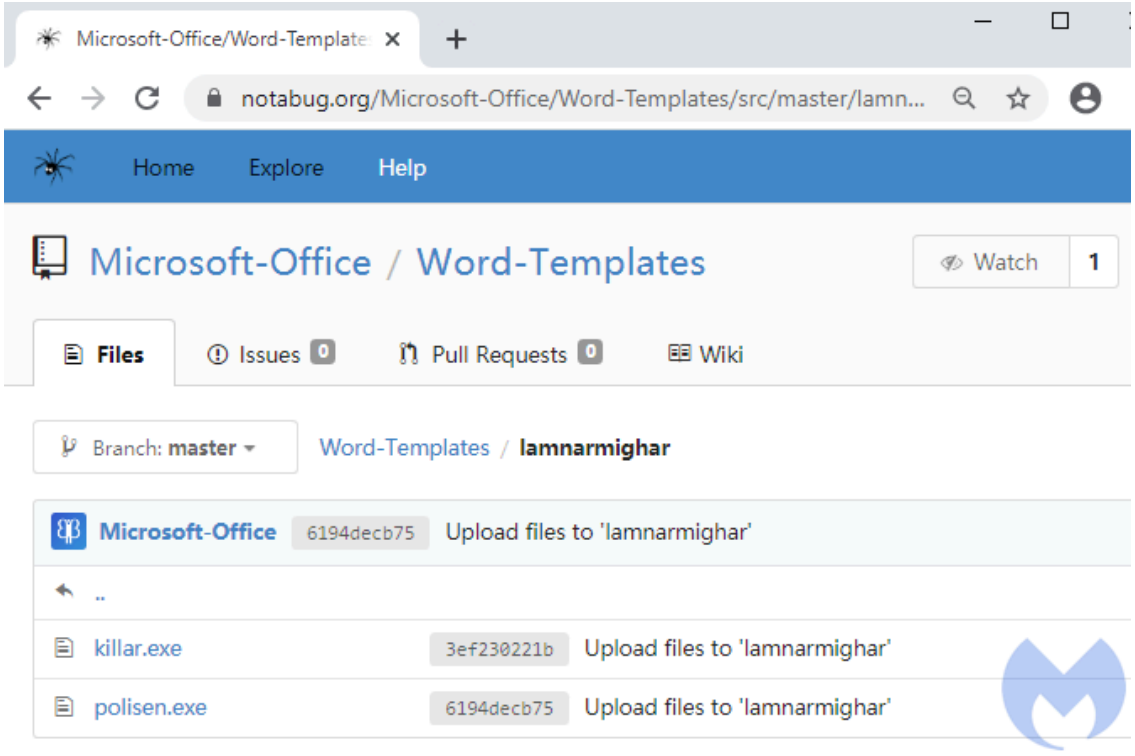
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



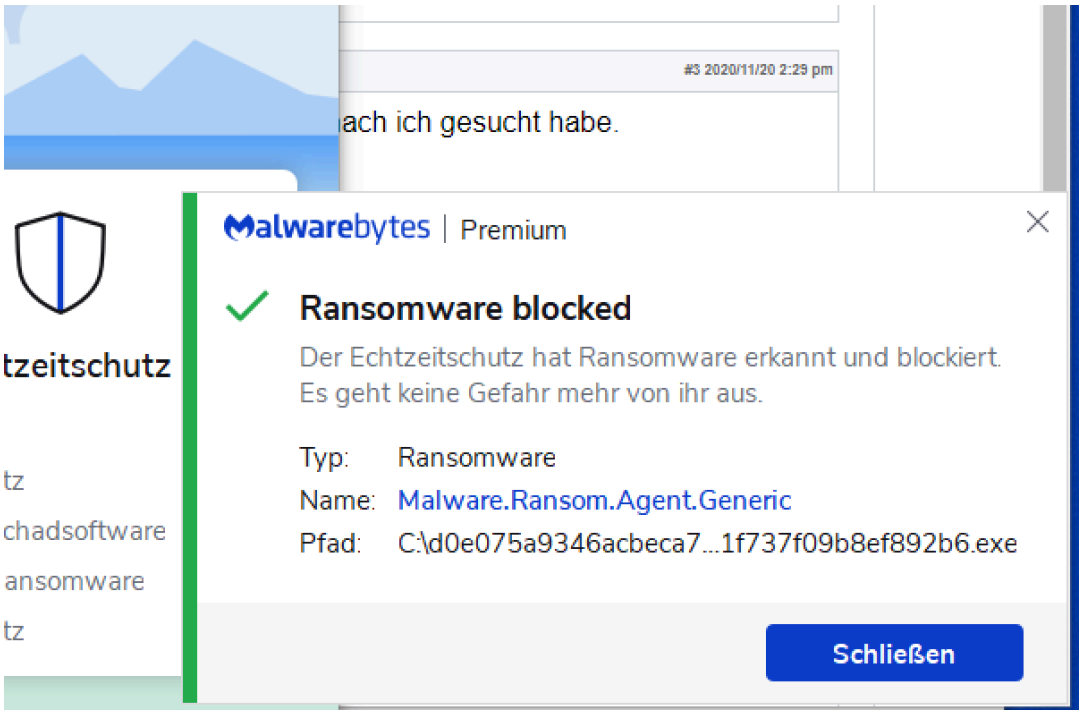
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

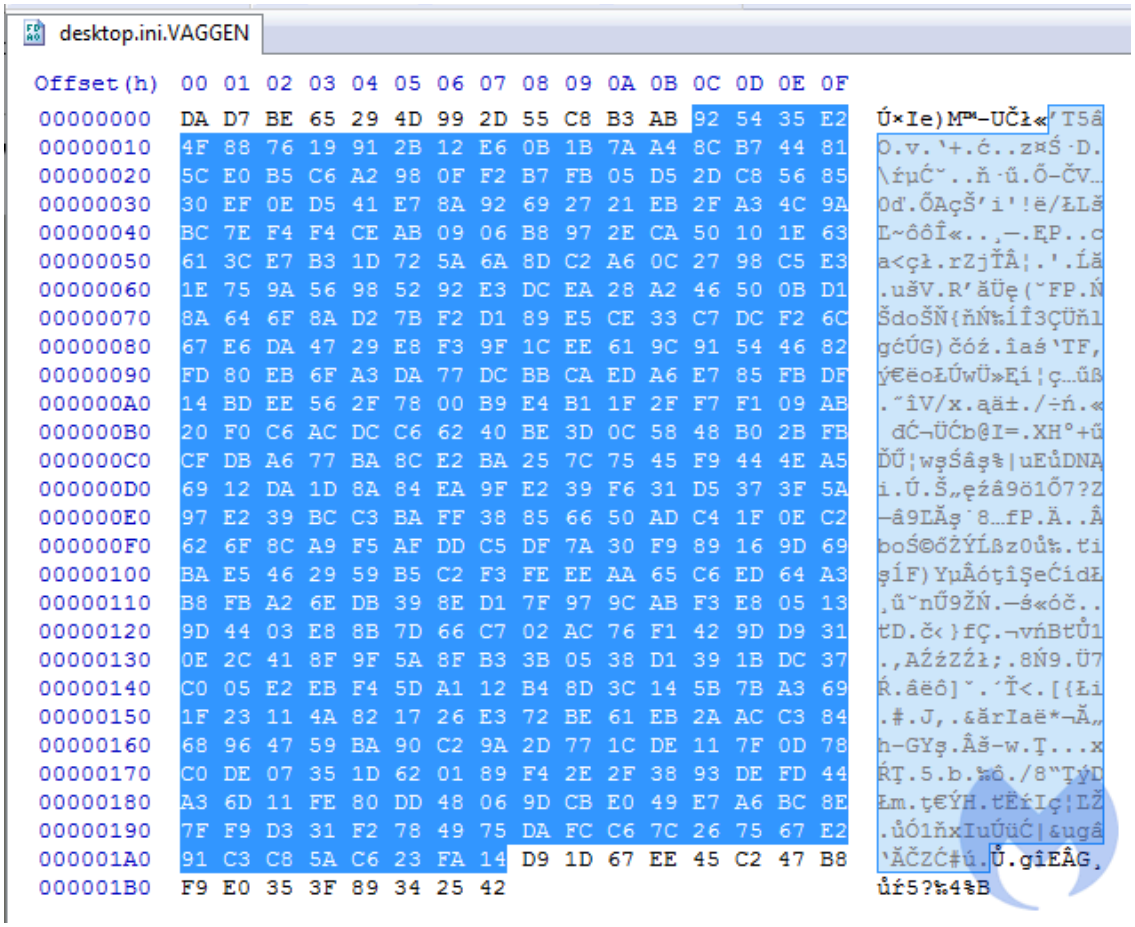
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



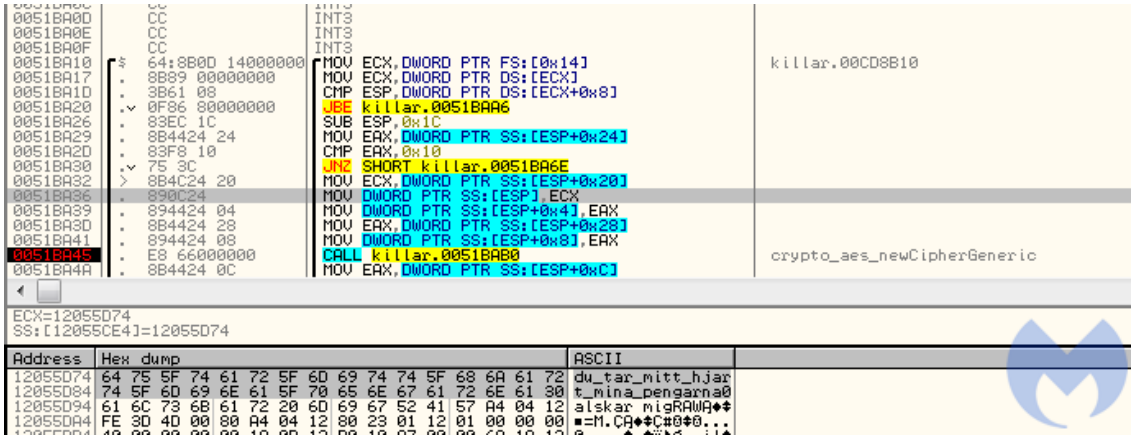
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

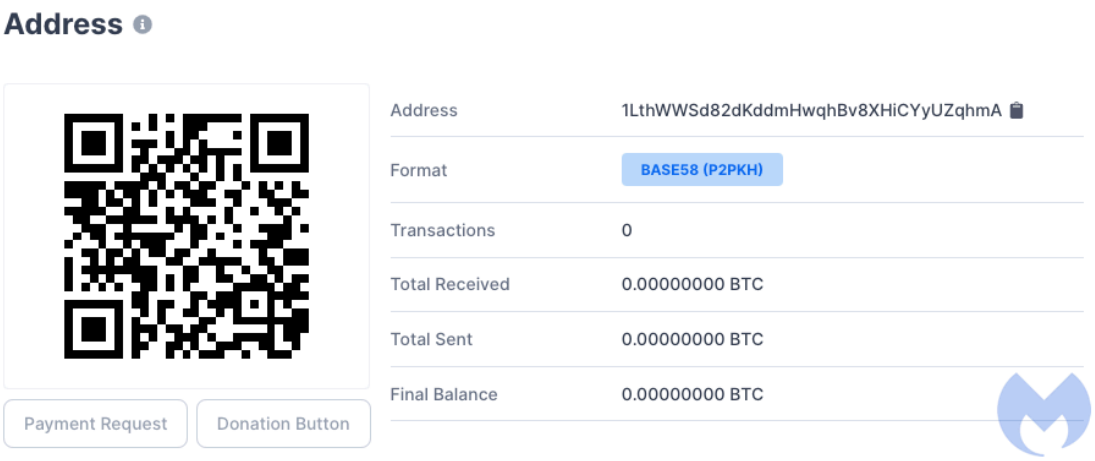
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

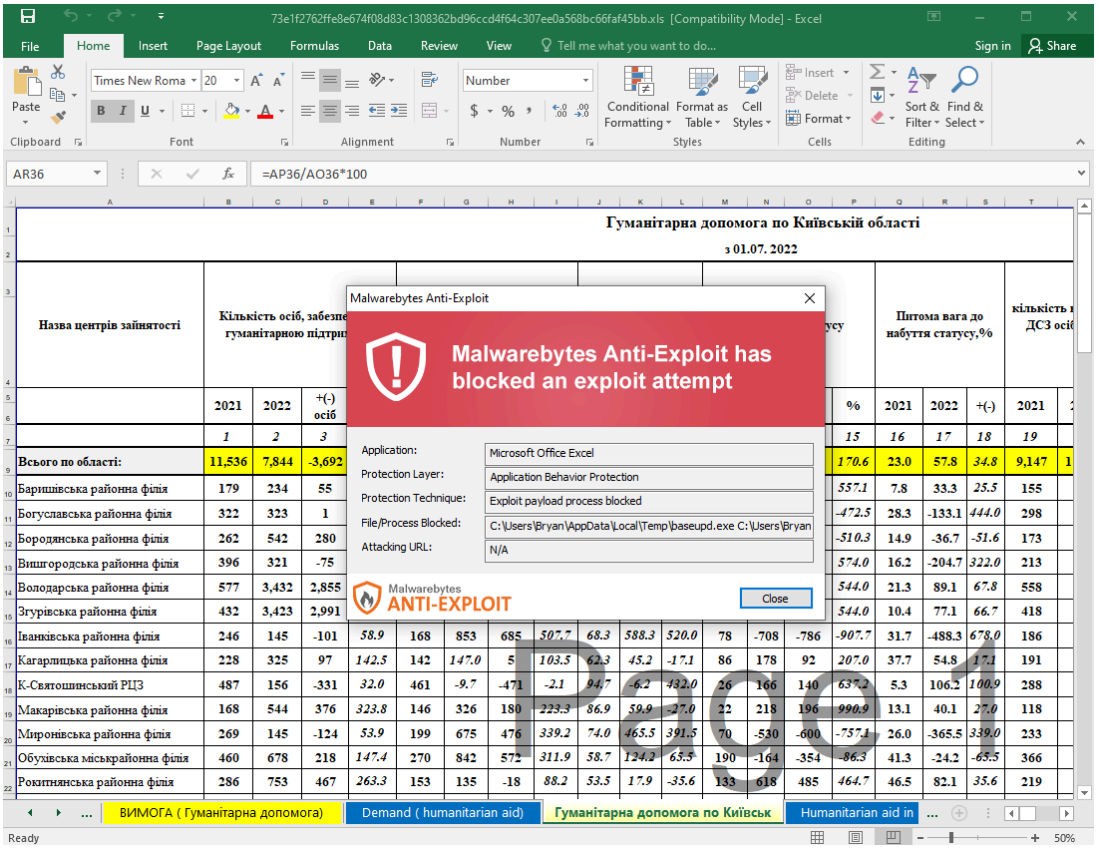
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

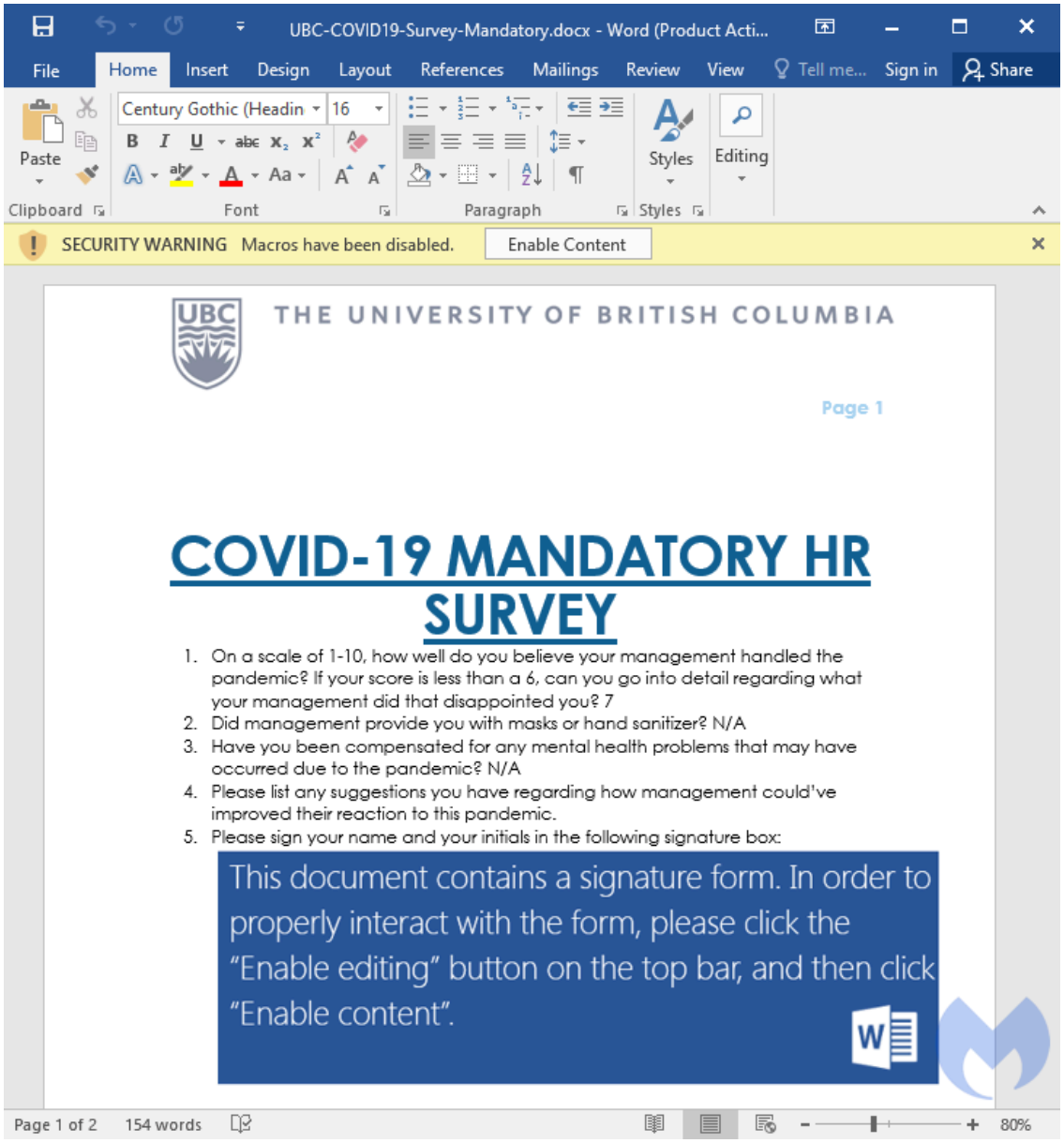
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

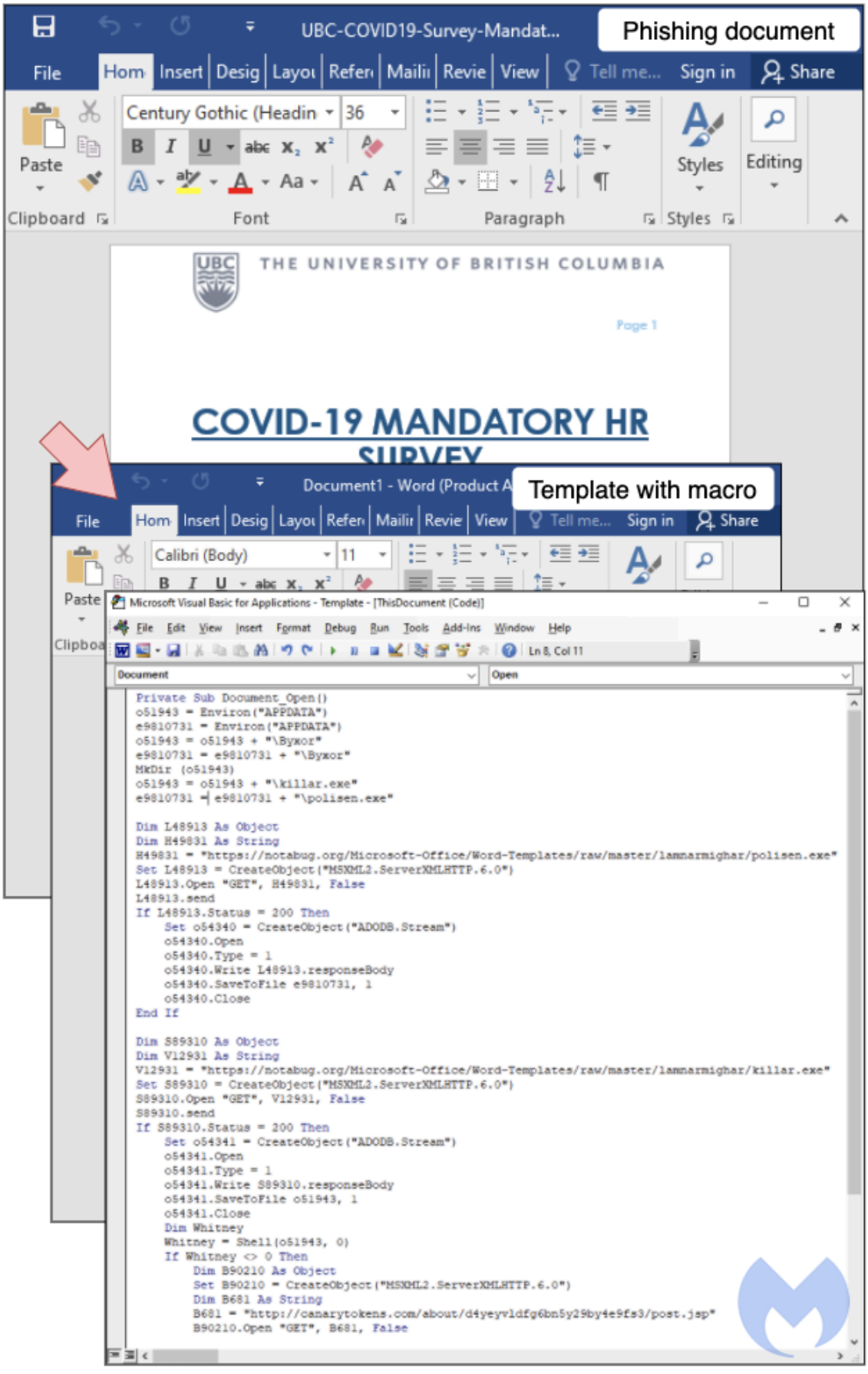
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

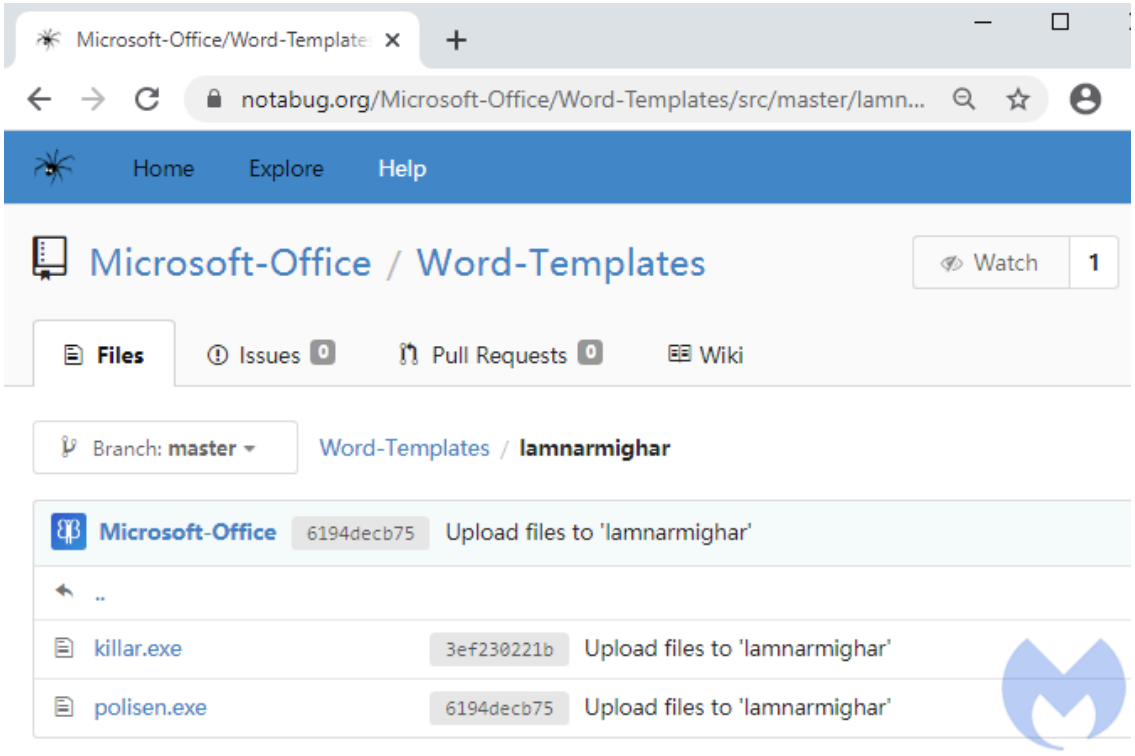
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



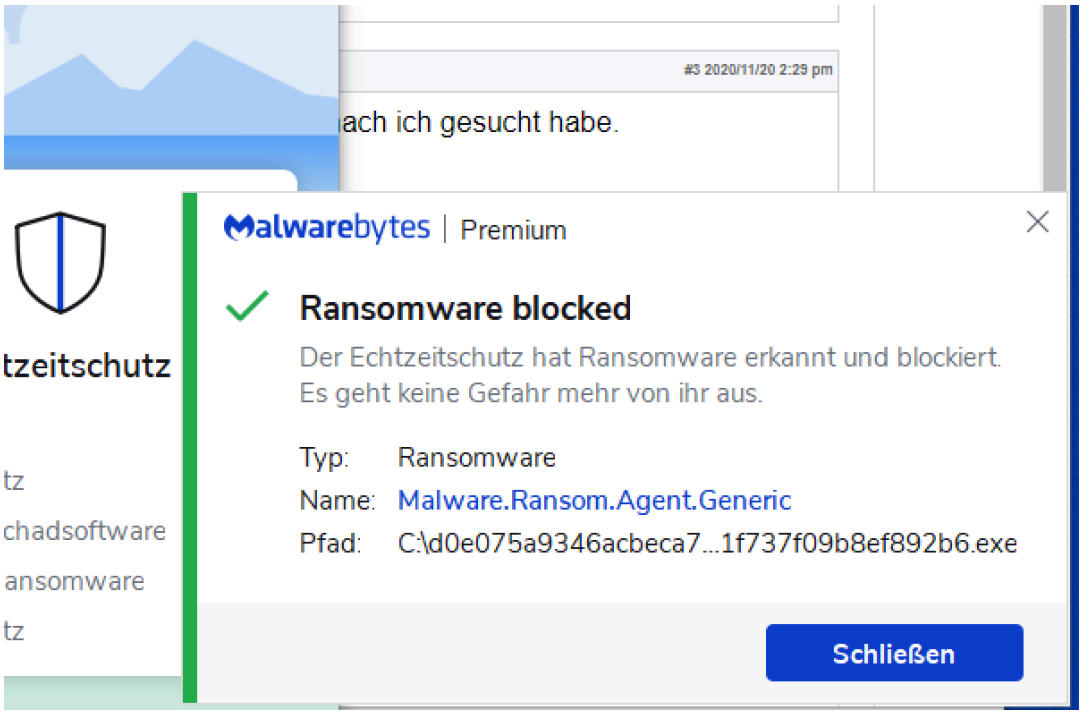
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

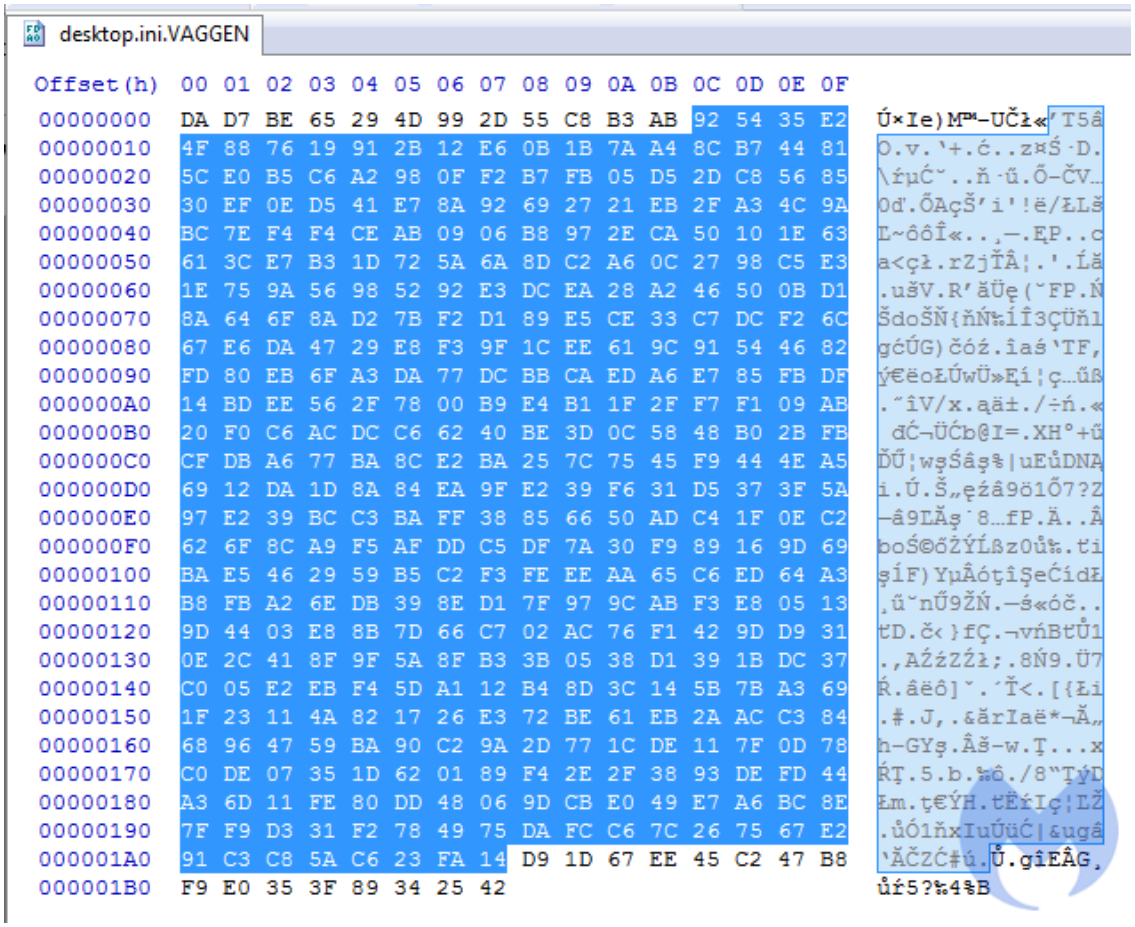
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



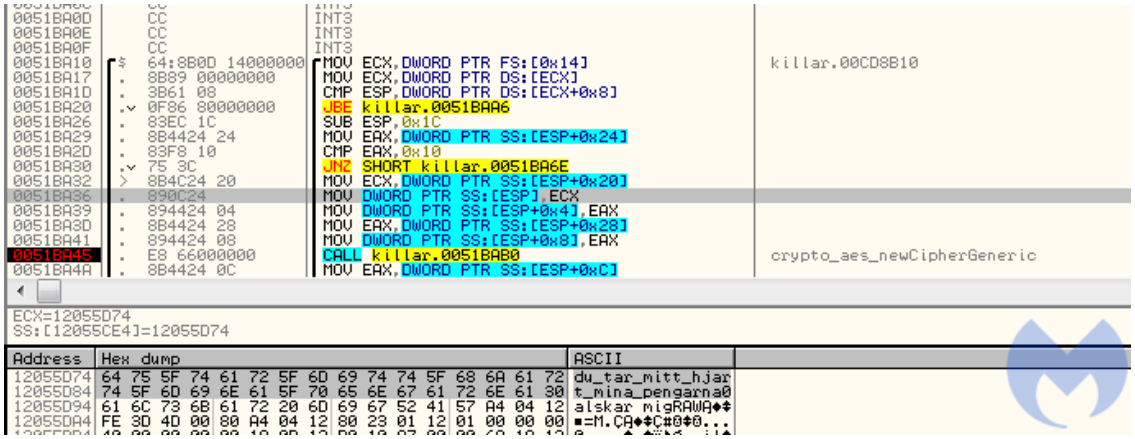
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

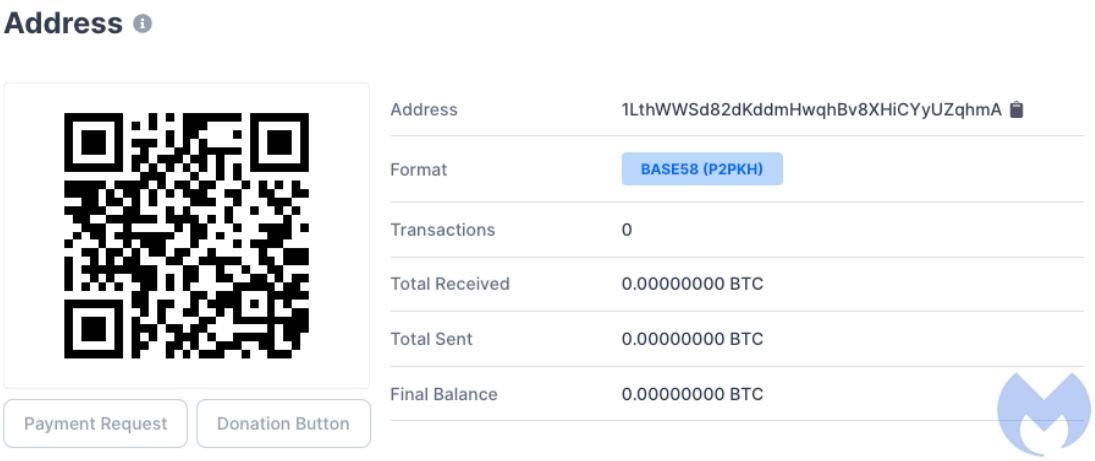
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

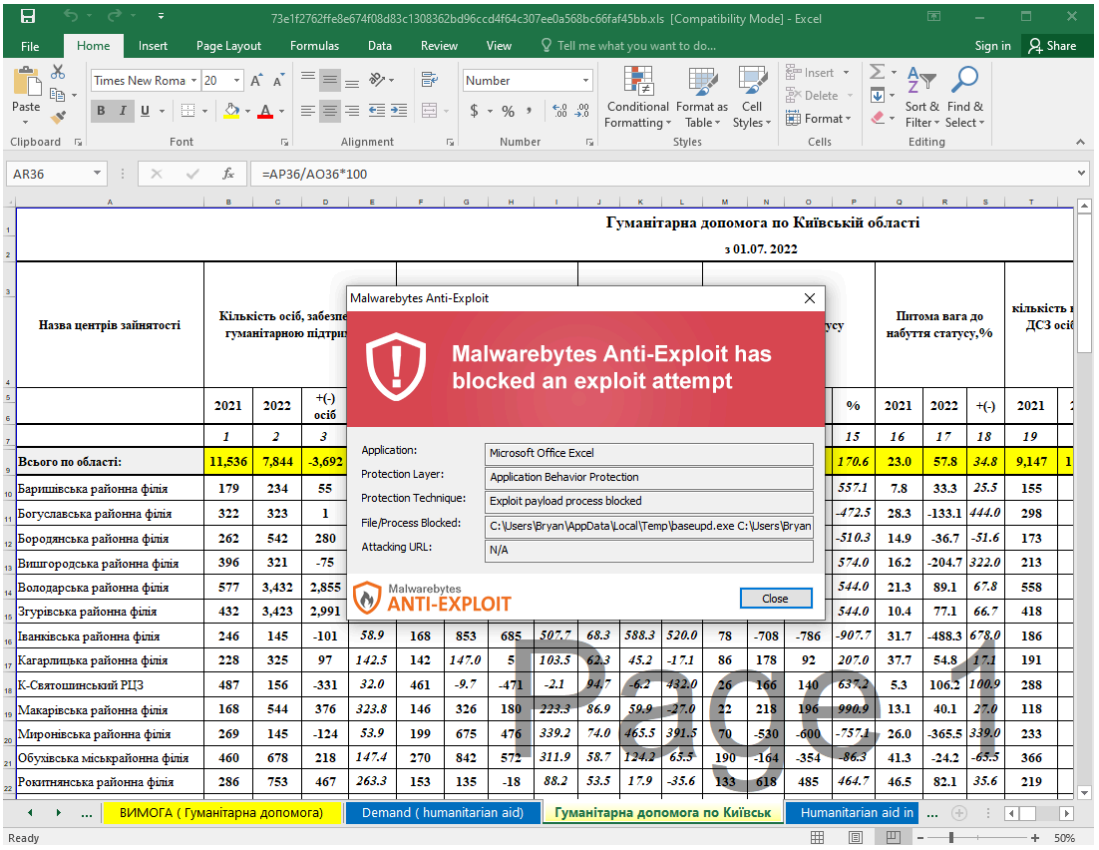
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
46  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
47    runtime_morestack_noctxt();
48  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
49  ((void (*)(void))loc_45CDA0)();
50  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
51  v38 = v4;
52  v30 = v5;
53  v31 = v6;
54  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
55  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
56  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
57  v32 = v10;
58  v11 = *(void (__cdecl **)(int))(v10 + 16);
59  v39 = v12;
60  v11(v12);
61  v29 = v13;
62  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
63  v40 = v15;
64  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
65  v17 = v27;
66  if ( v27 )
67  {
68    if ( v27 )
69      v17 = *(_DWORD *)(v27 + 4);
70    v43 = v17;
71    v44 = v28;
72    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
73  }
```

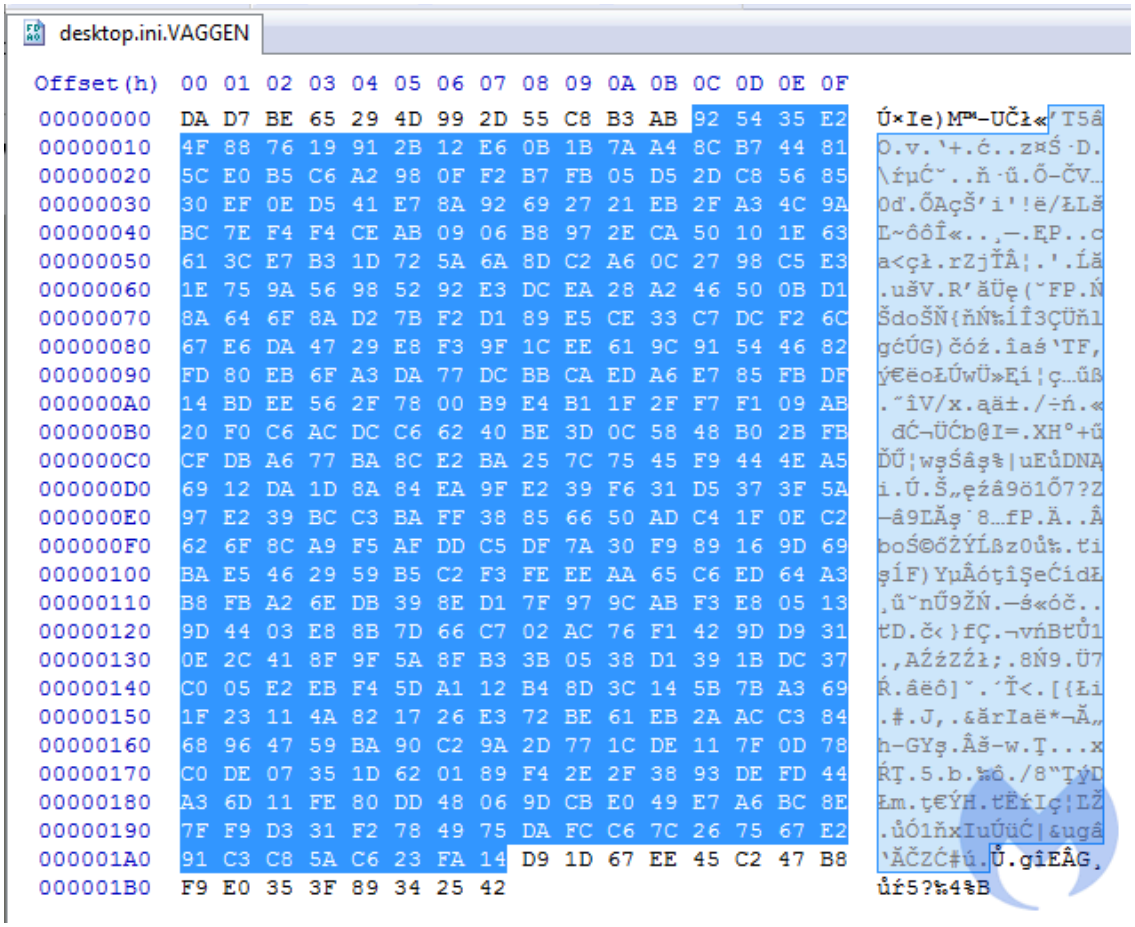


The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

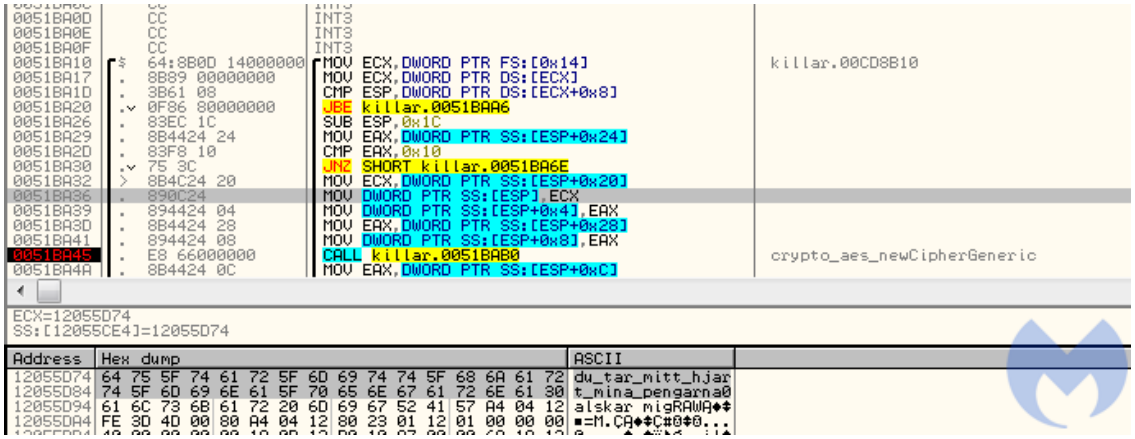


The content of the output file (with .VAGGEN extension) contains:

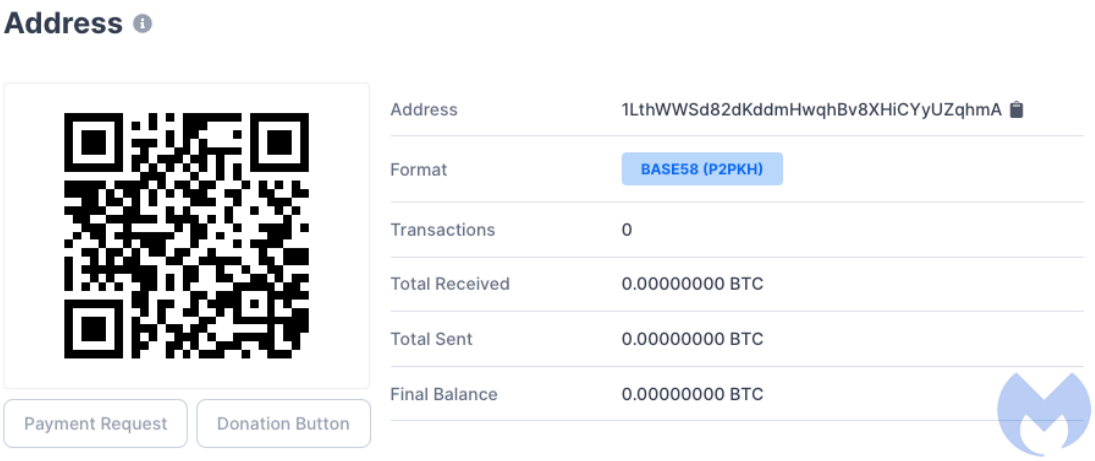
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

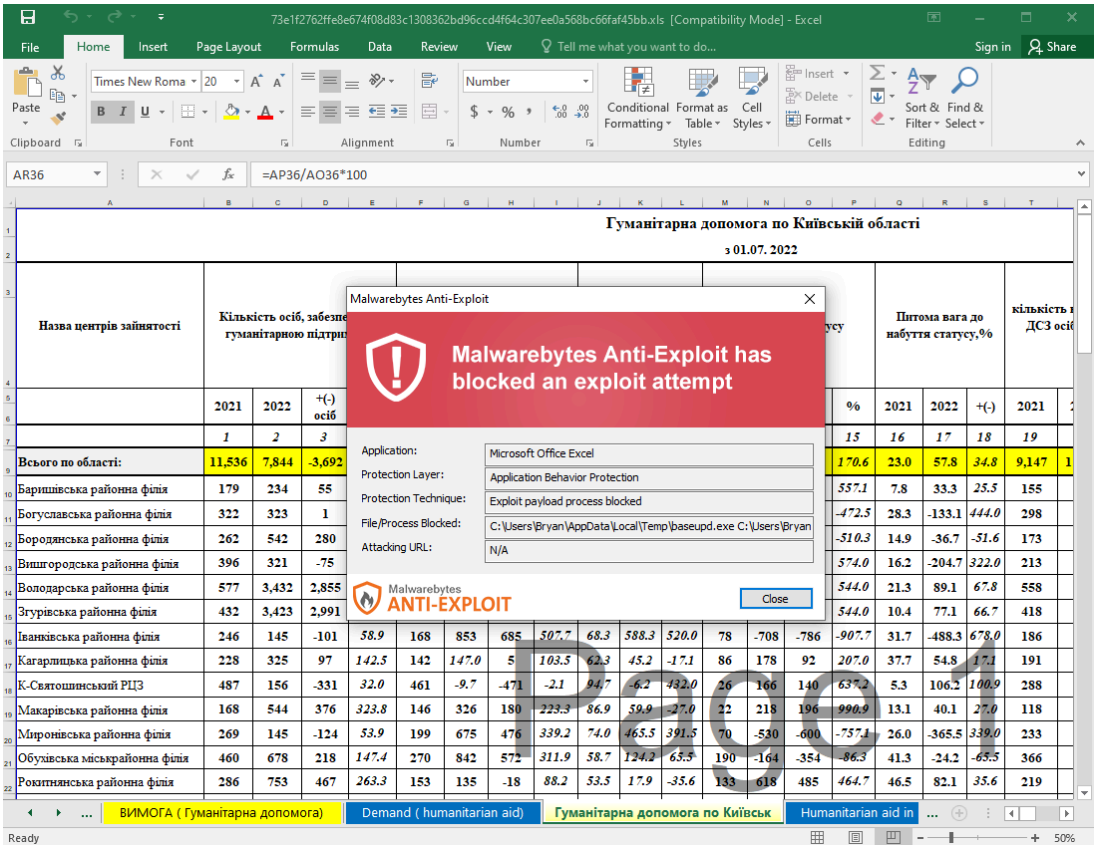
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

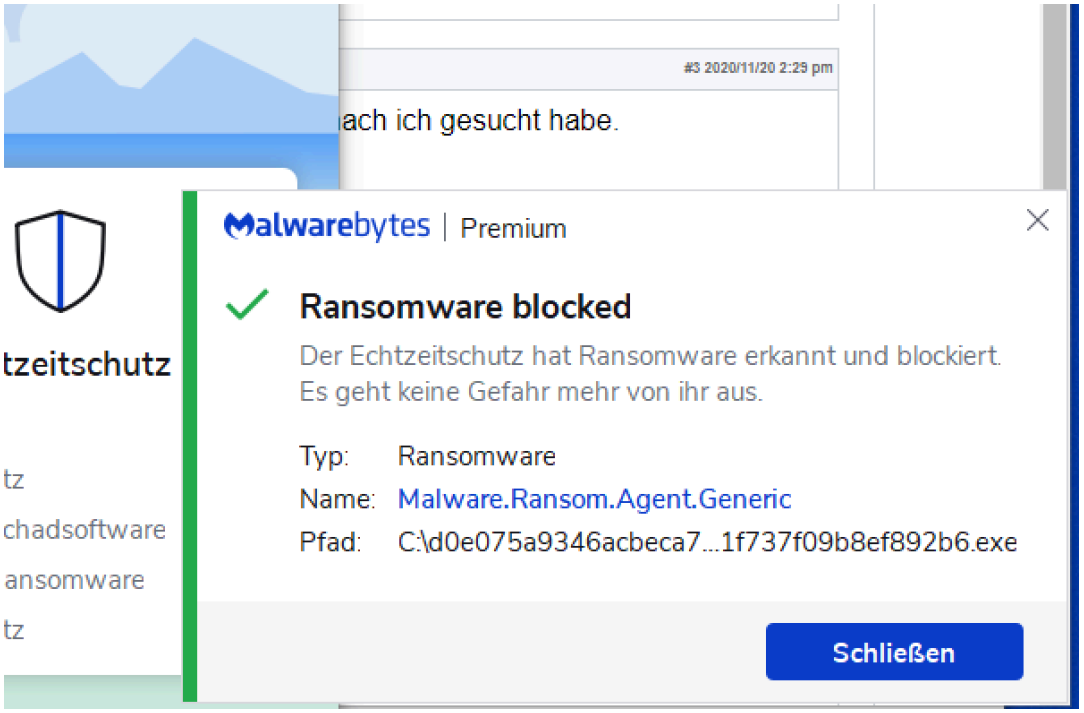
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

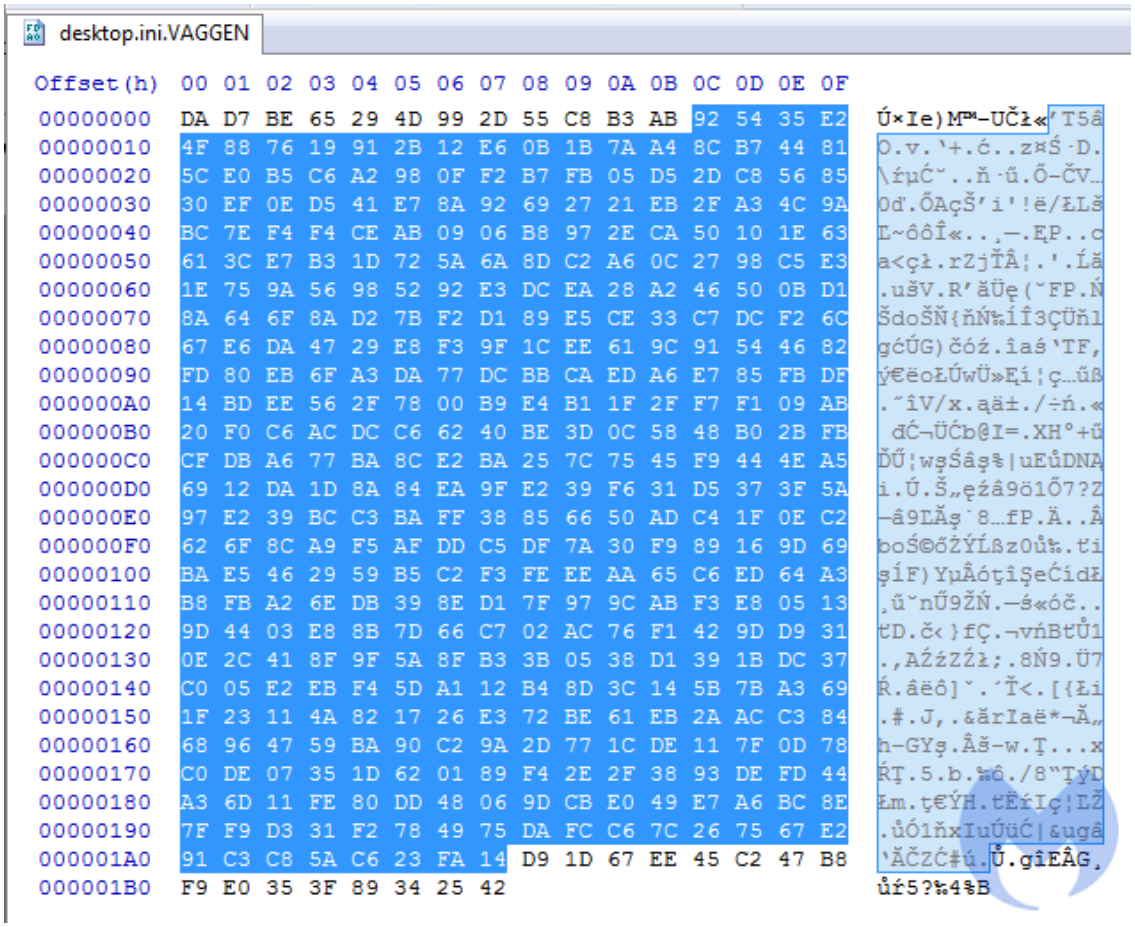
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

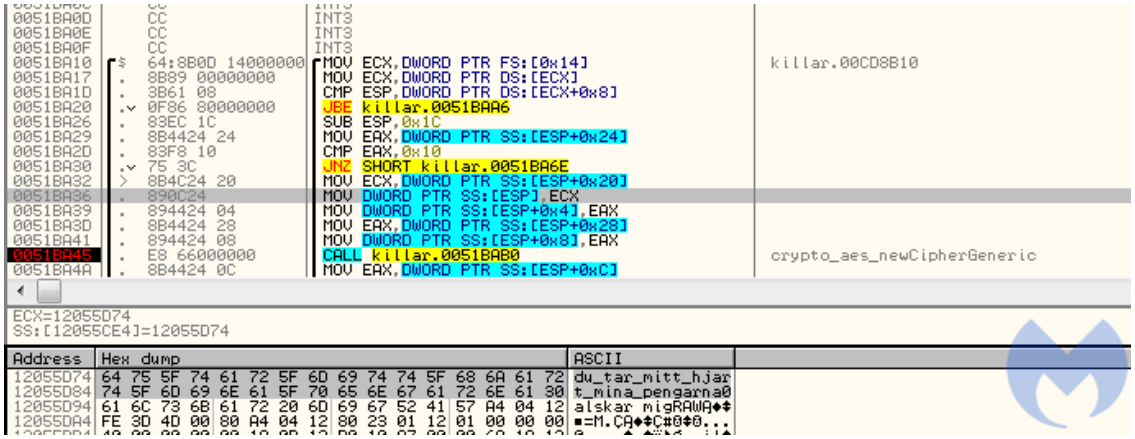


The content of the output file (with .VAGGEN extension) contains:

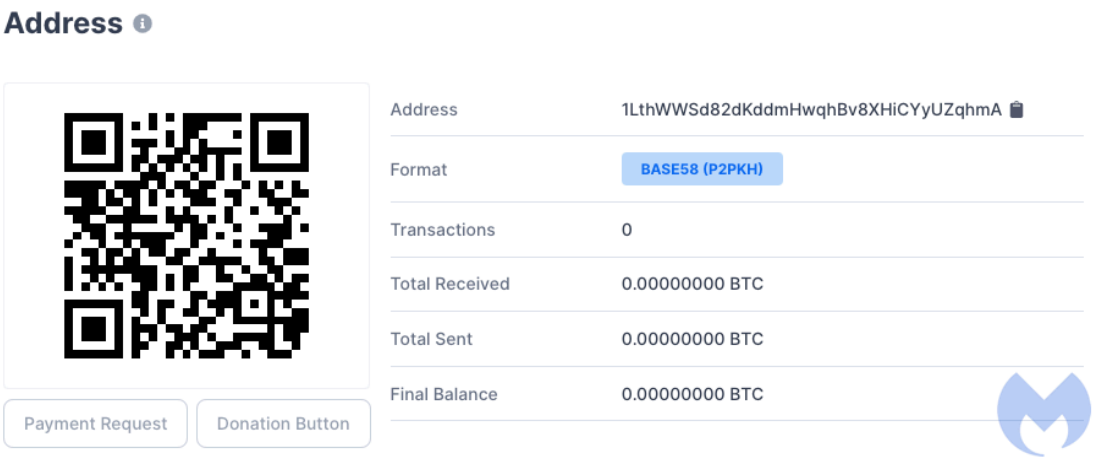
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

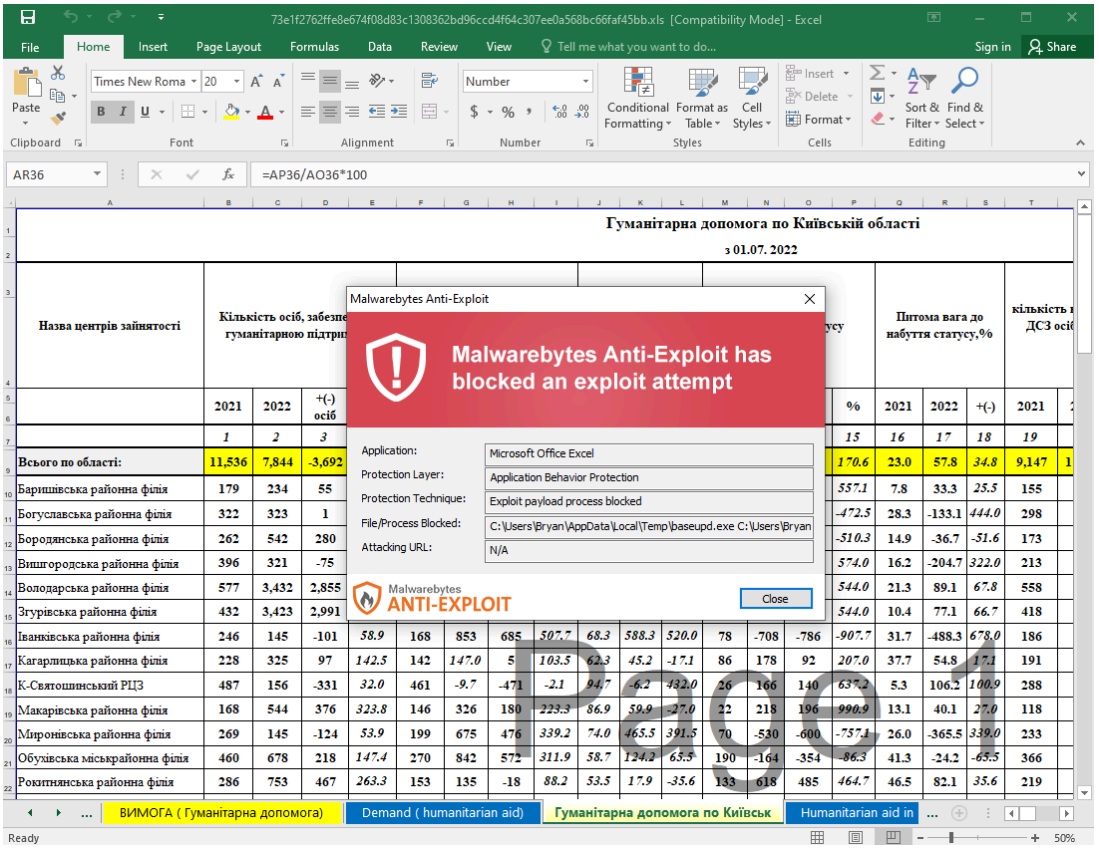
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

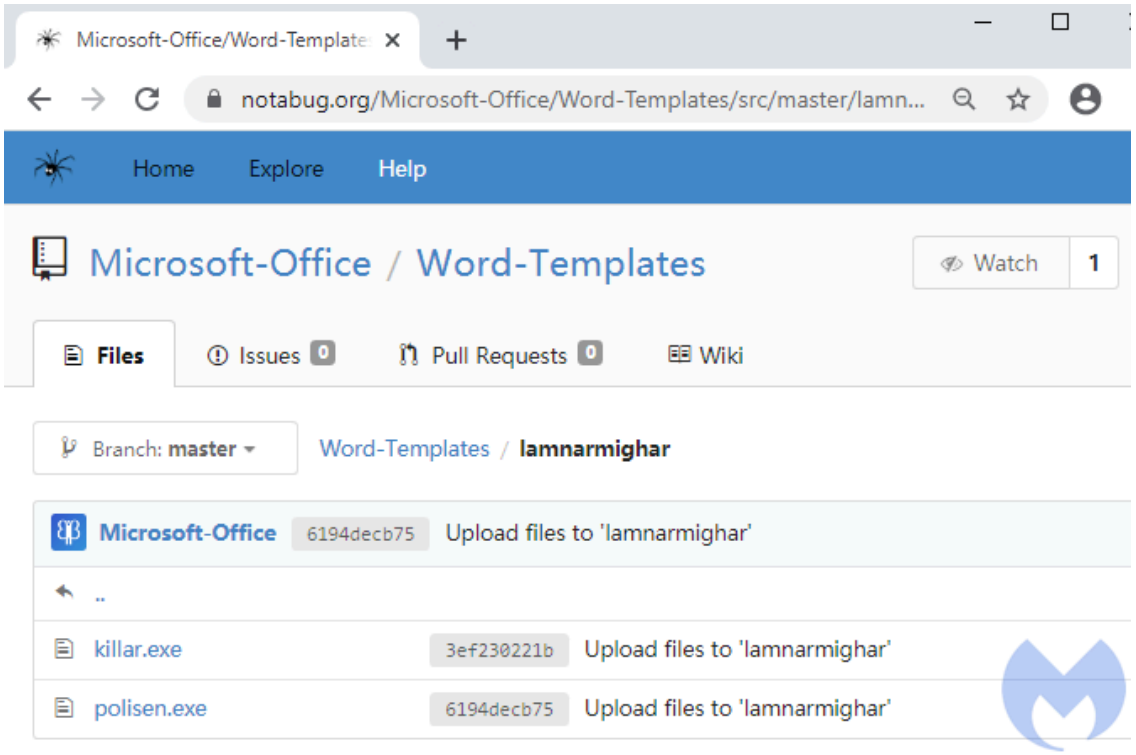
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



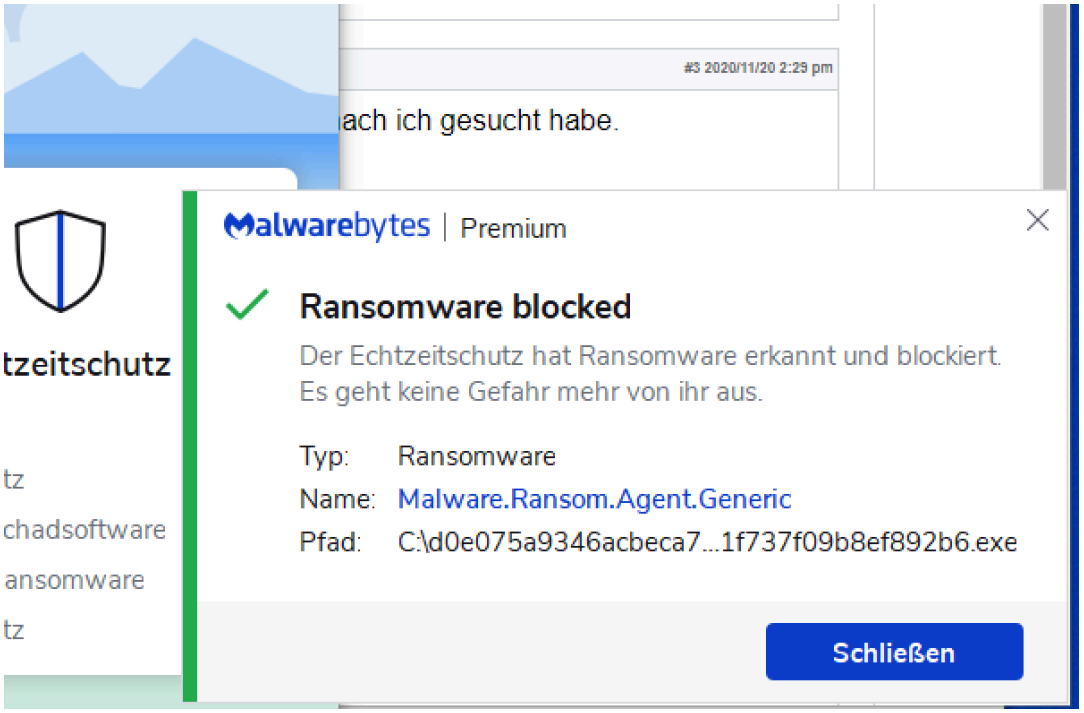
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

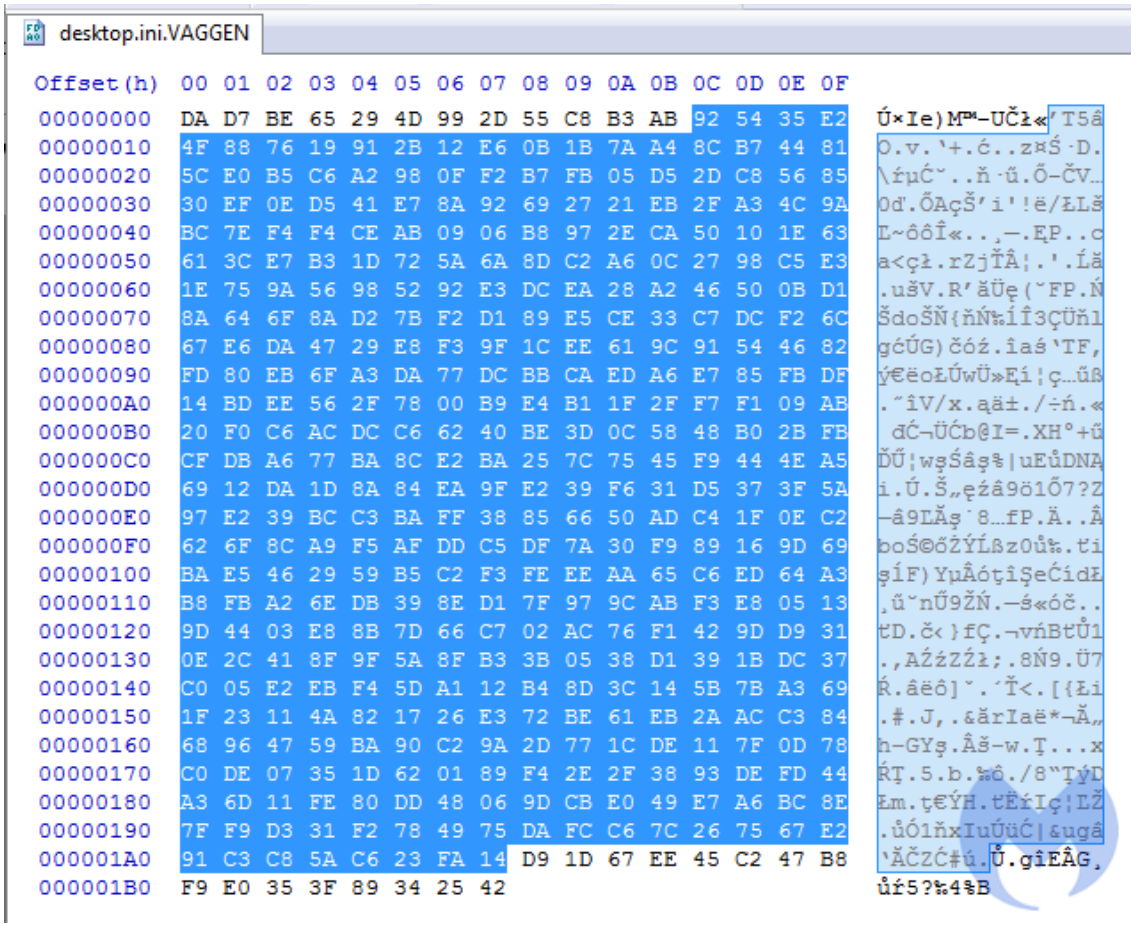
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



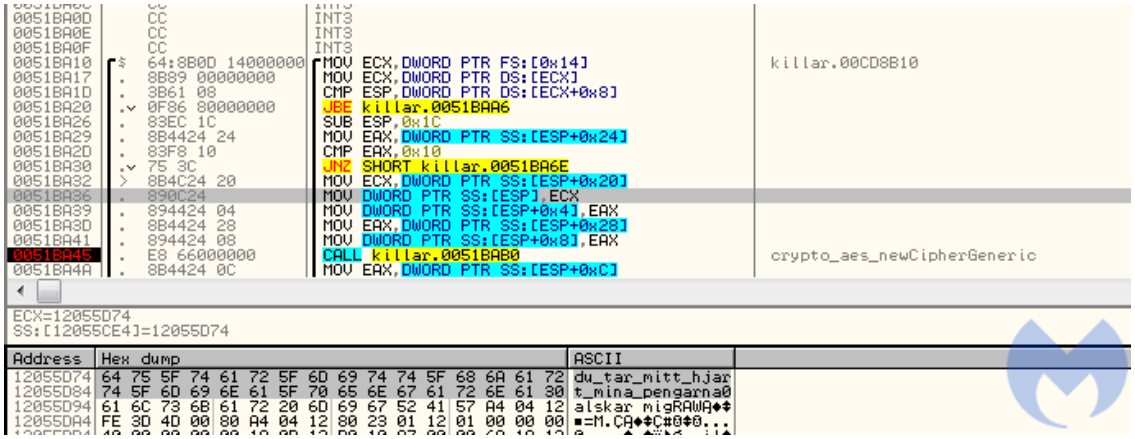
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

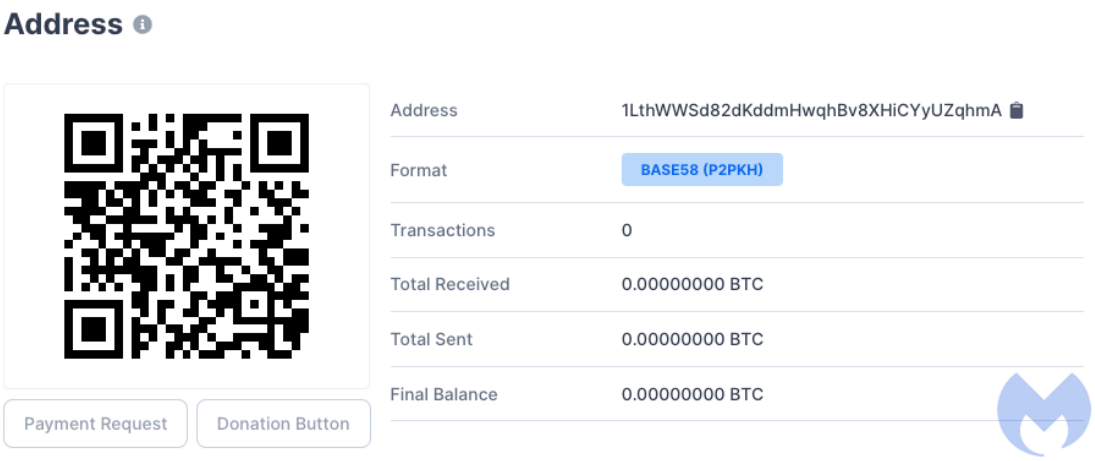
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

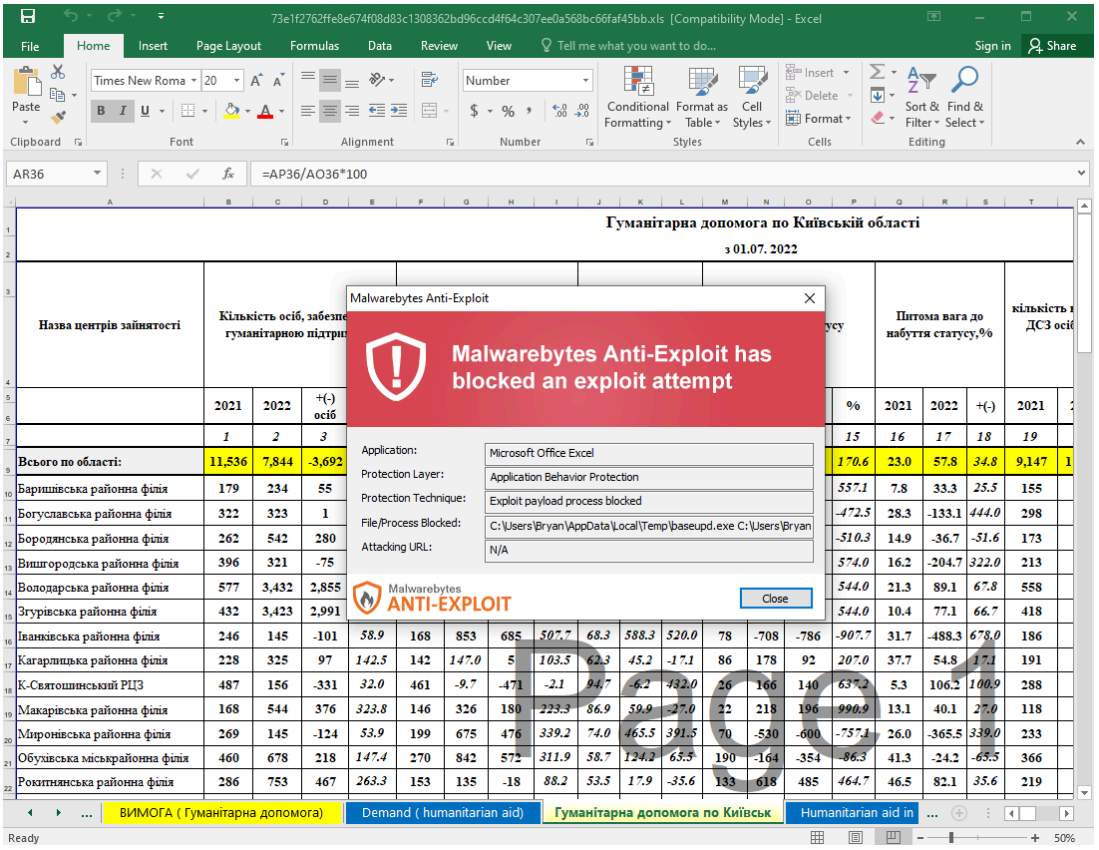
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

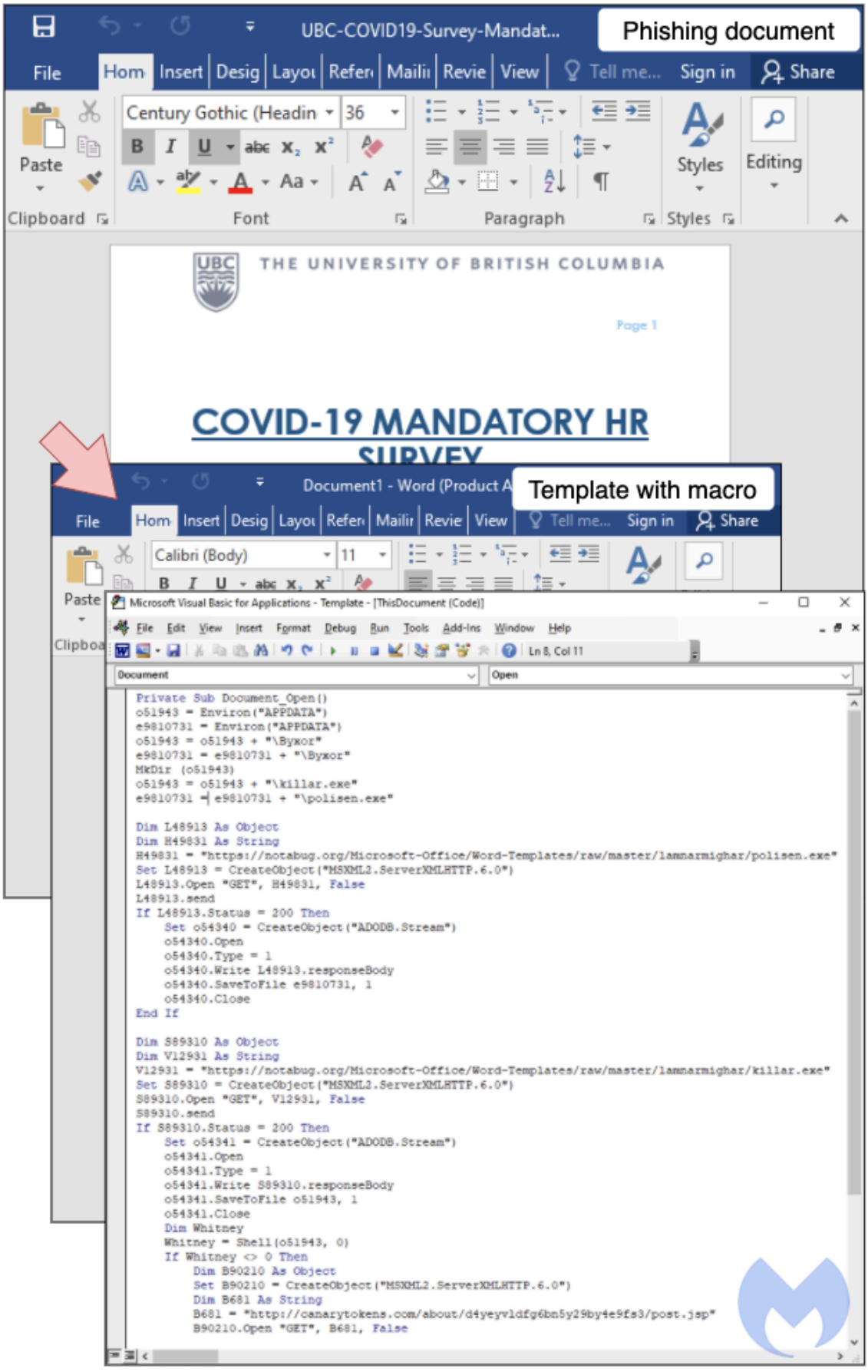
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

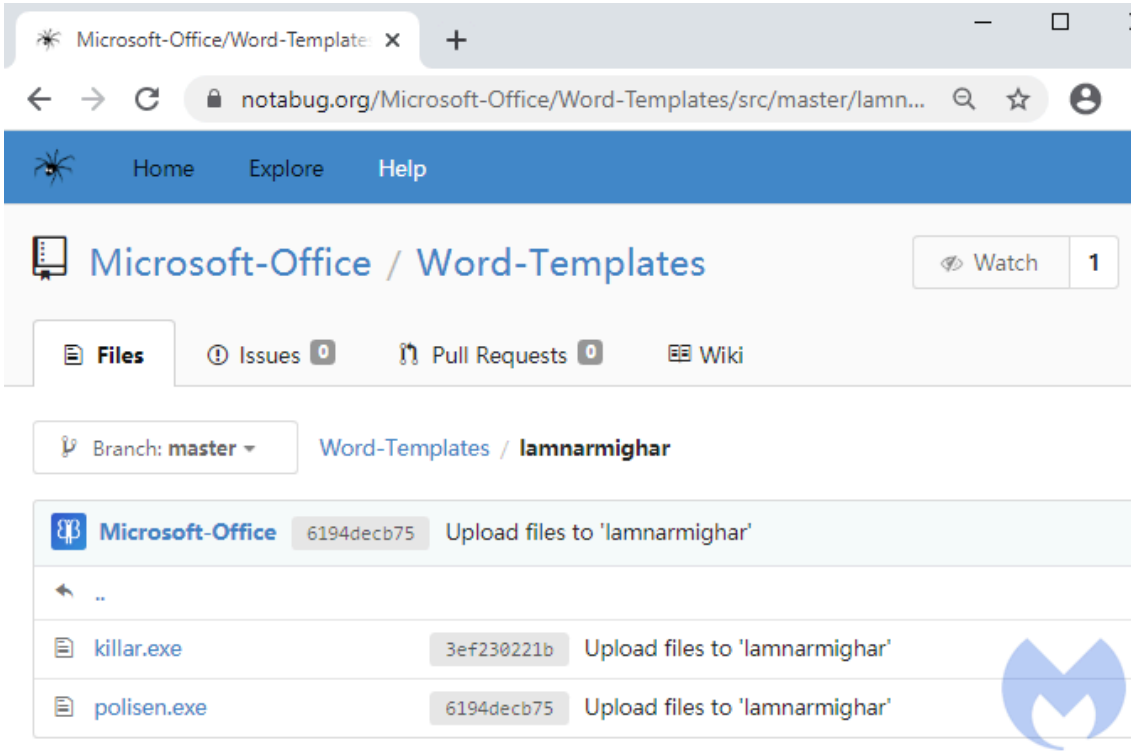
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



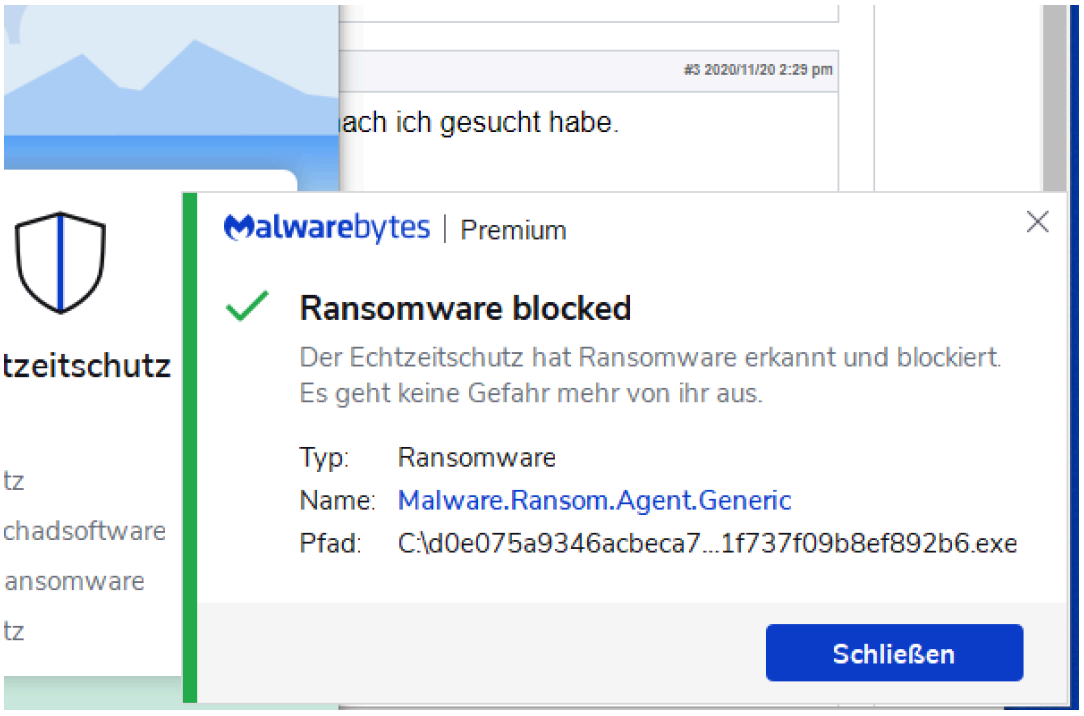
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

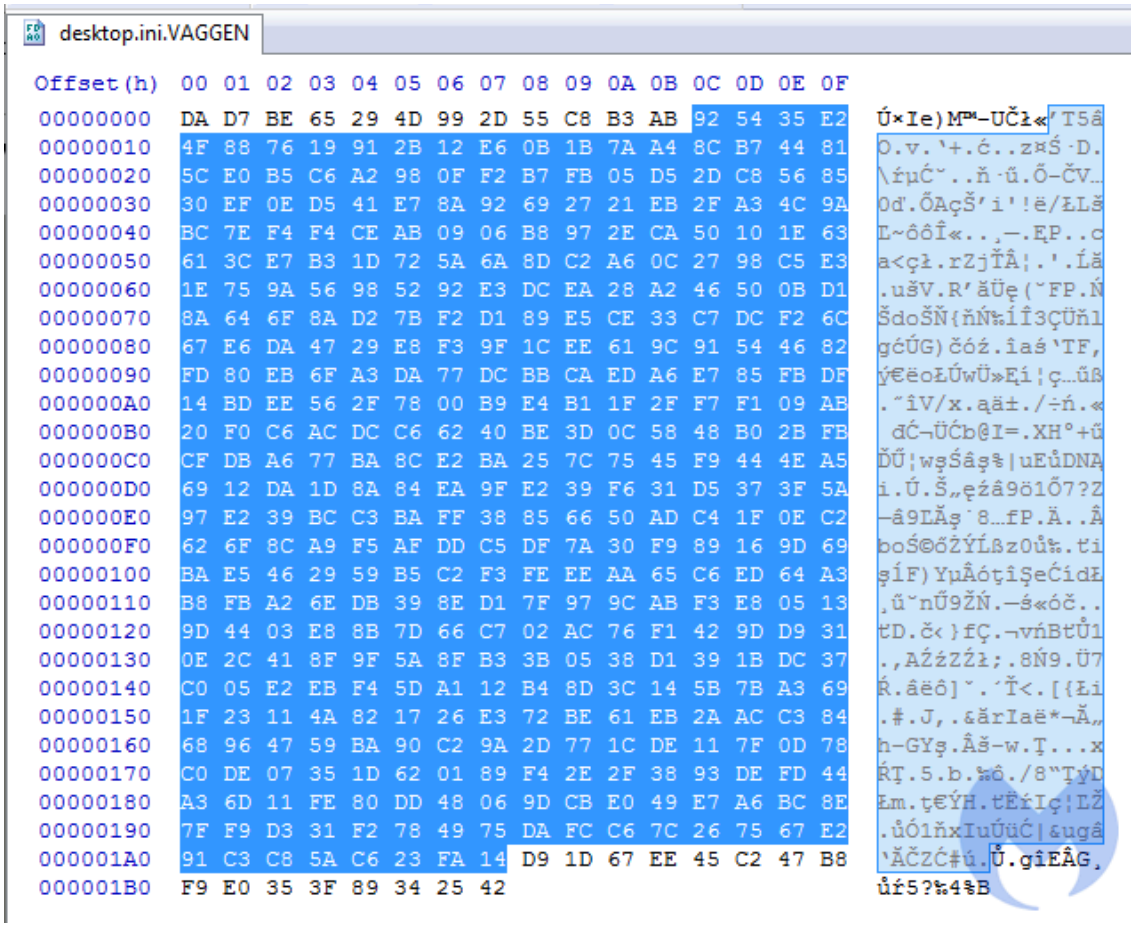
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

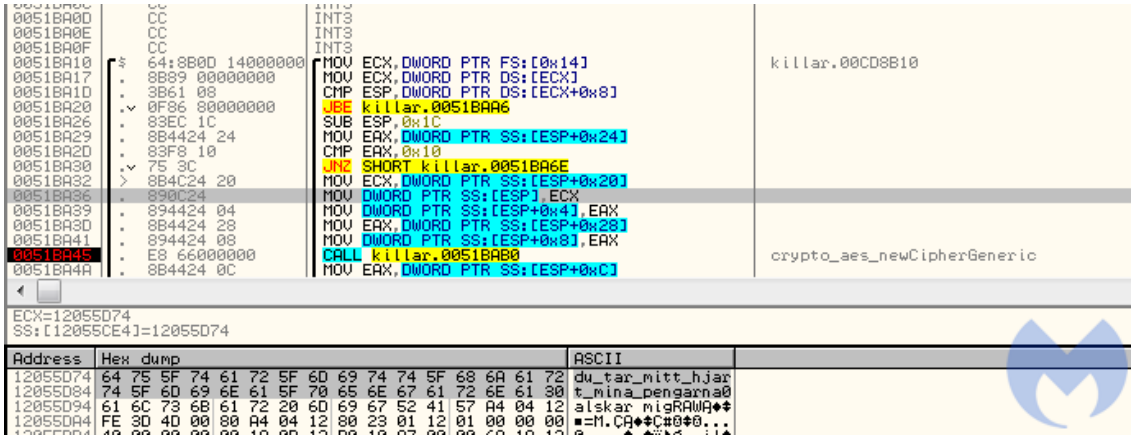


The content of the output file (with .VAGGEN extension) contains:

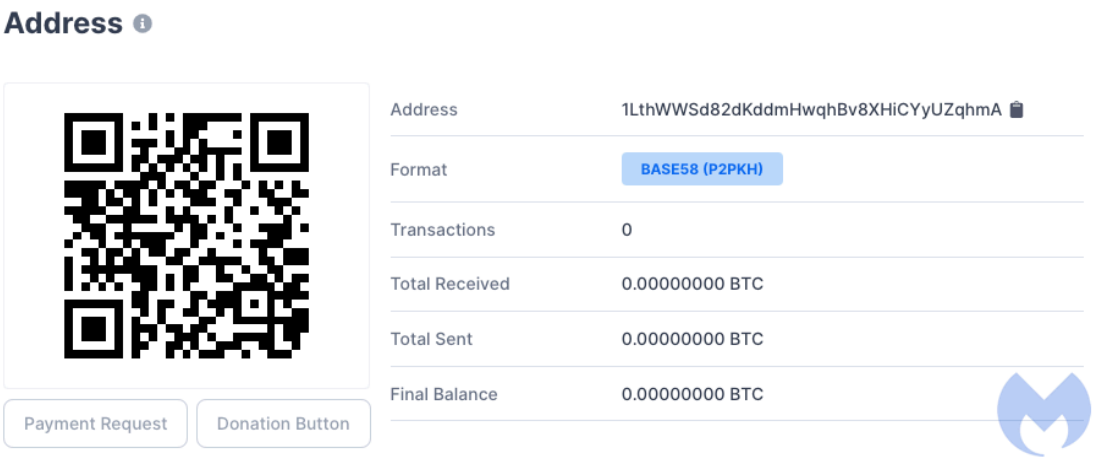
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

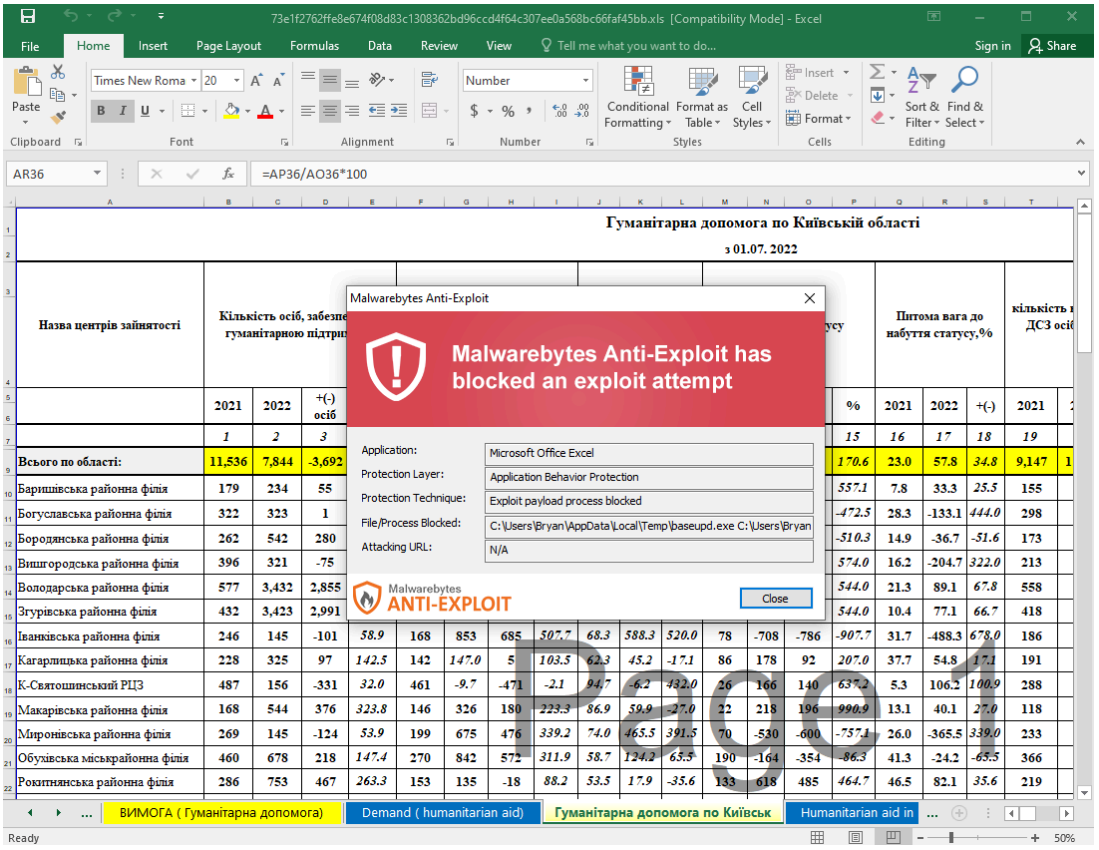
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

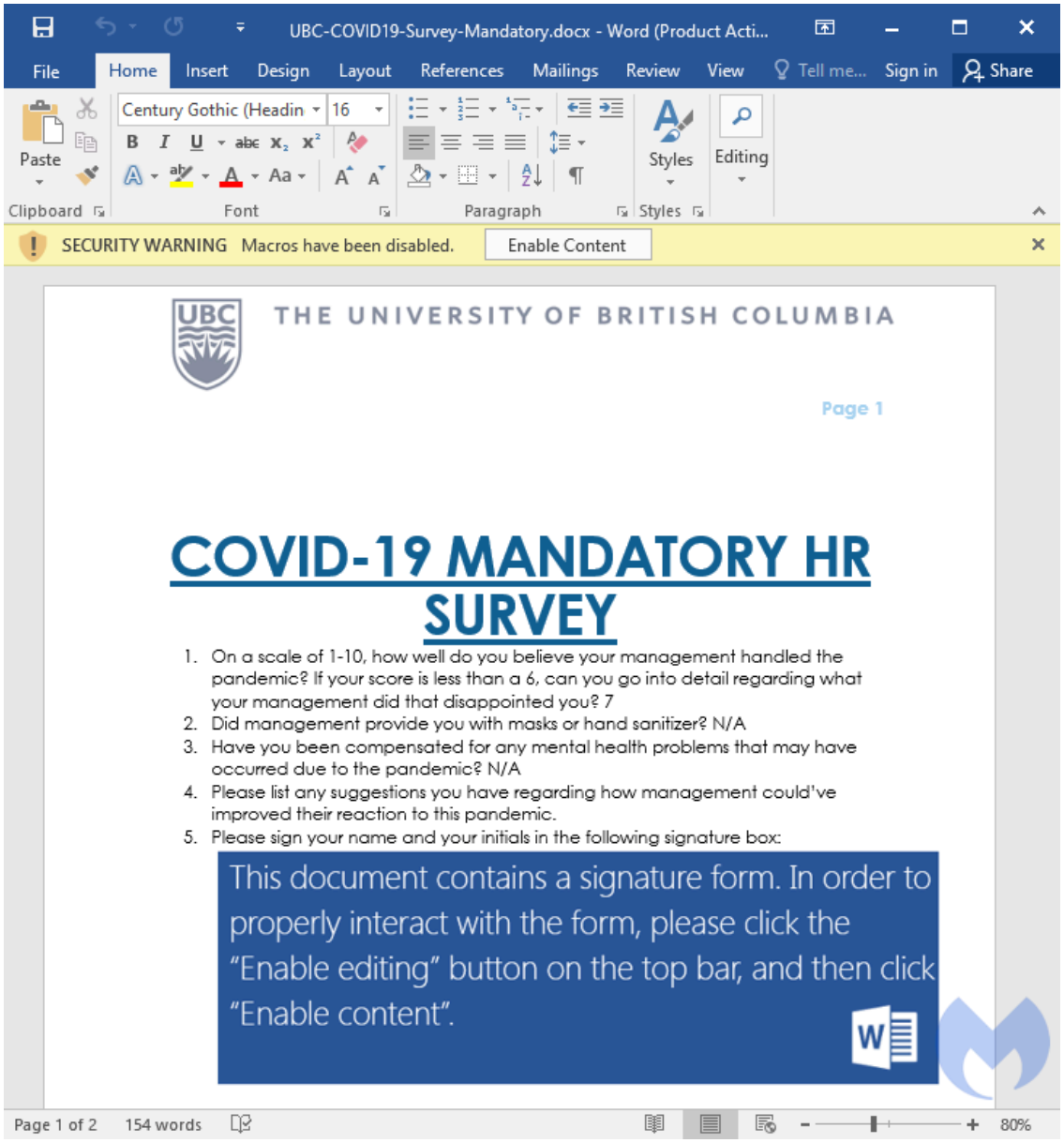
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

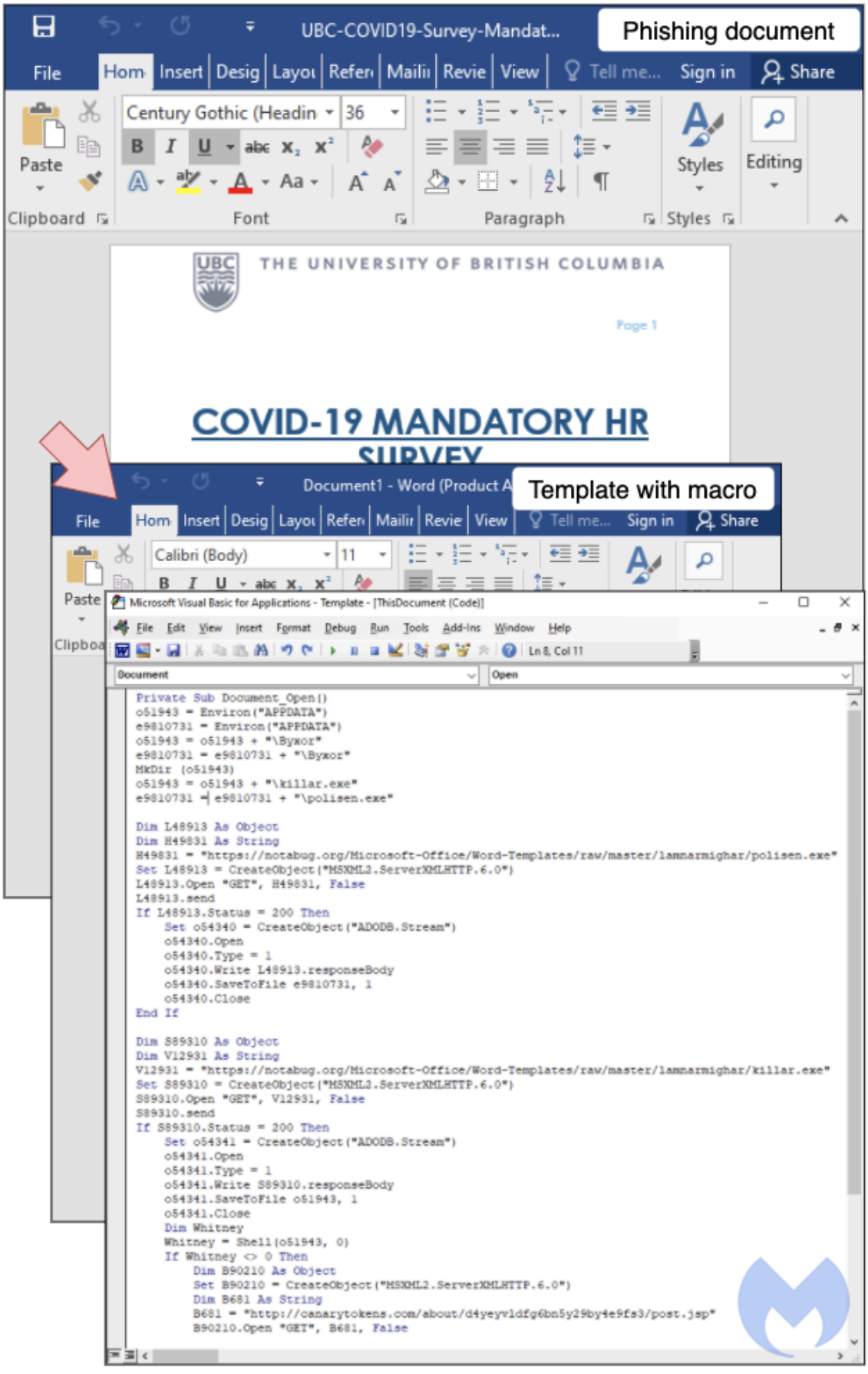
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

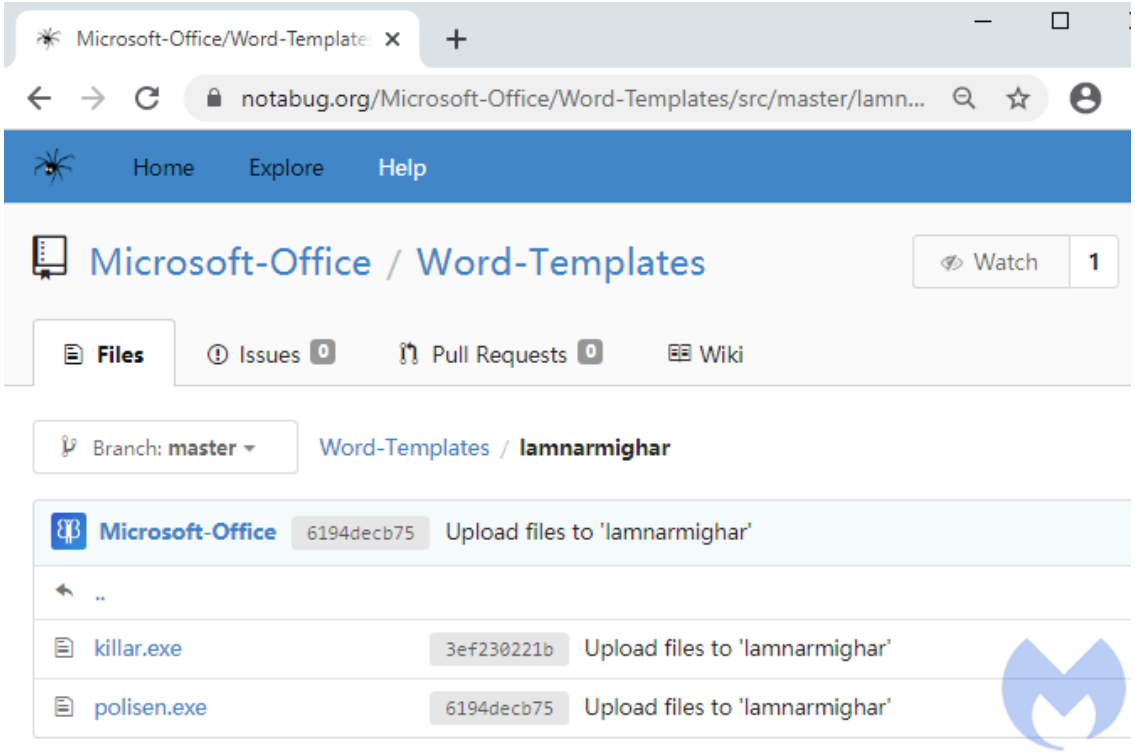
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



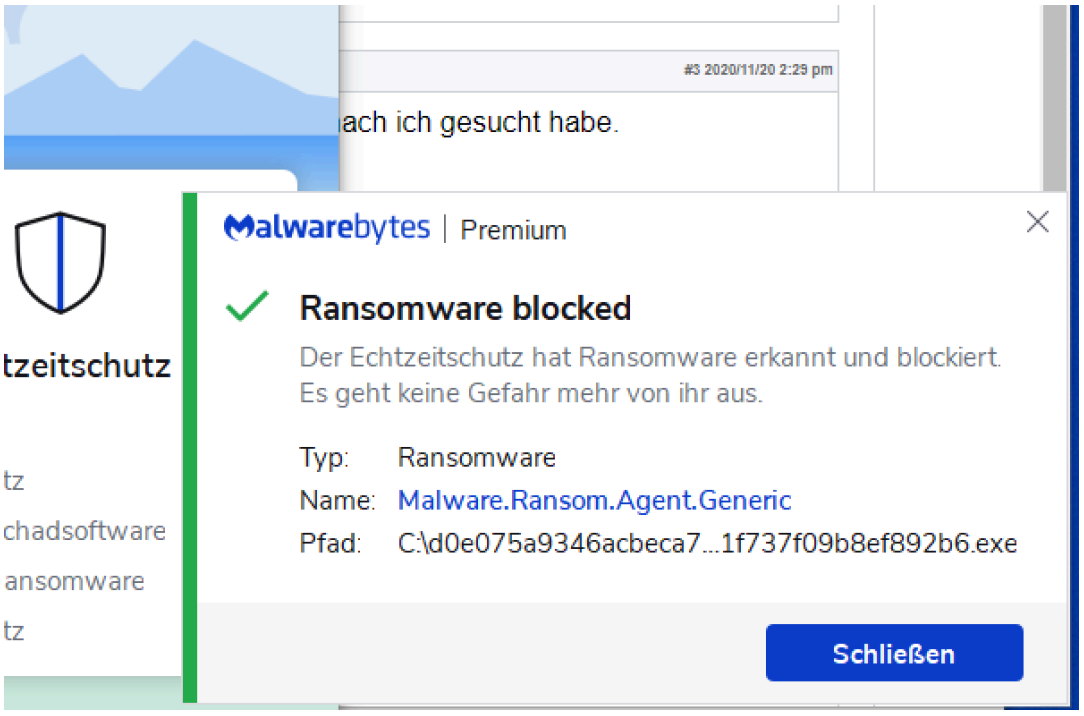
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

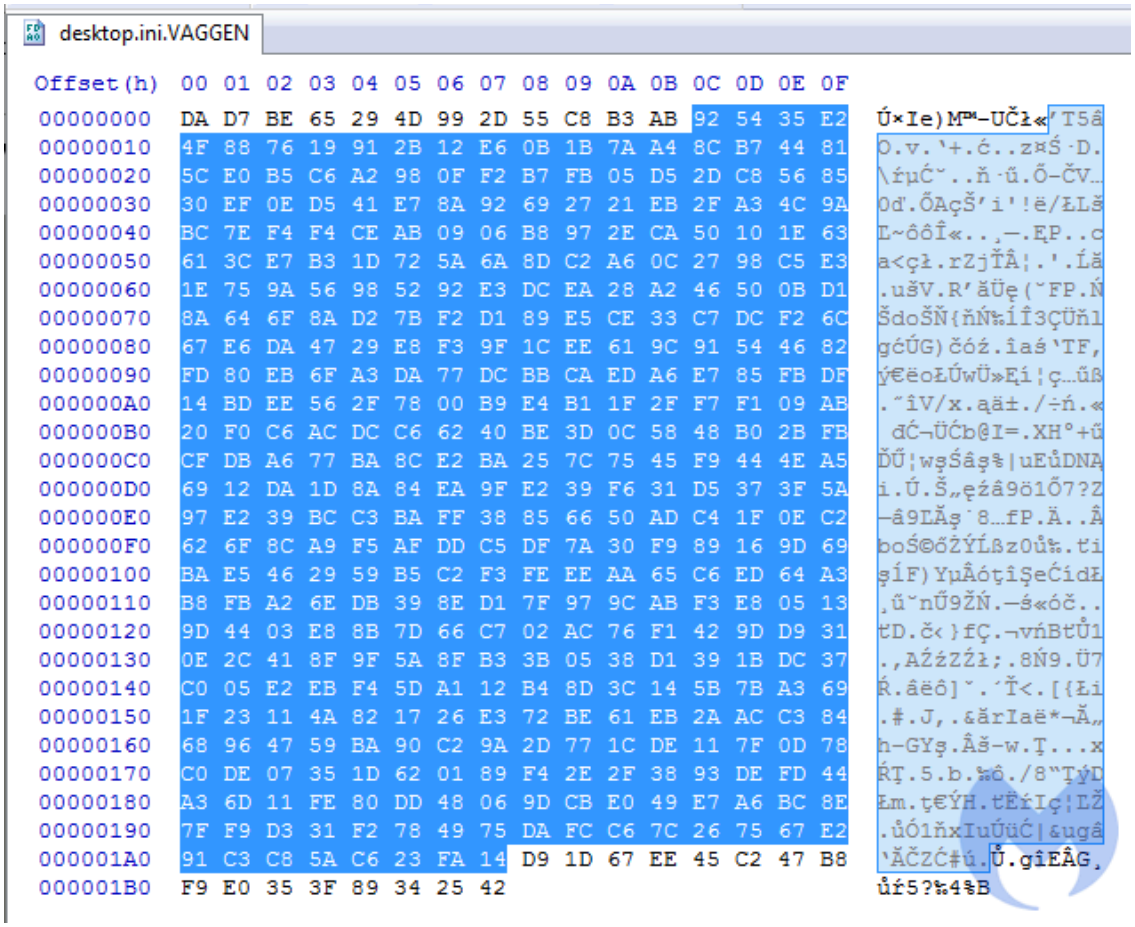
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

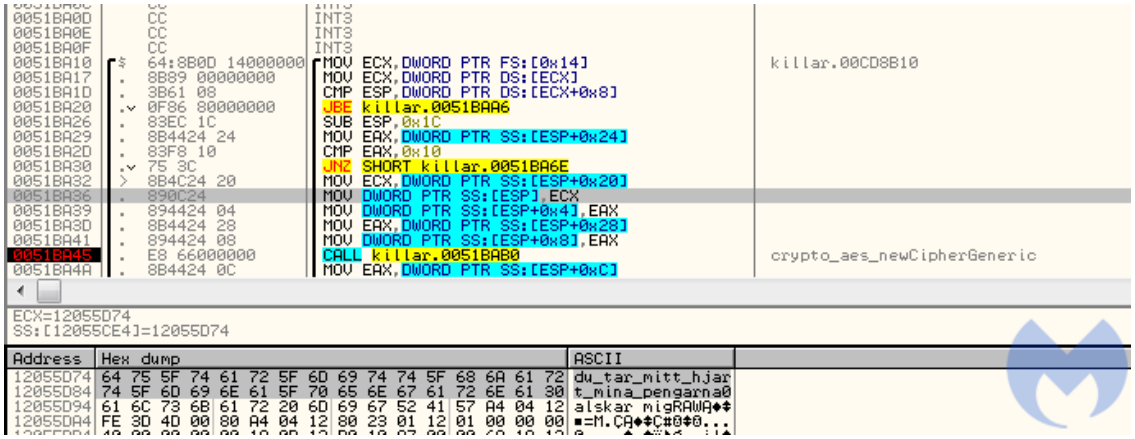


The content of the output file (with .VAGGEN extension) contains:

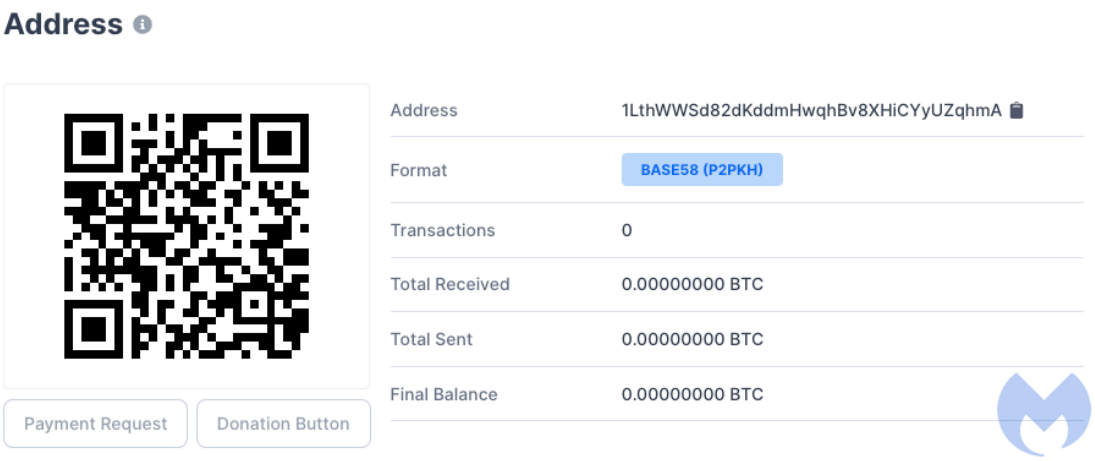
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

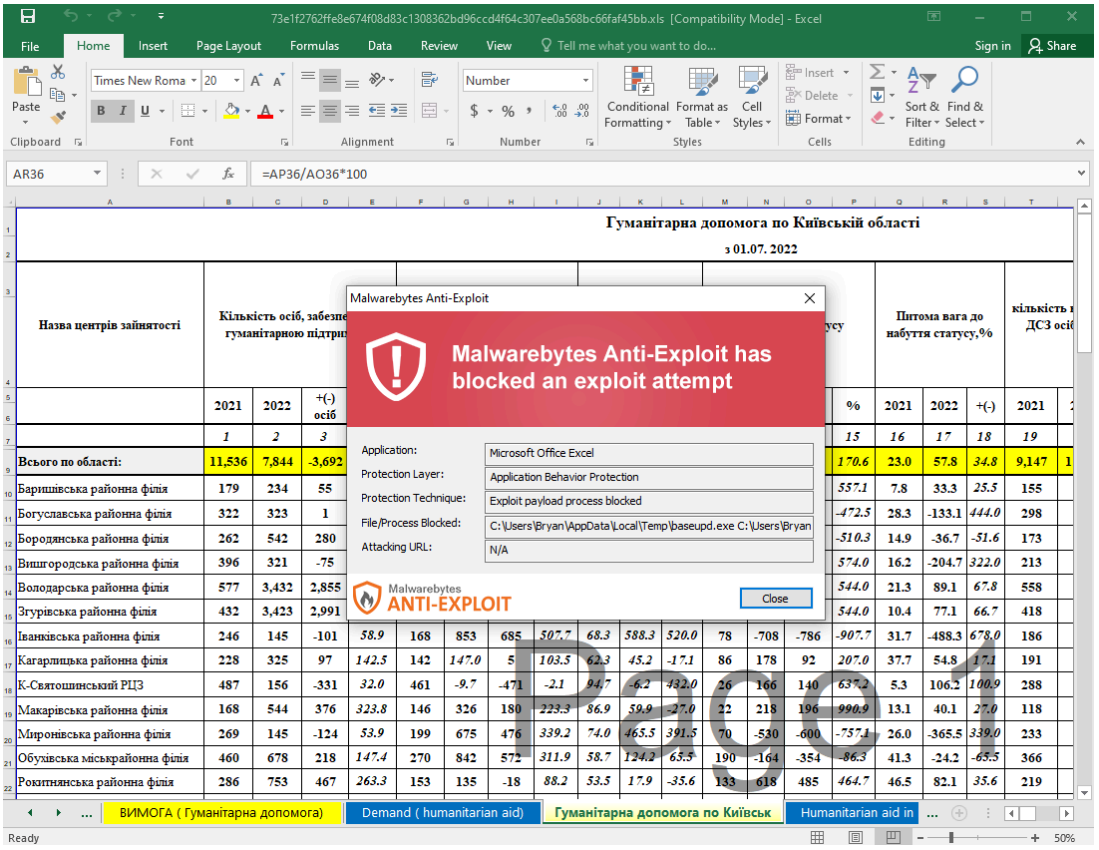
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

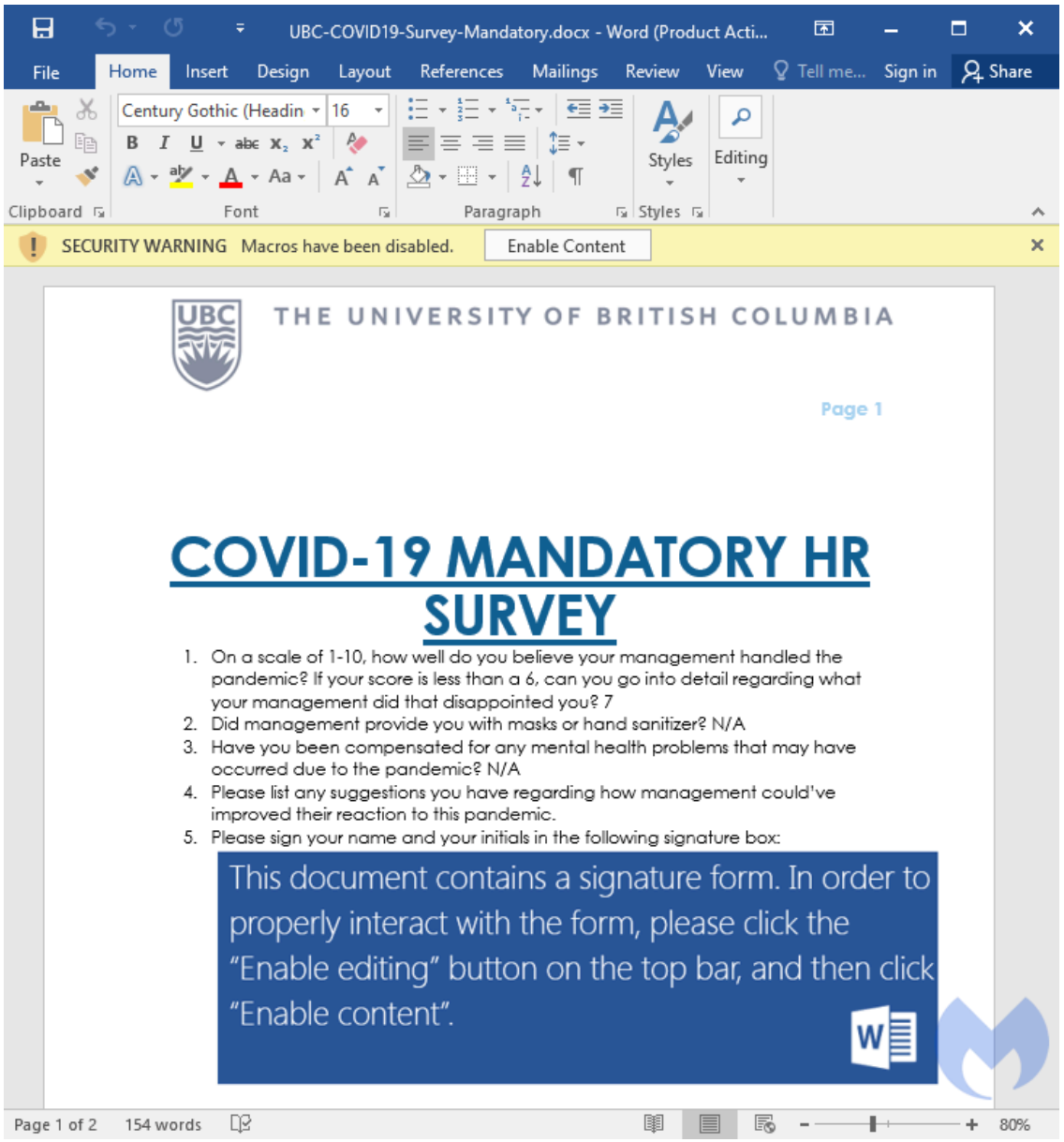
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

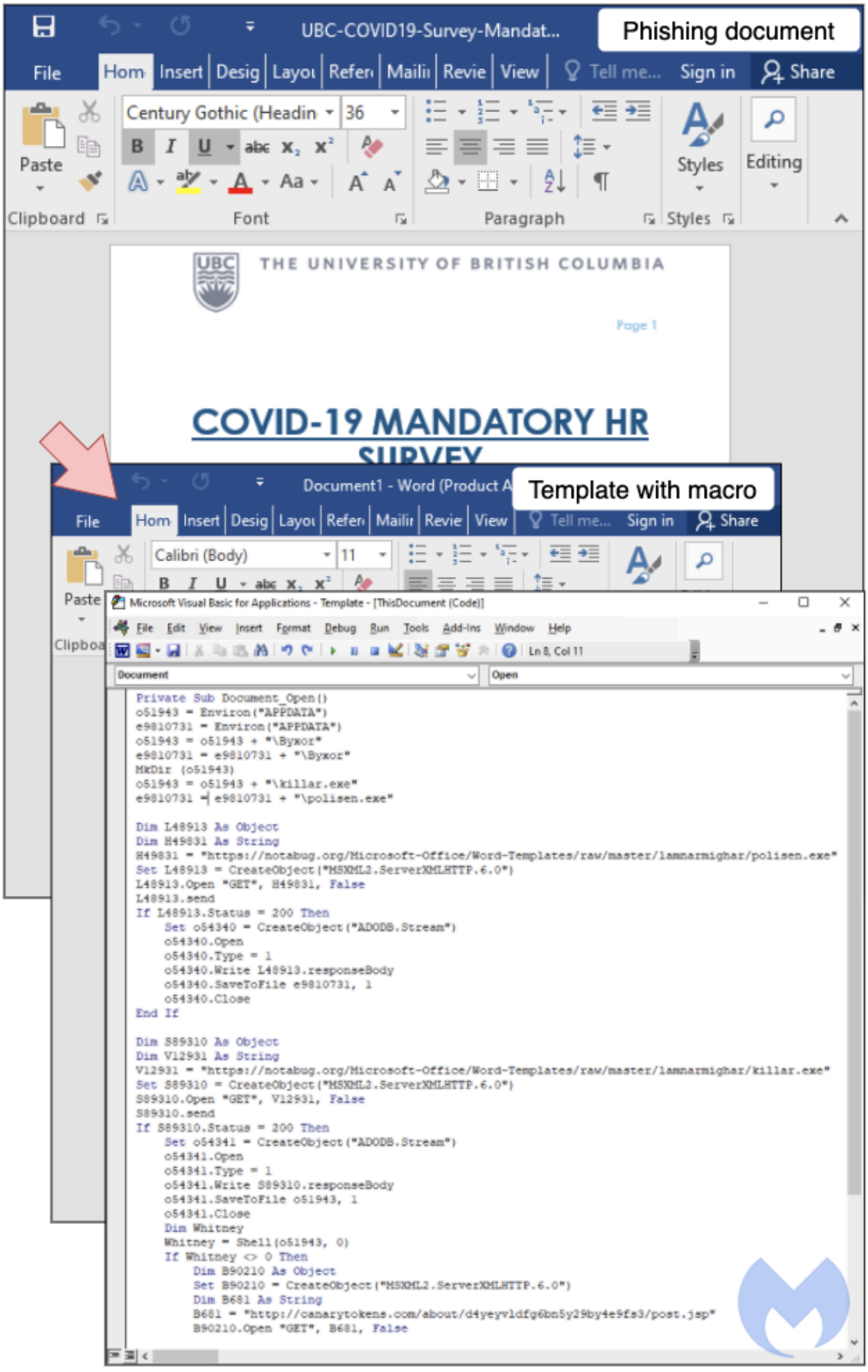
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

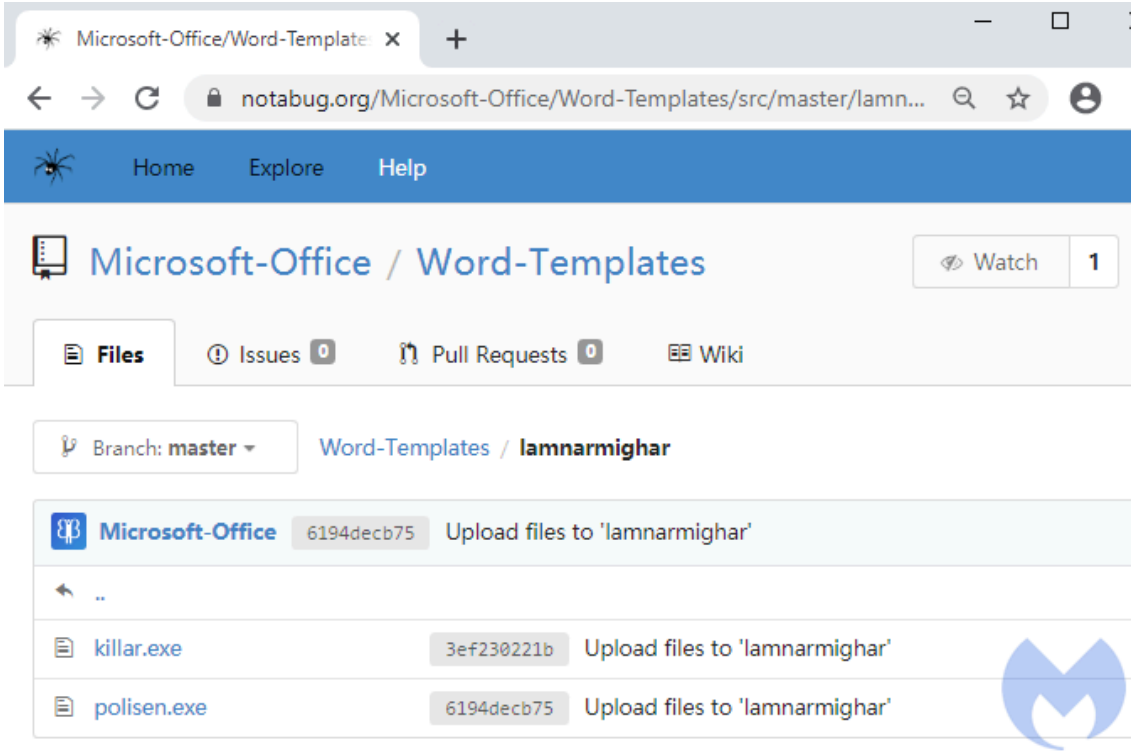
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



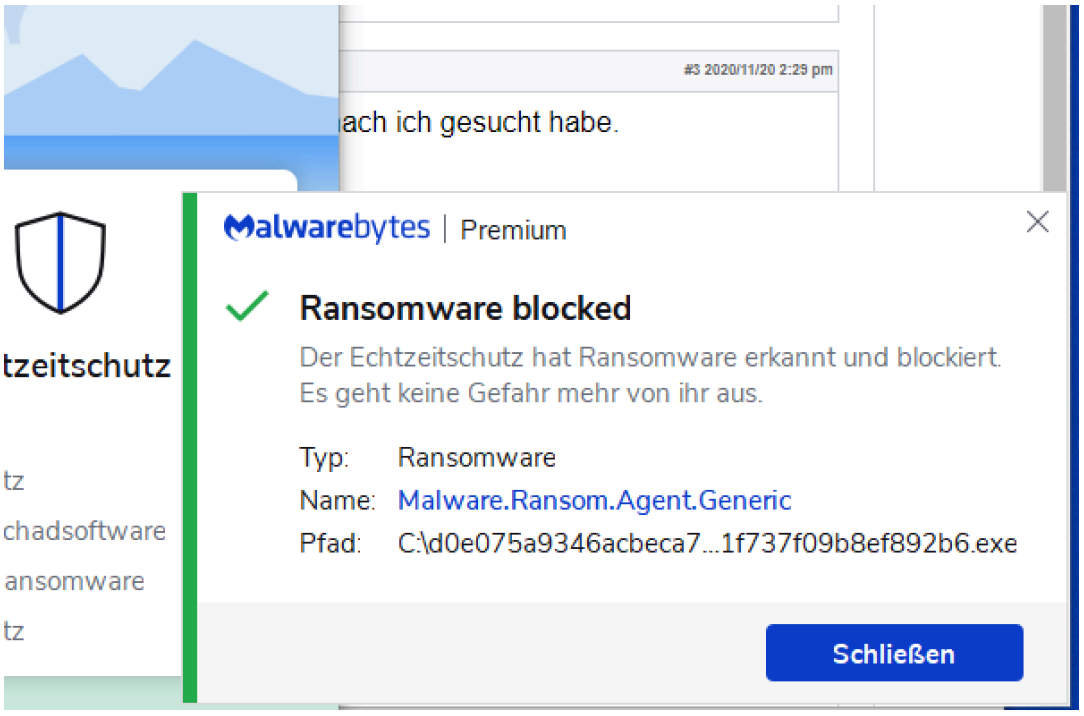
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

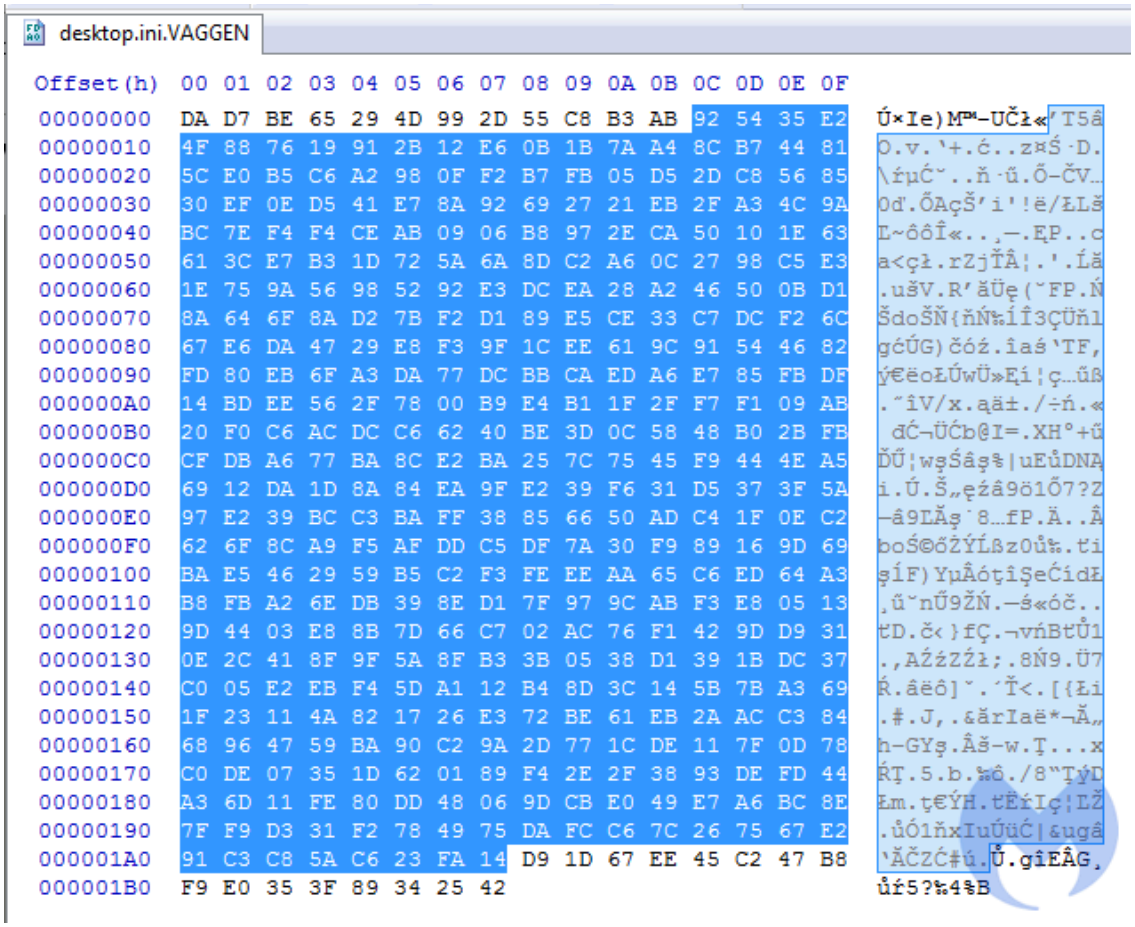
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



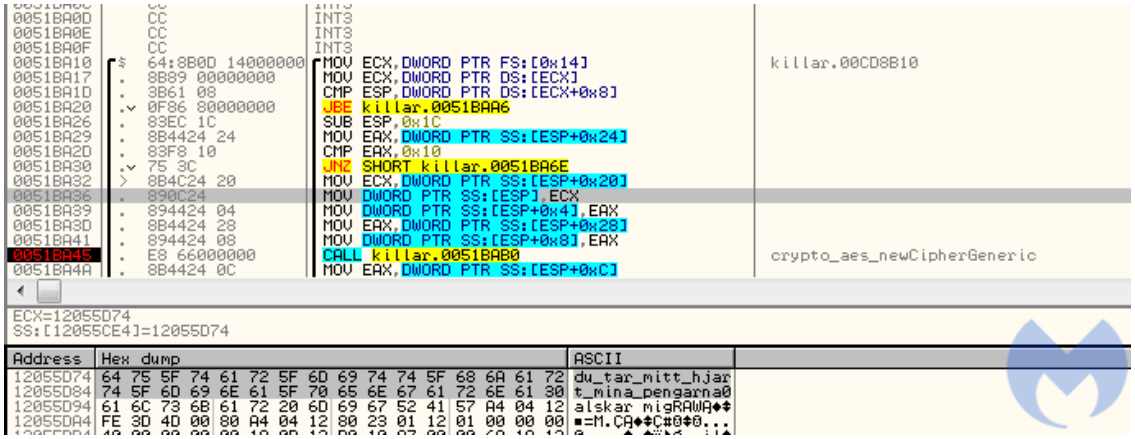
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

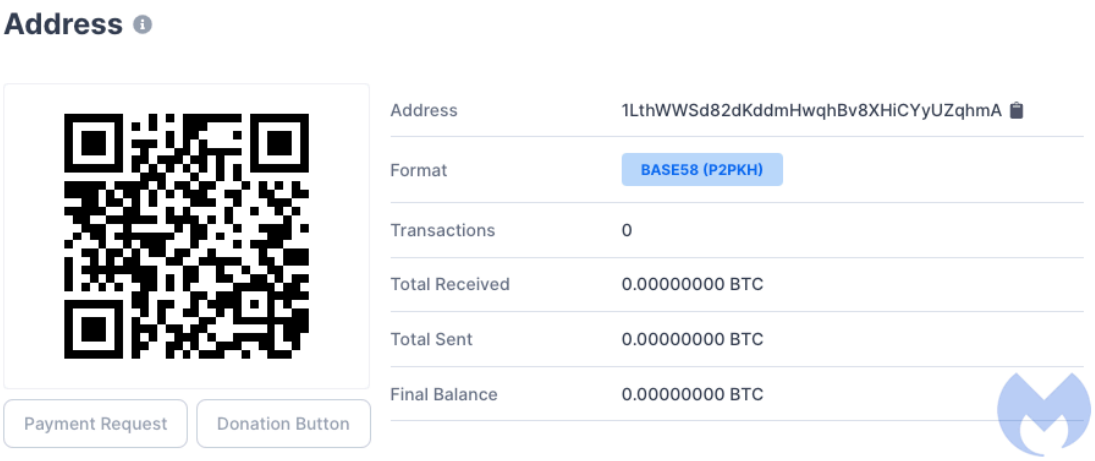
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

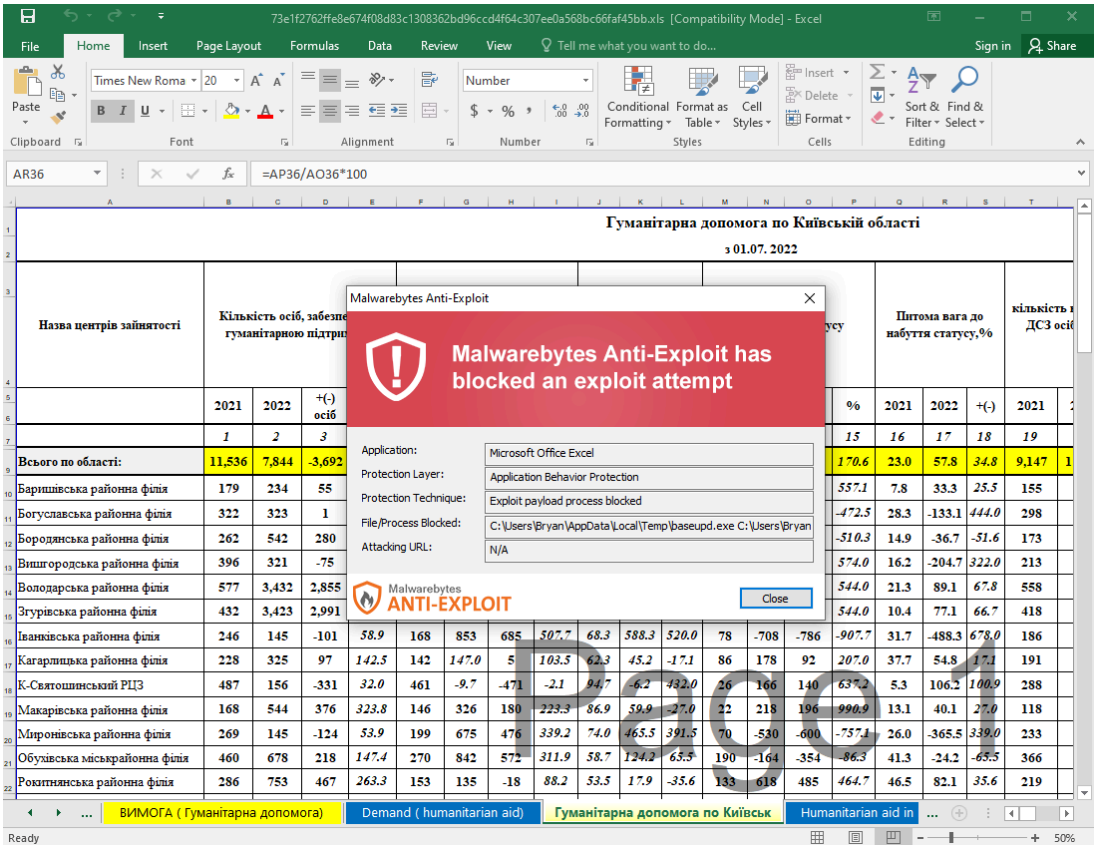
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

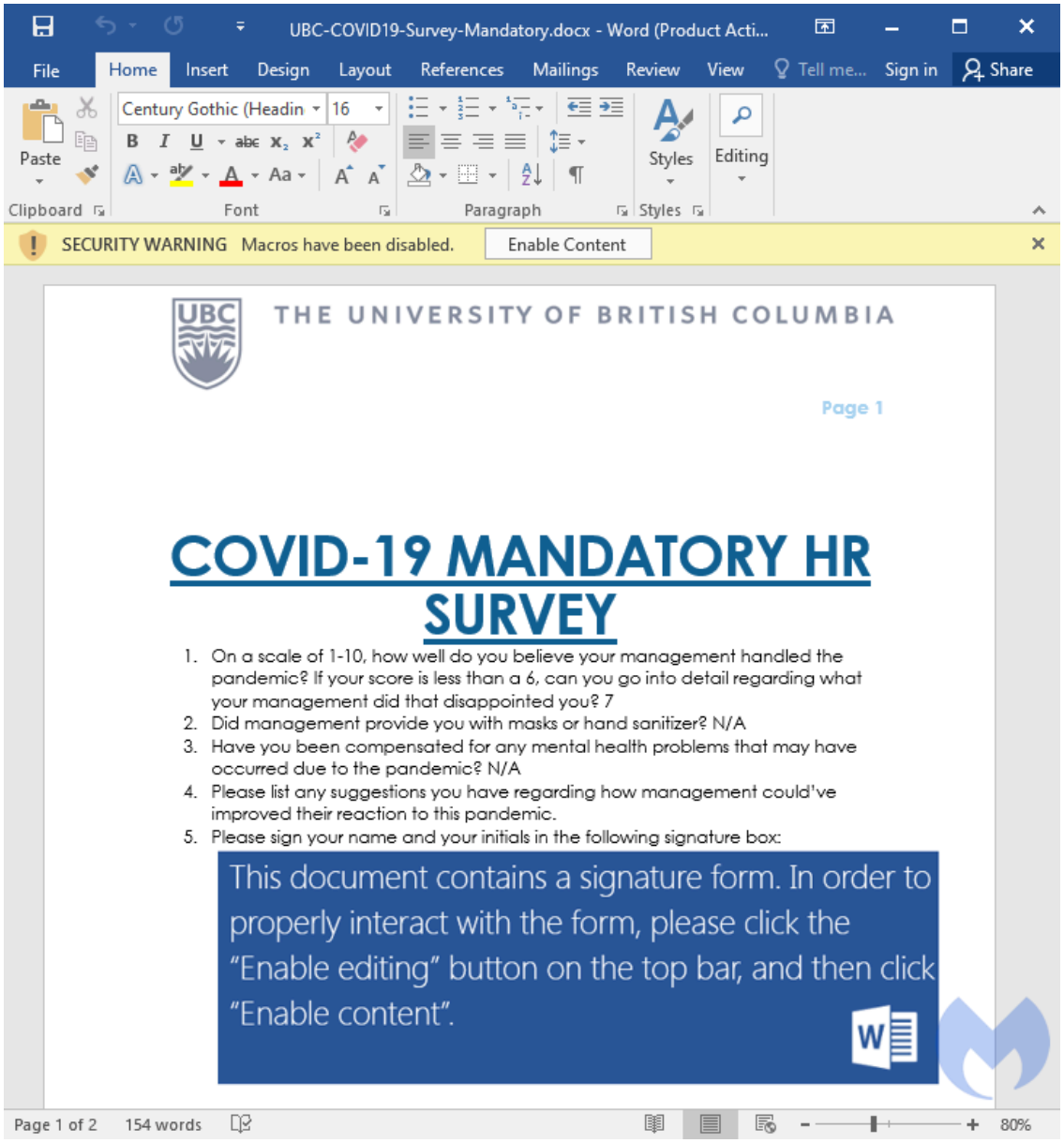
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

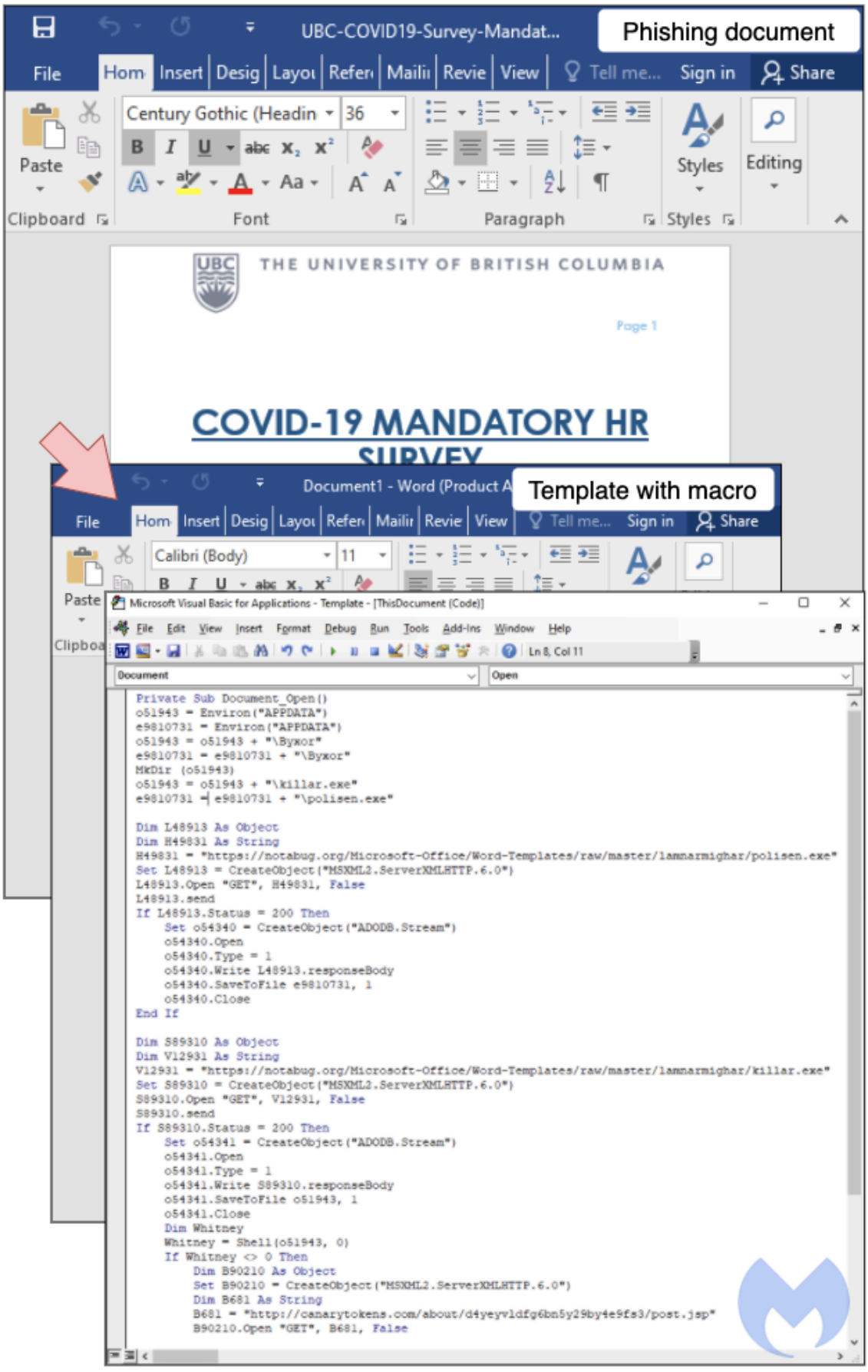
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

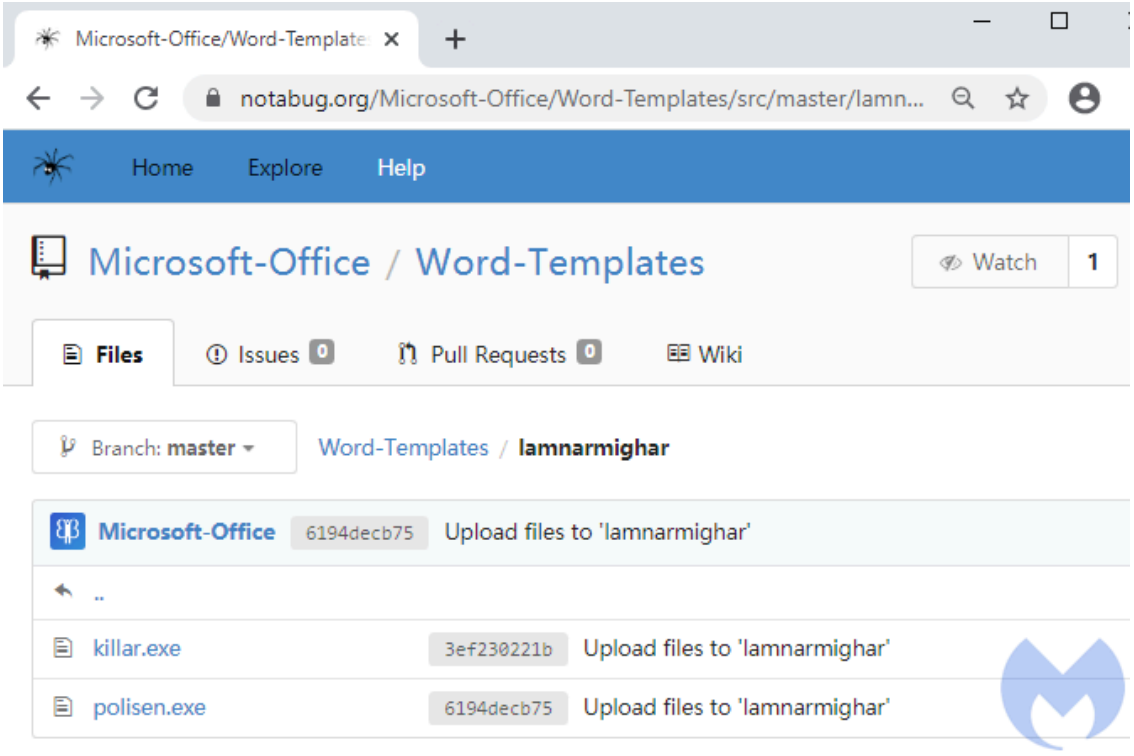
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



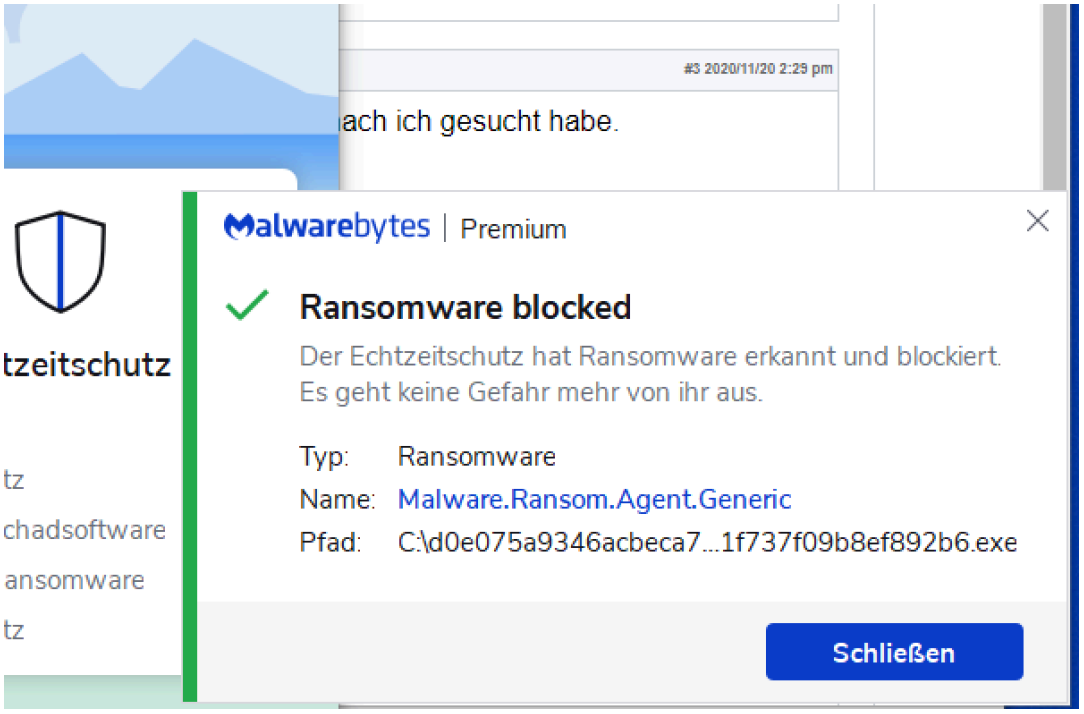
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

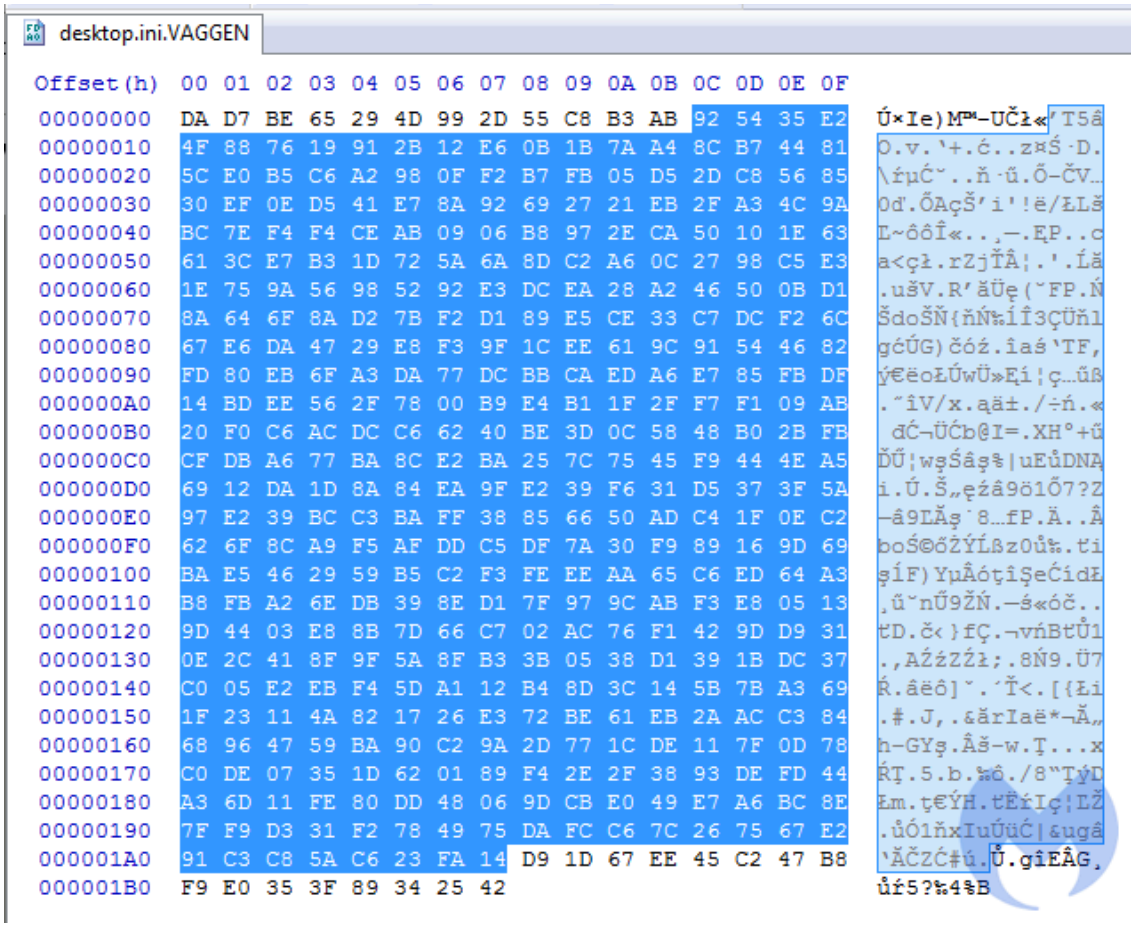
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



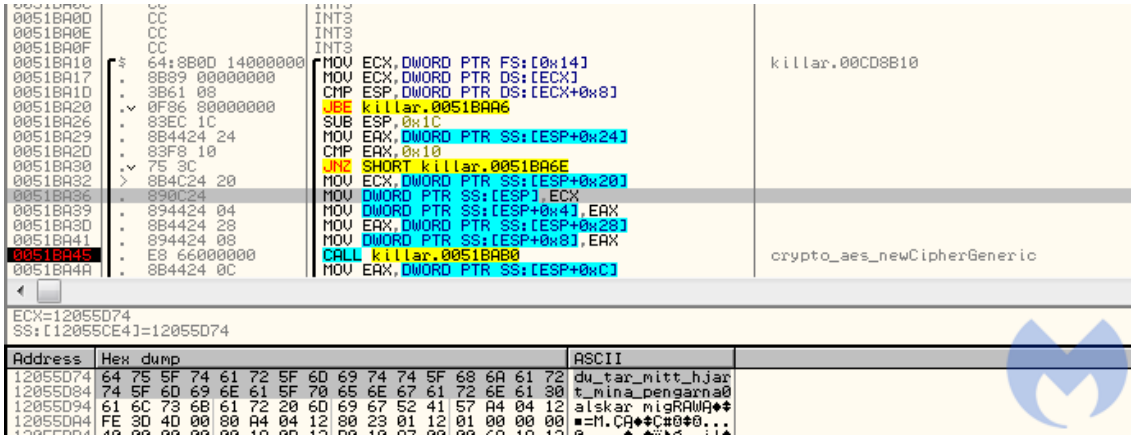
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

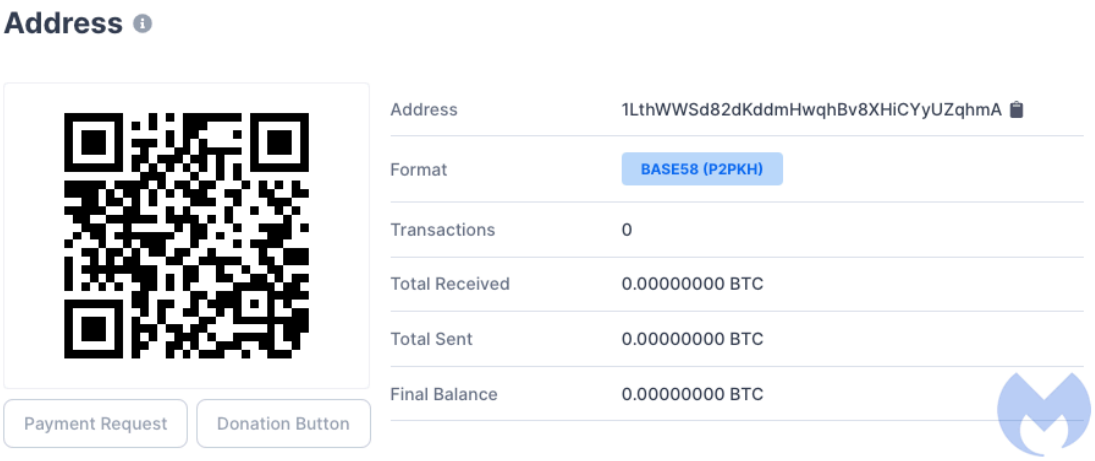
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

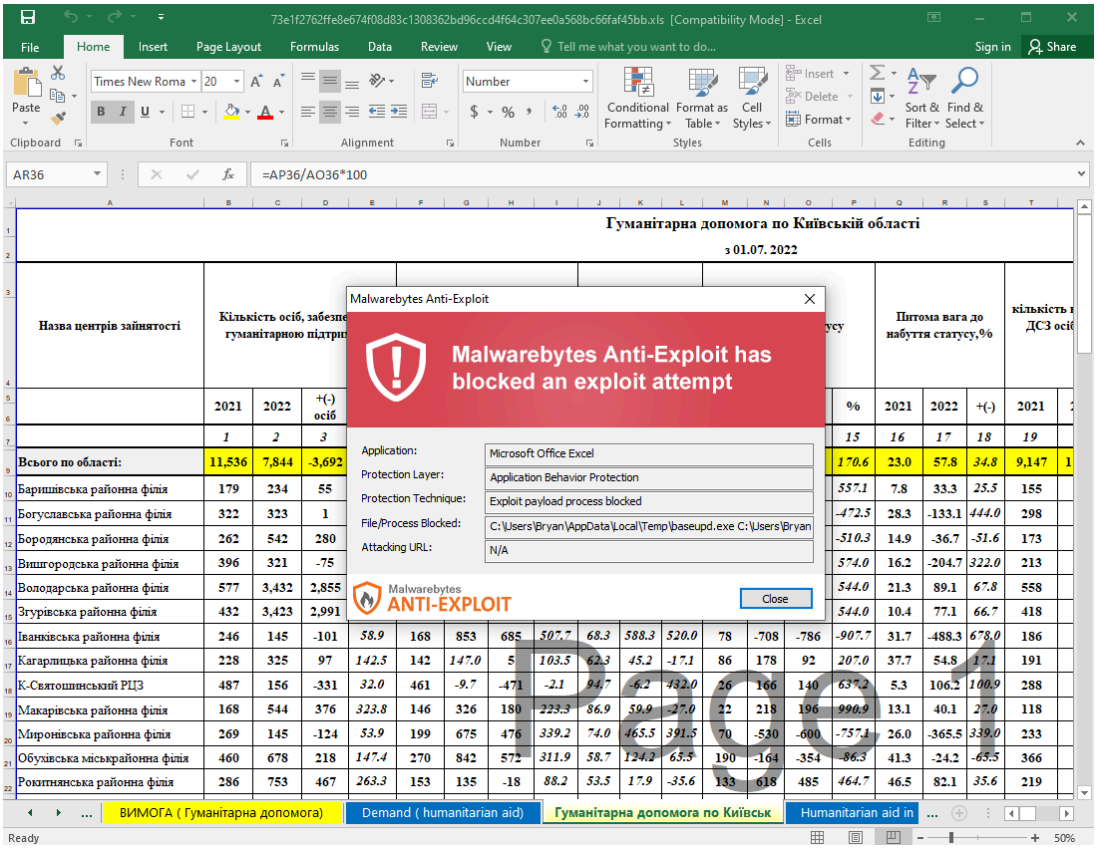
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

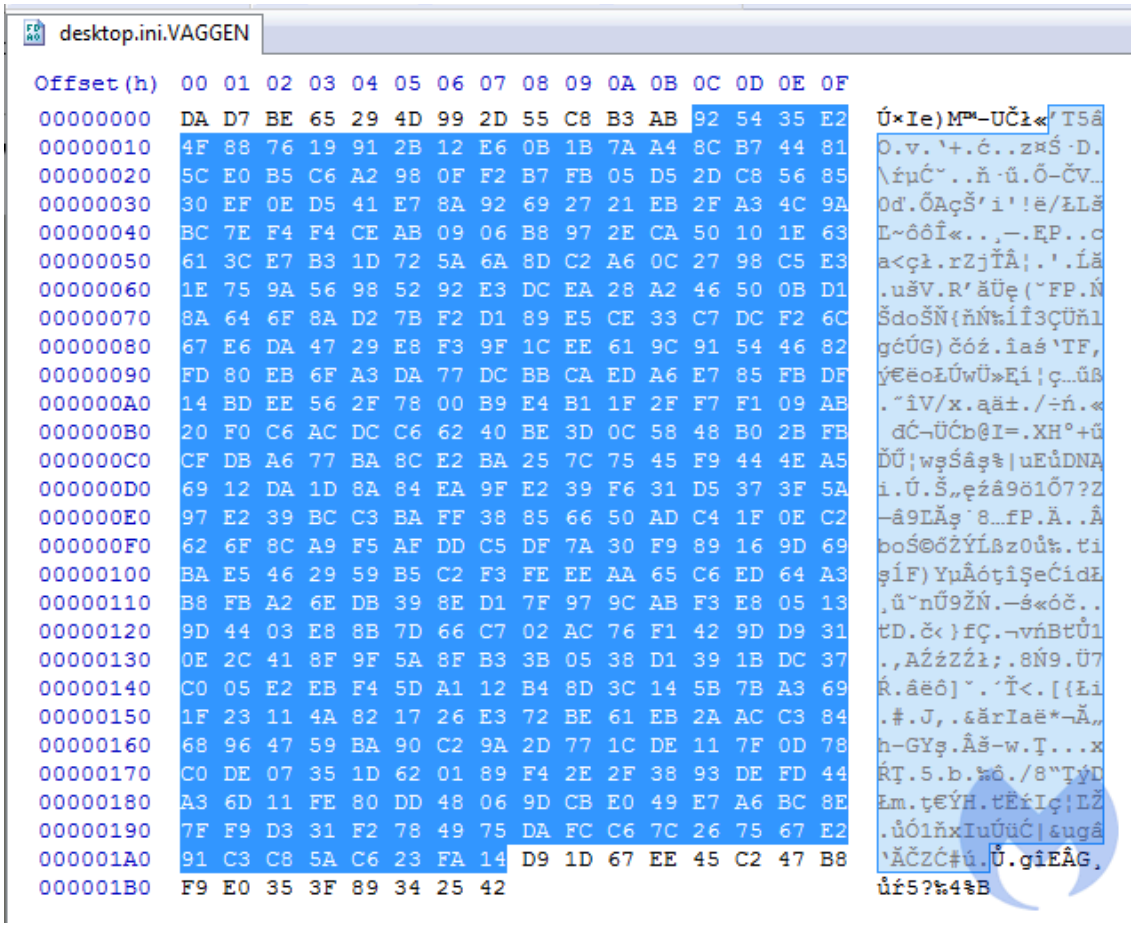
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

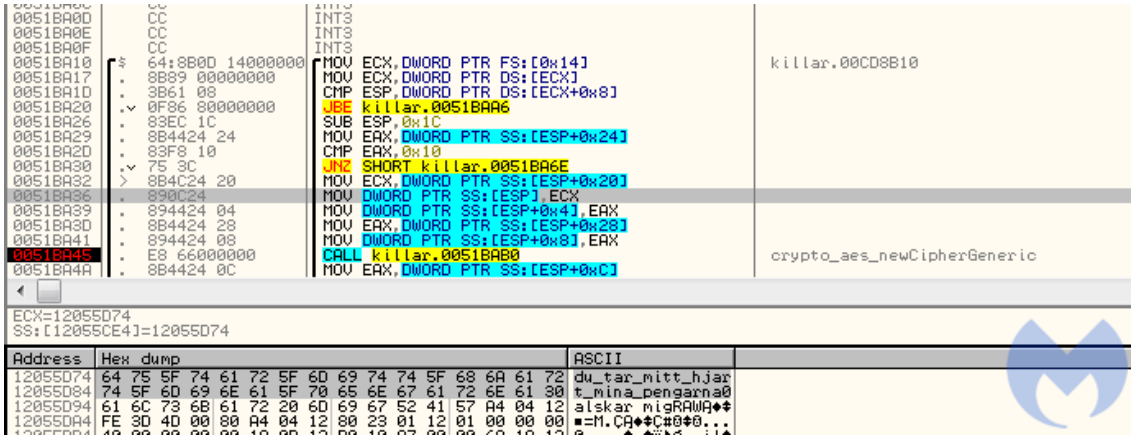


The content of the output file (with .VAGGEN extension) contains:

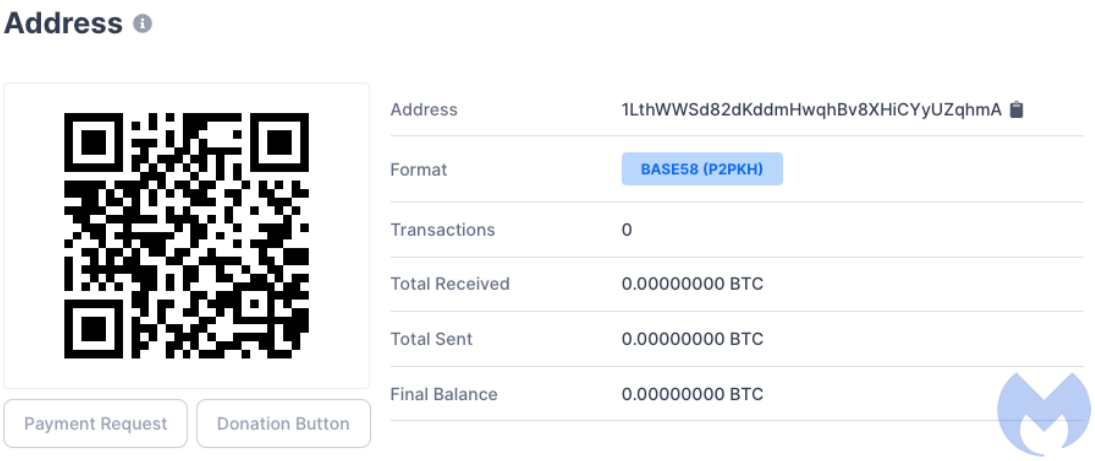
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

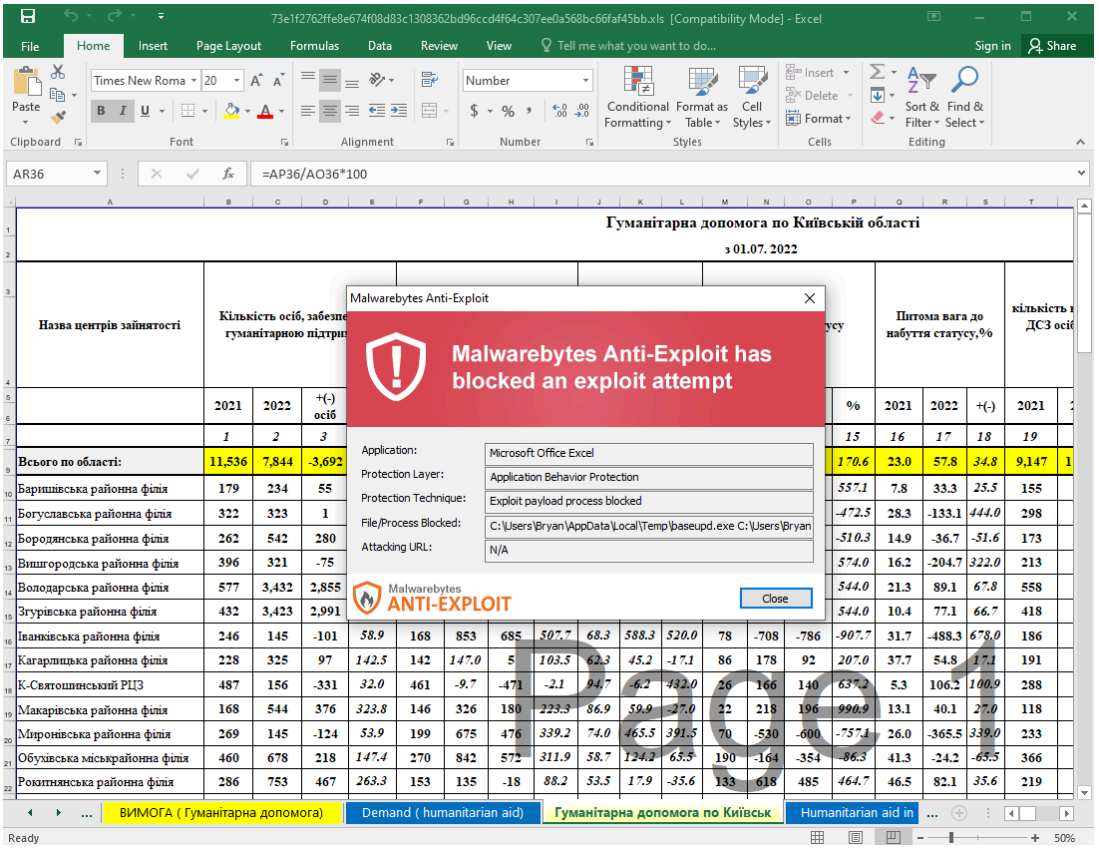
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

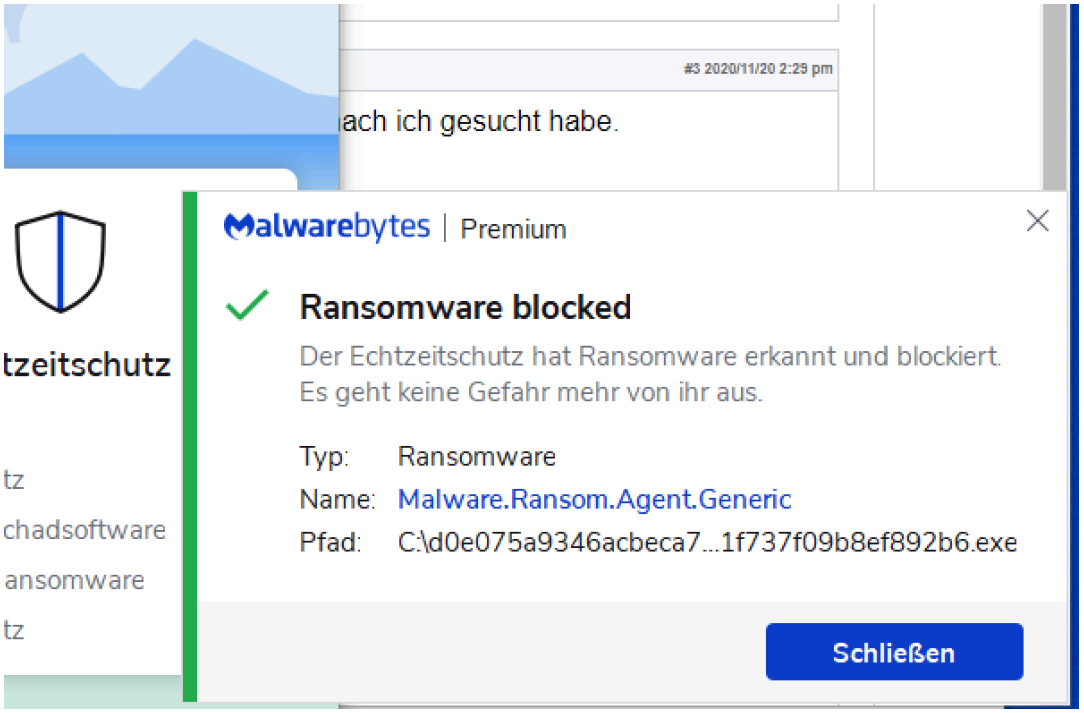
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

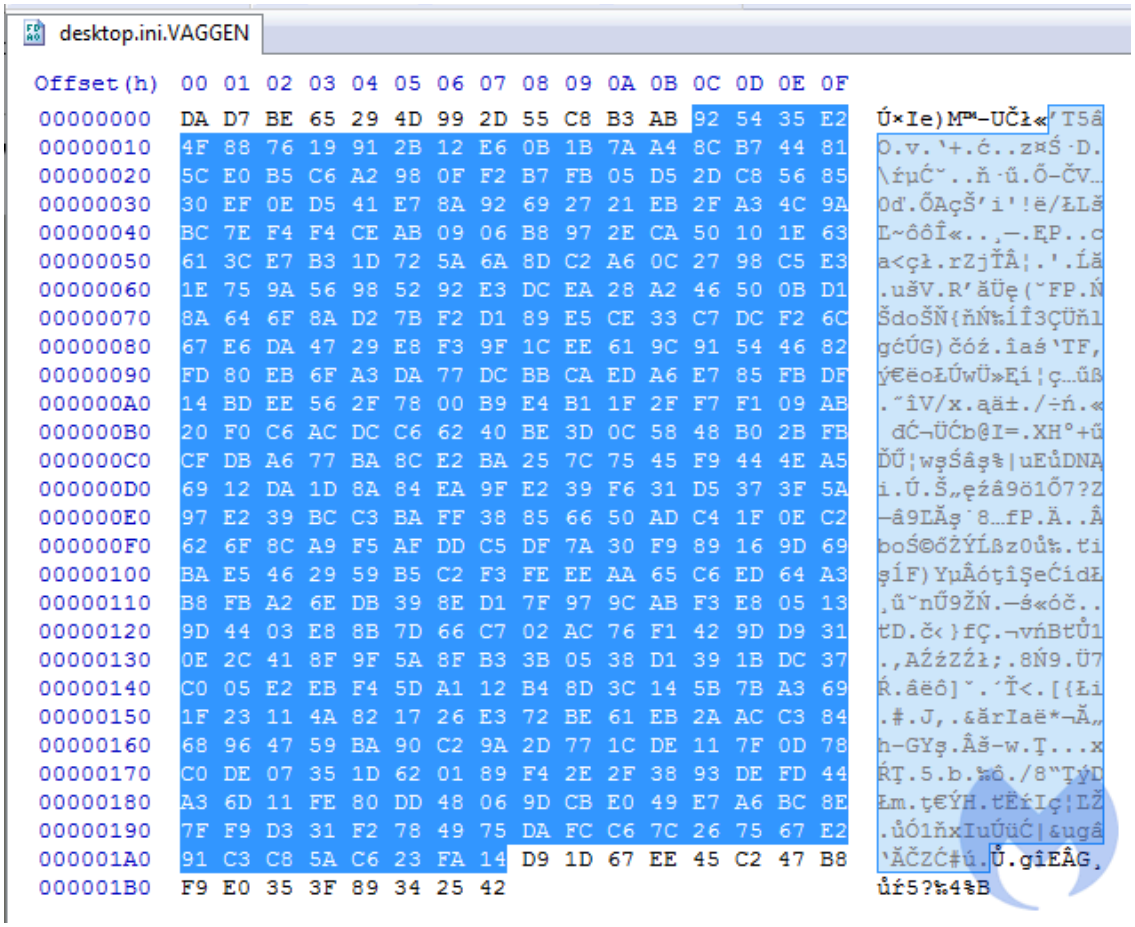
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

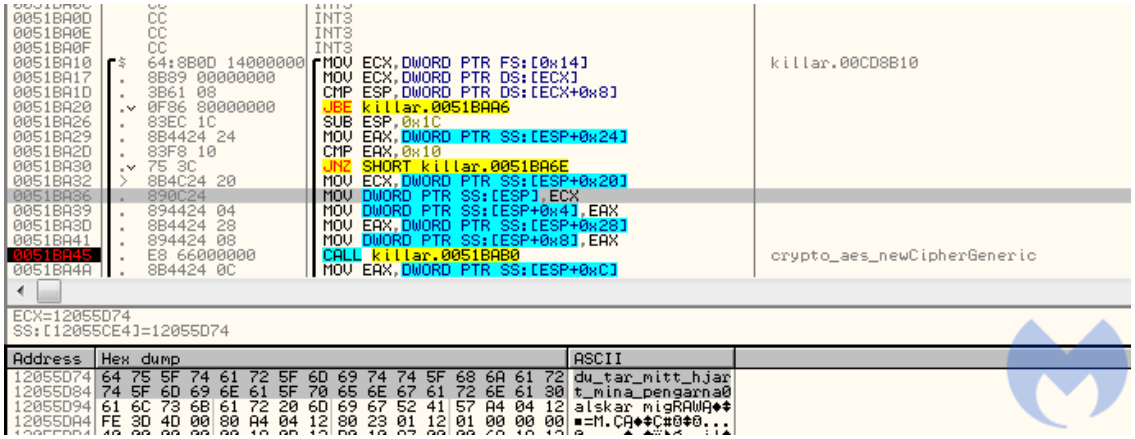


The content of the output file (with .VAGGEN extension) contains:

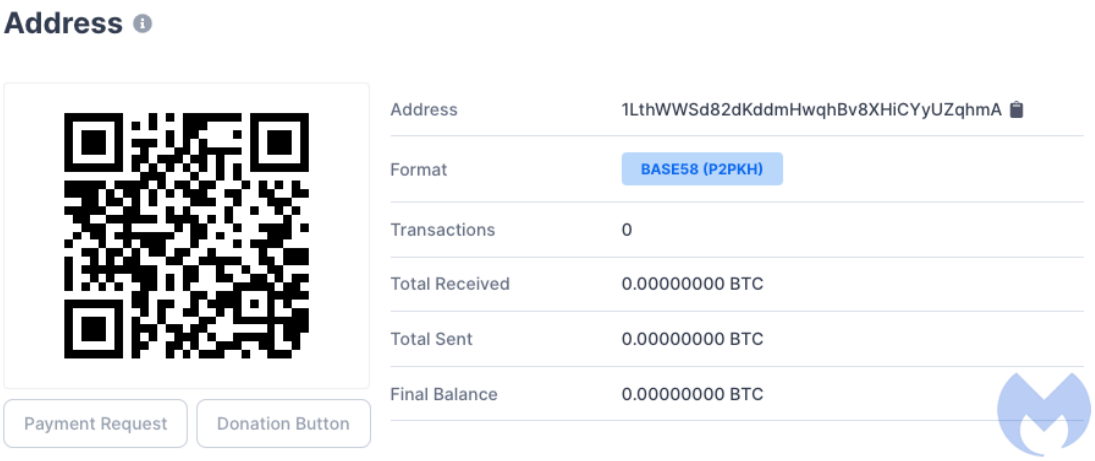
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

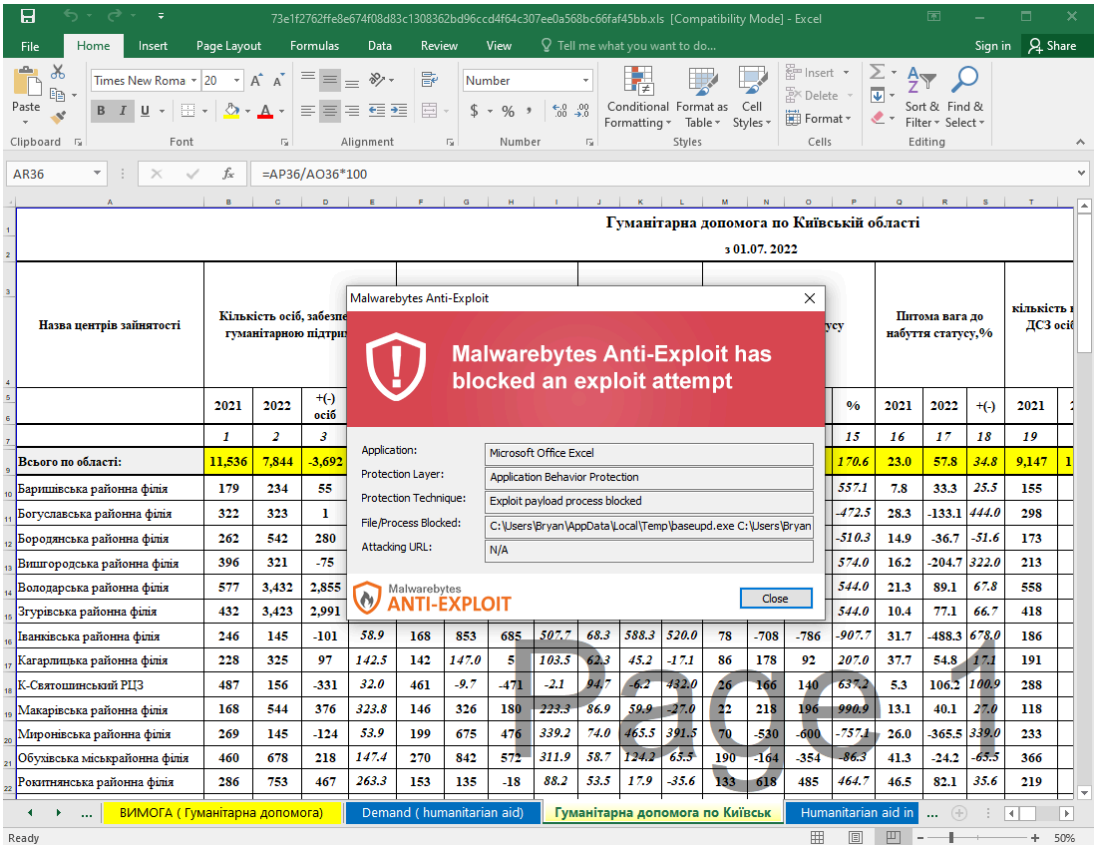
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

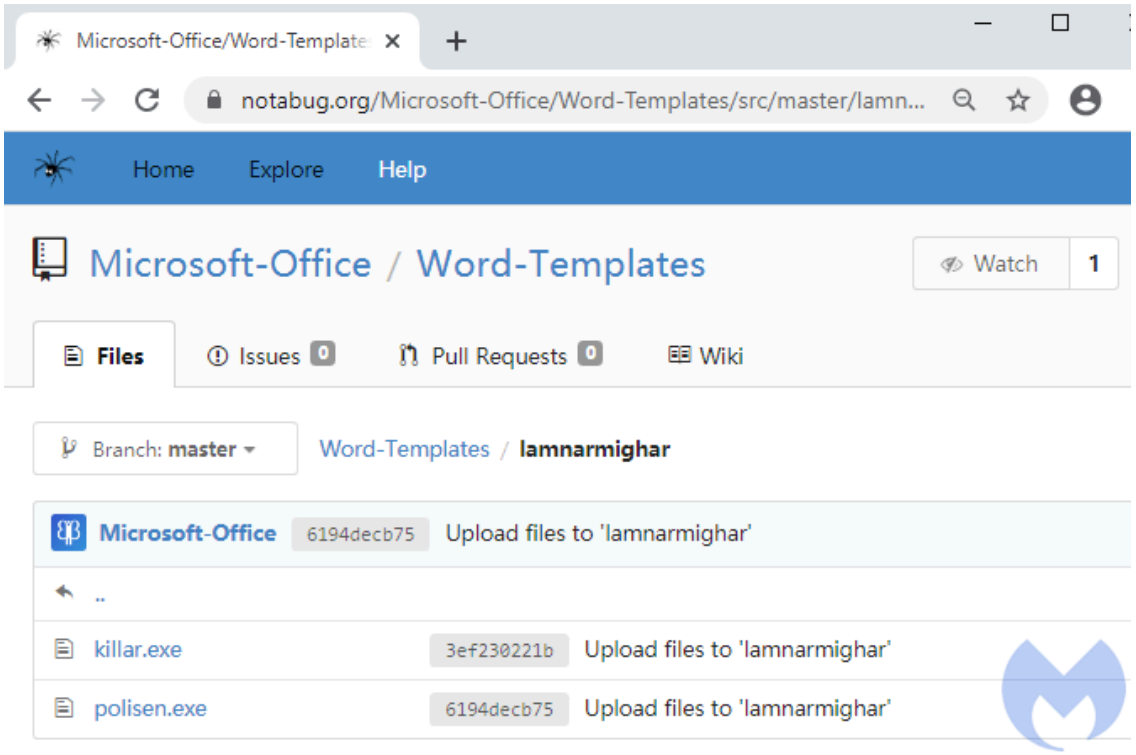
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



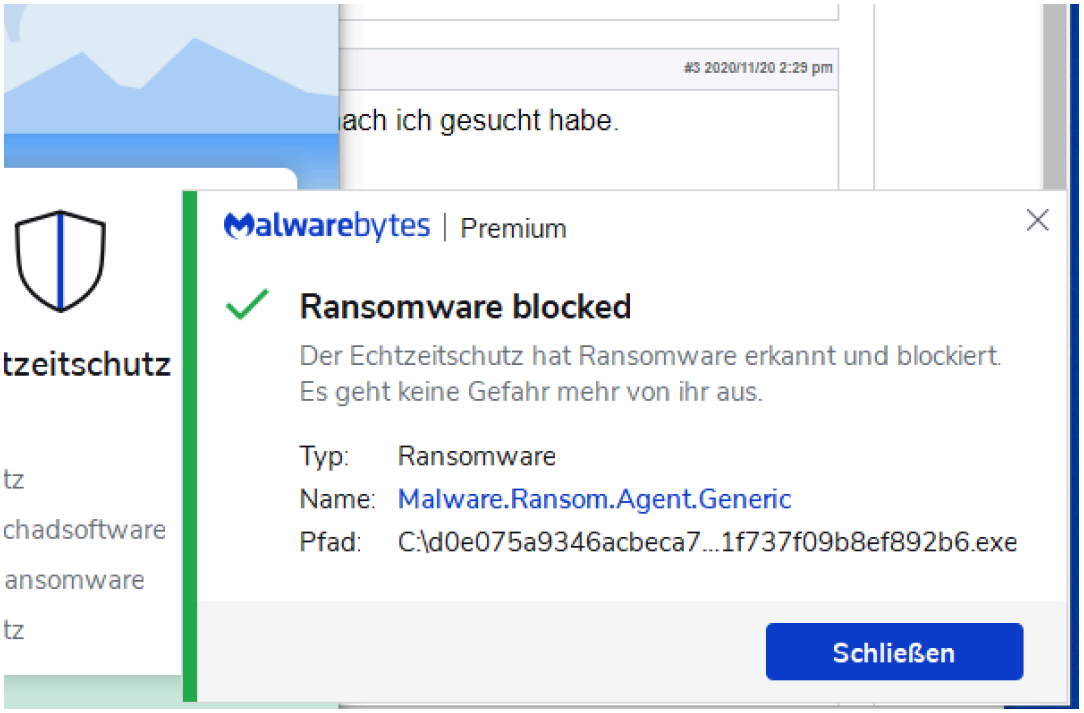
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

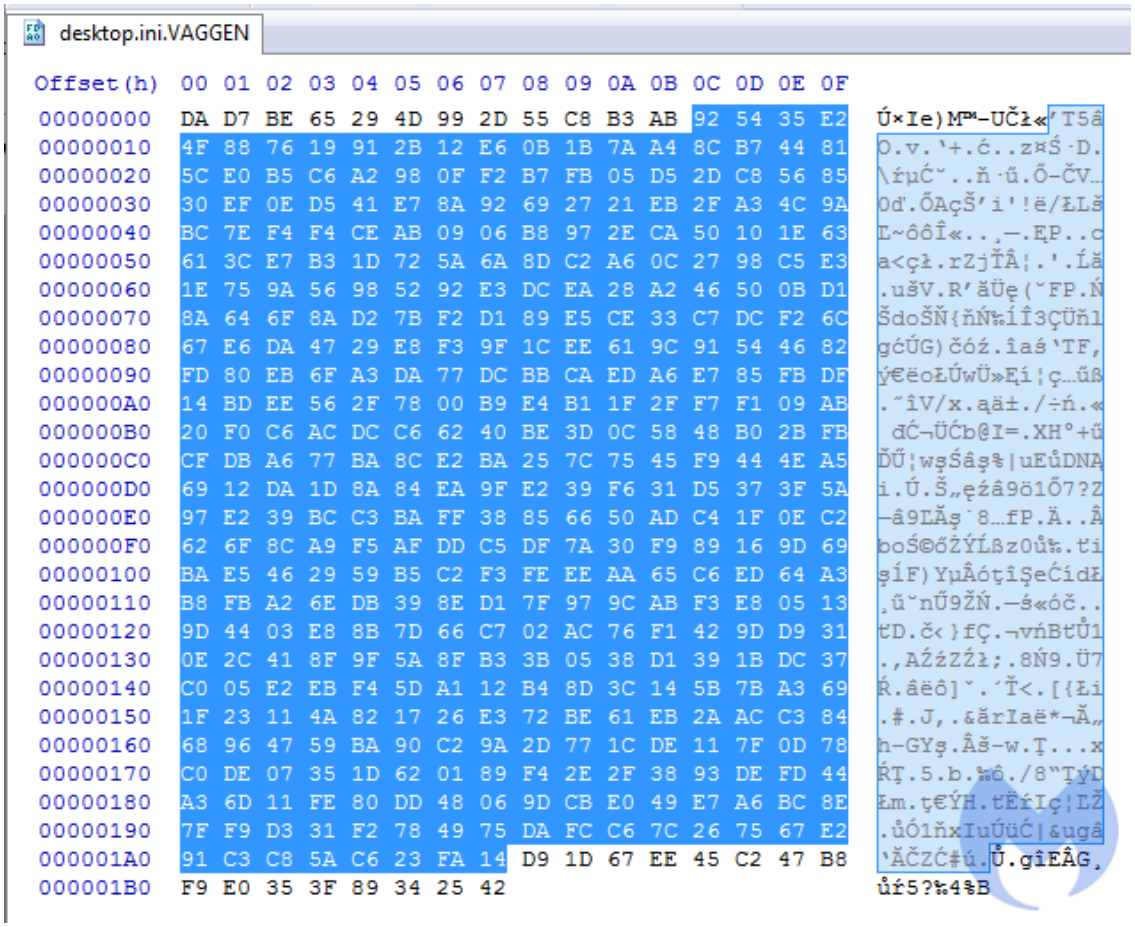
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

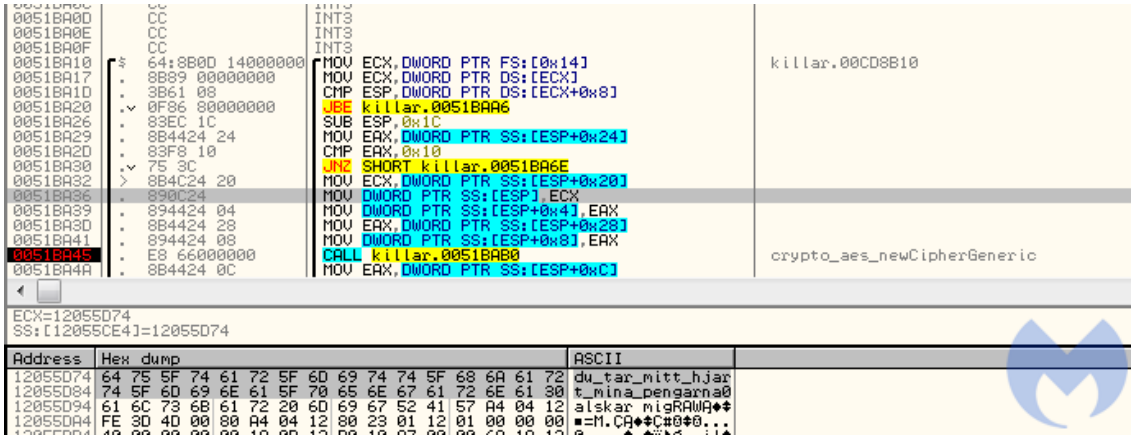


The content of the output file (with .VAGGEN extension) contains:

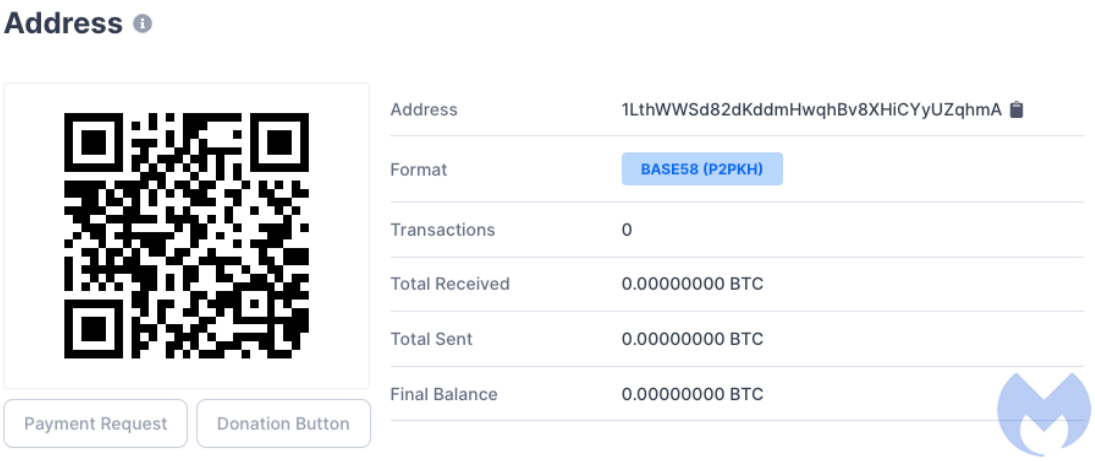
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

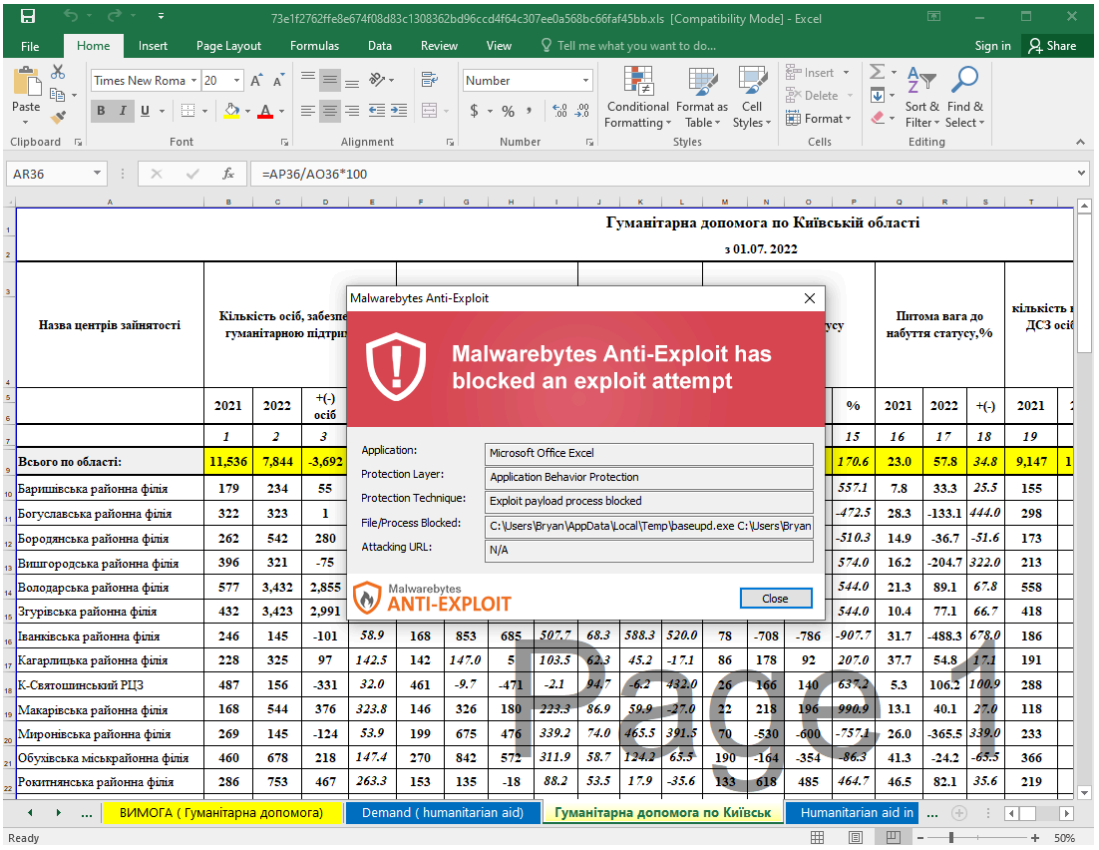
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

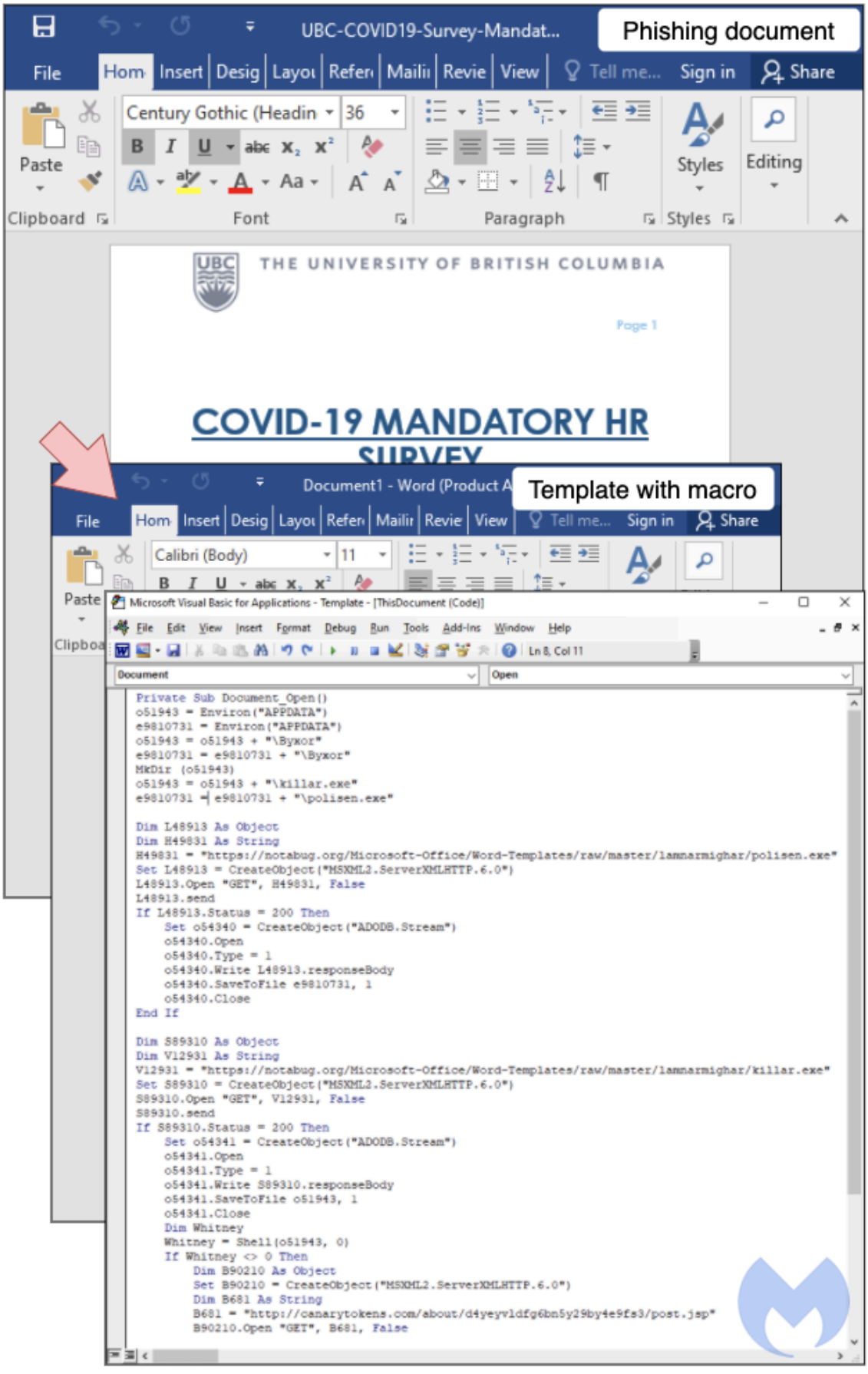
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

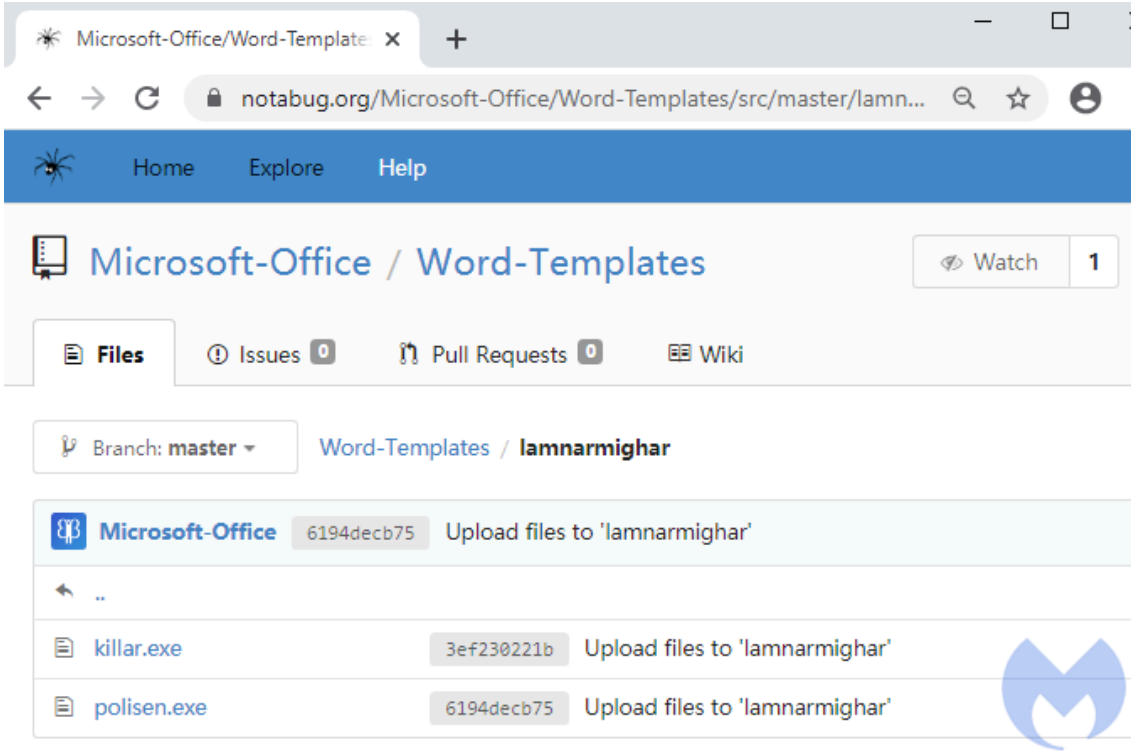
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



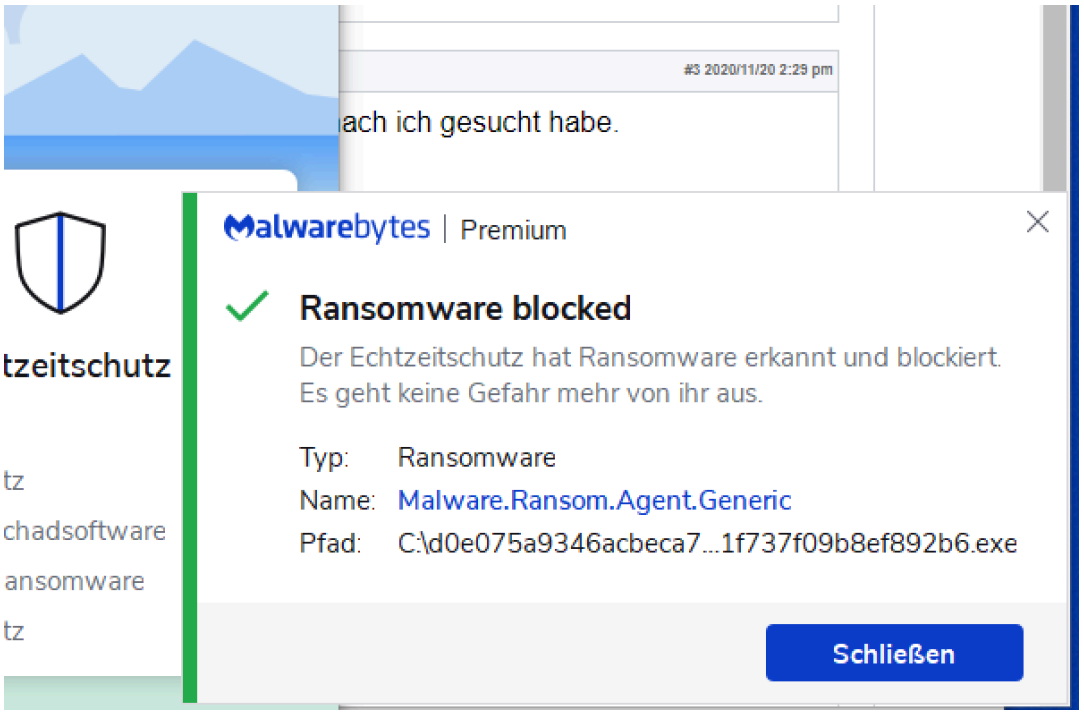
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

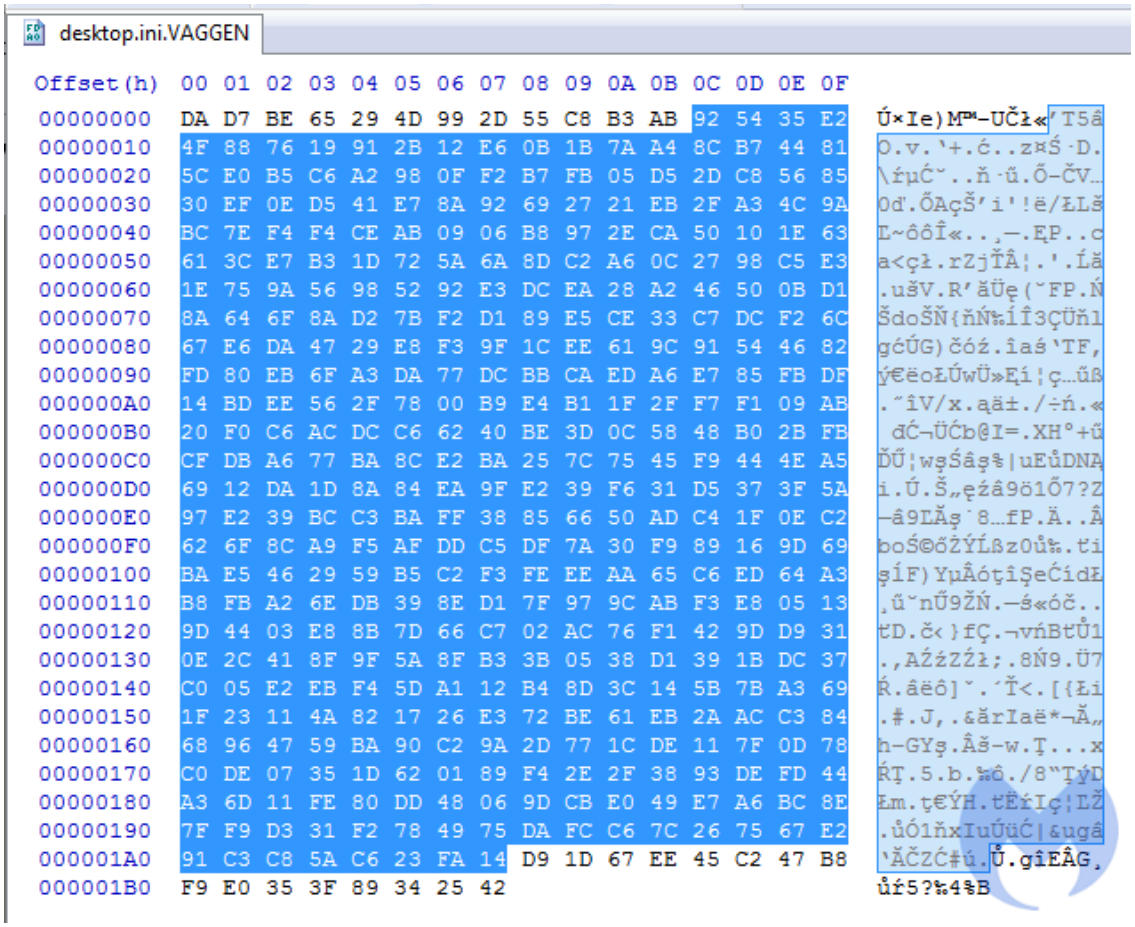
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



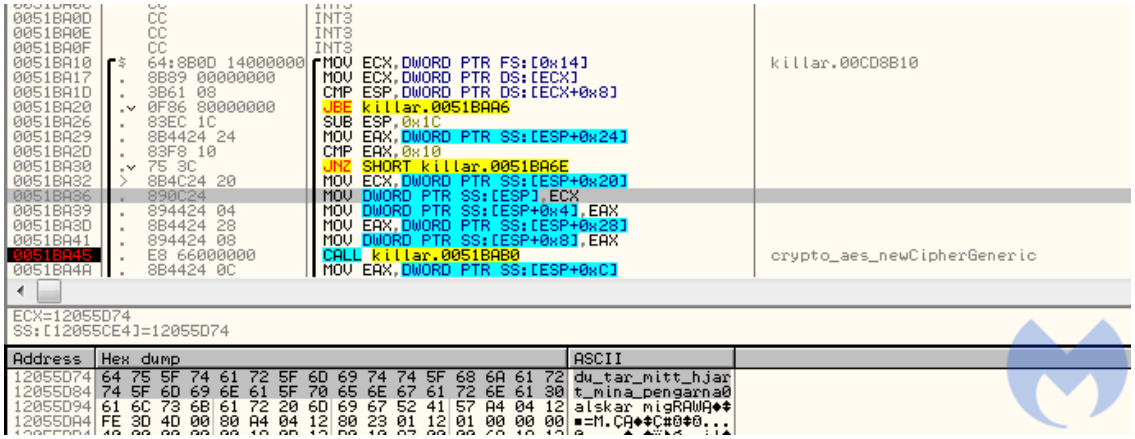
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

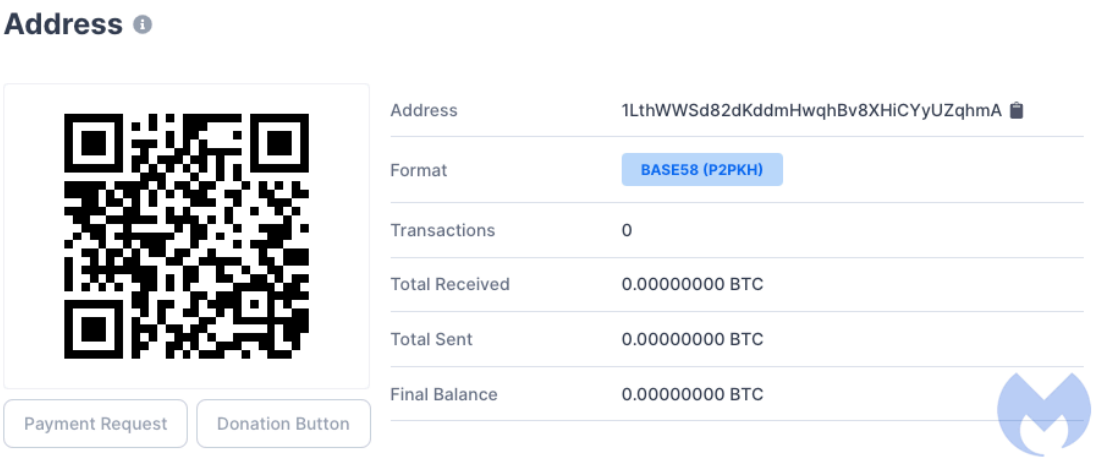
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

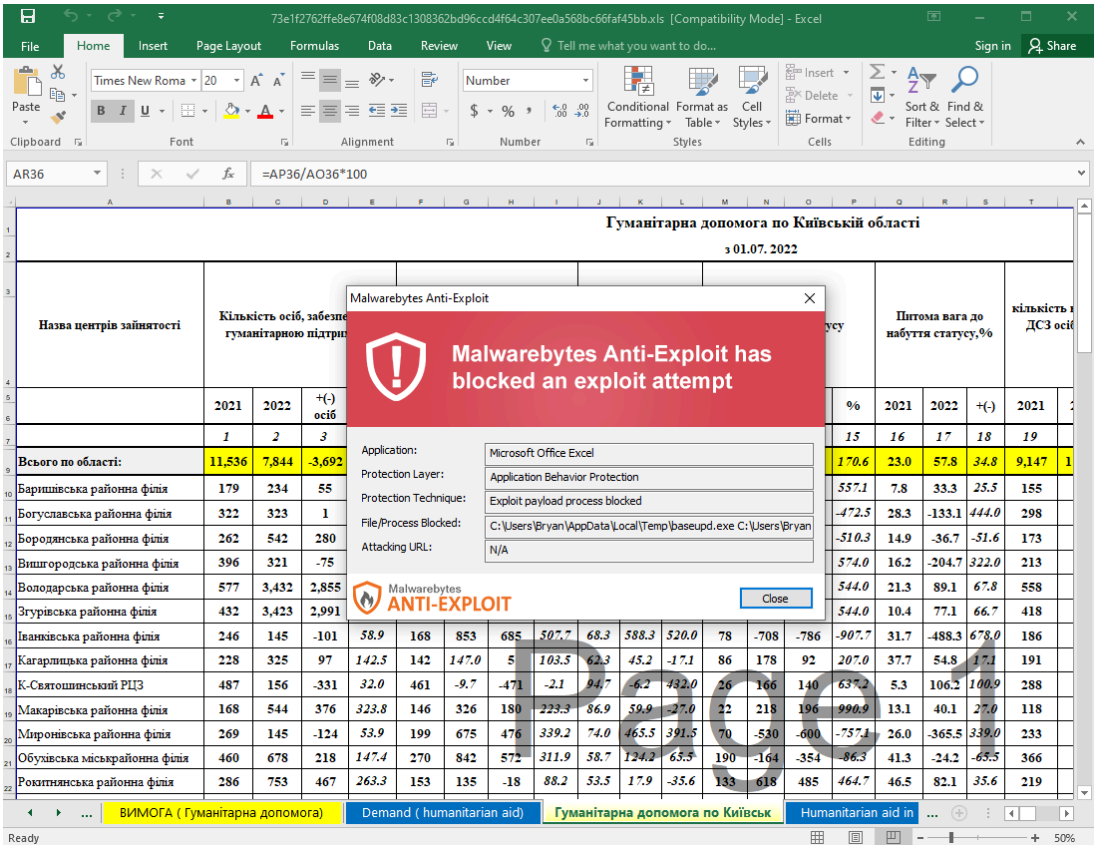
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

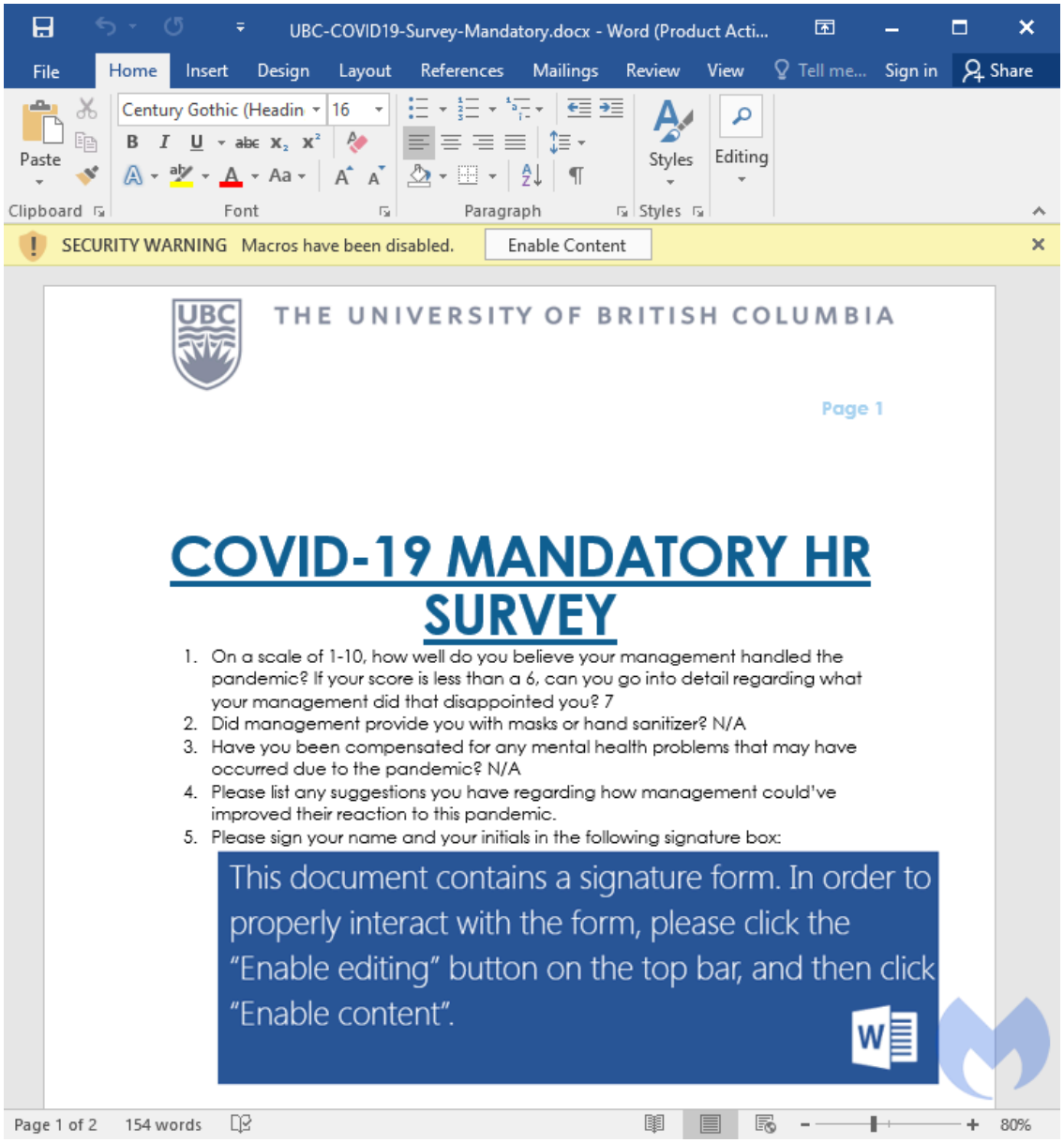
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

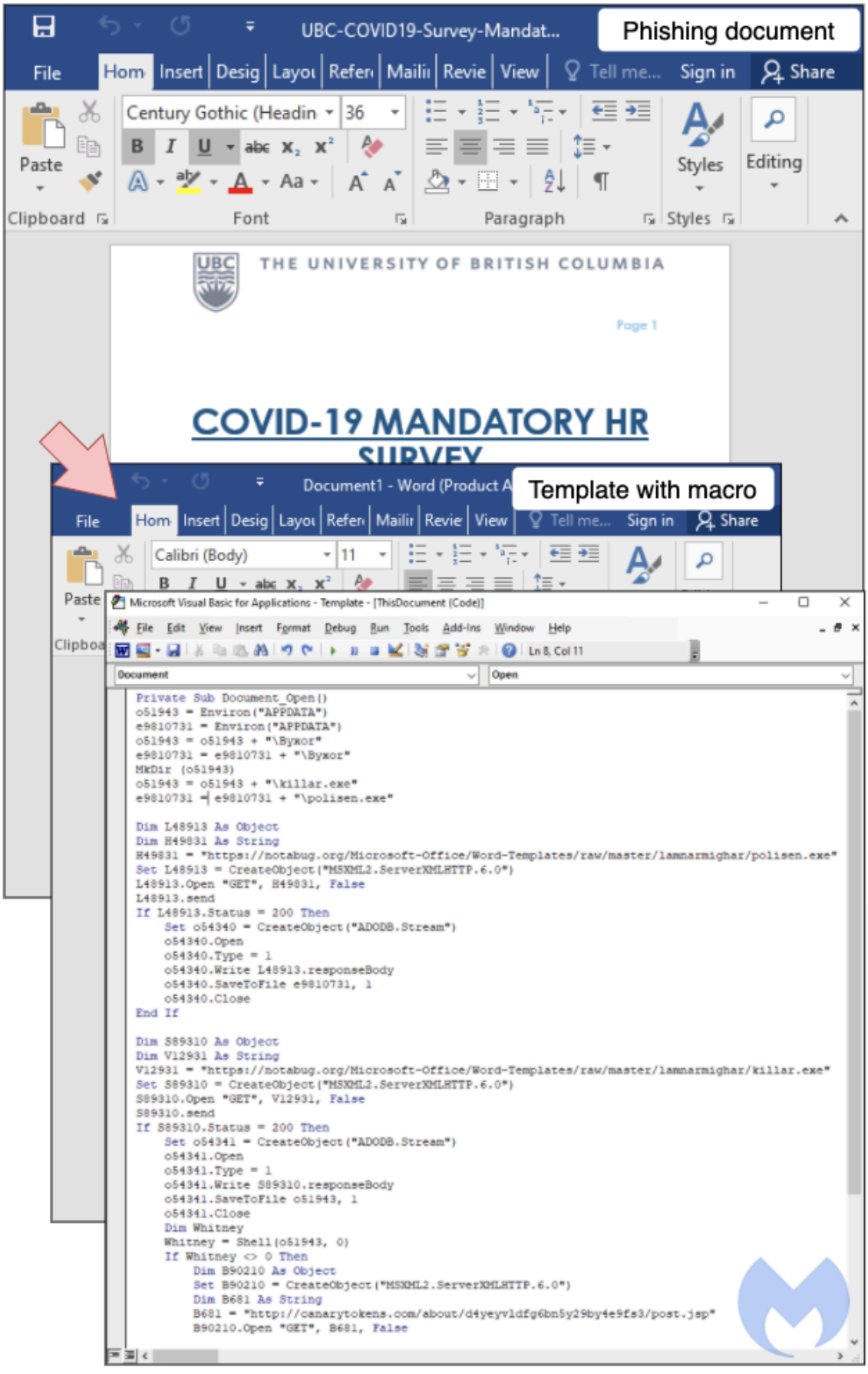
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

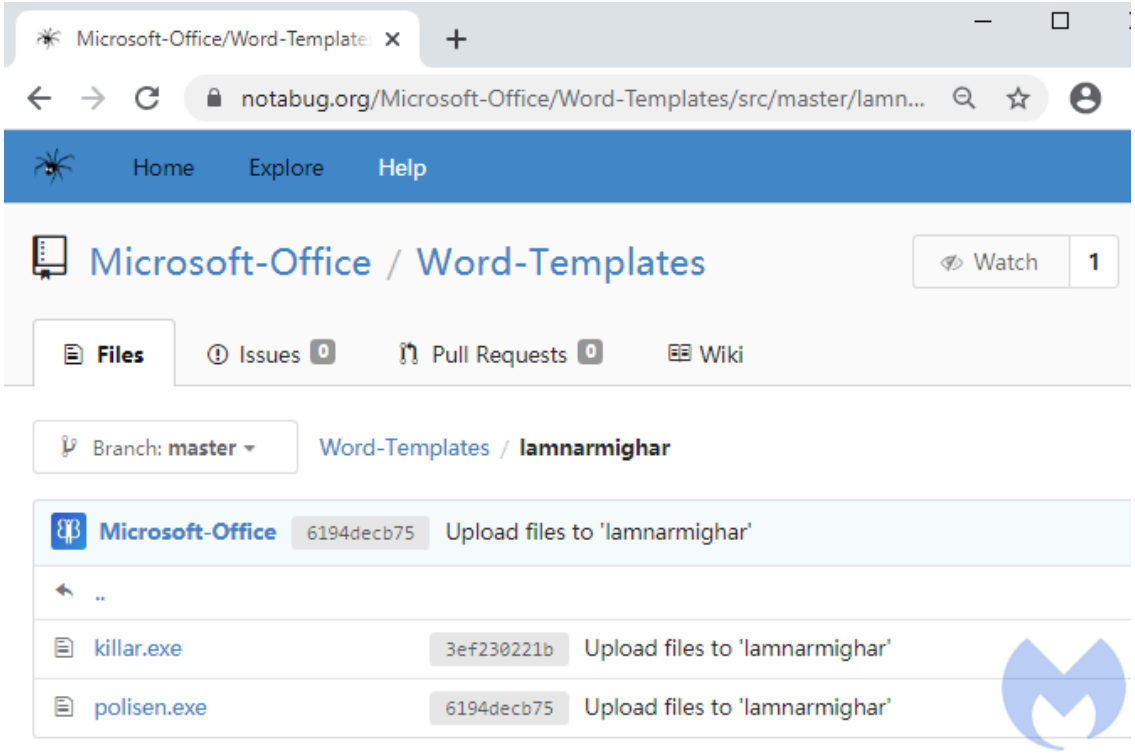
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



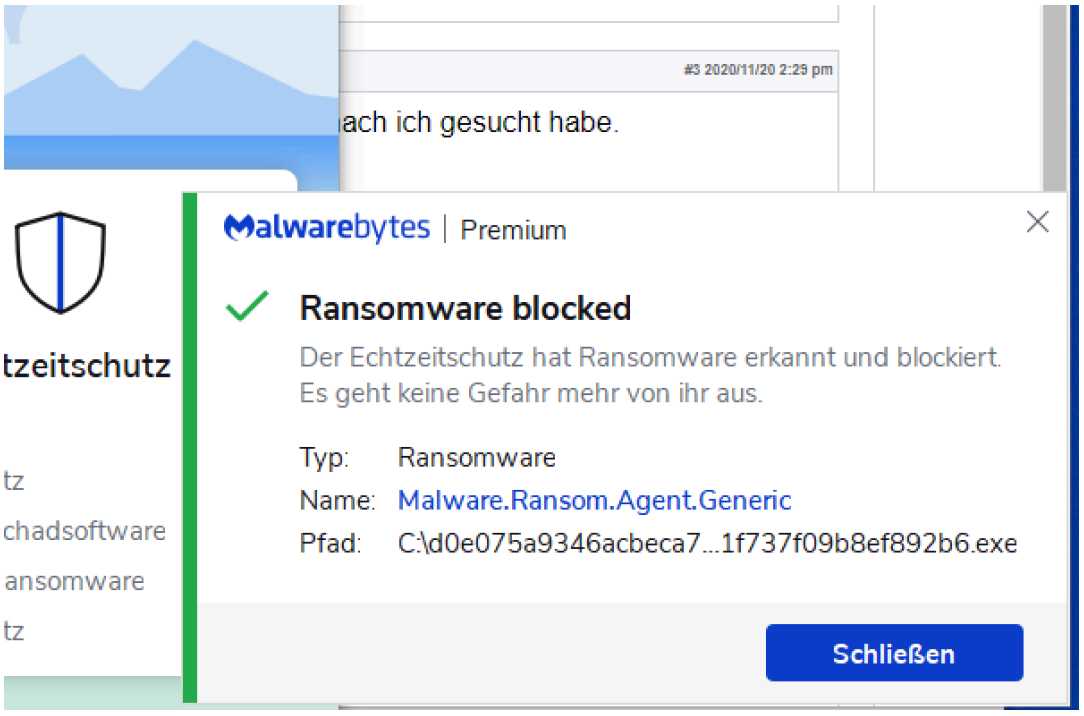
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

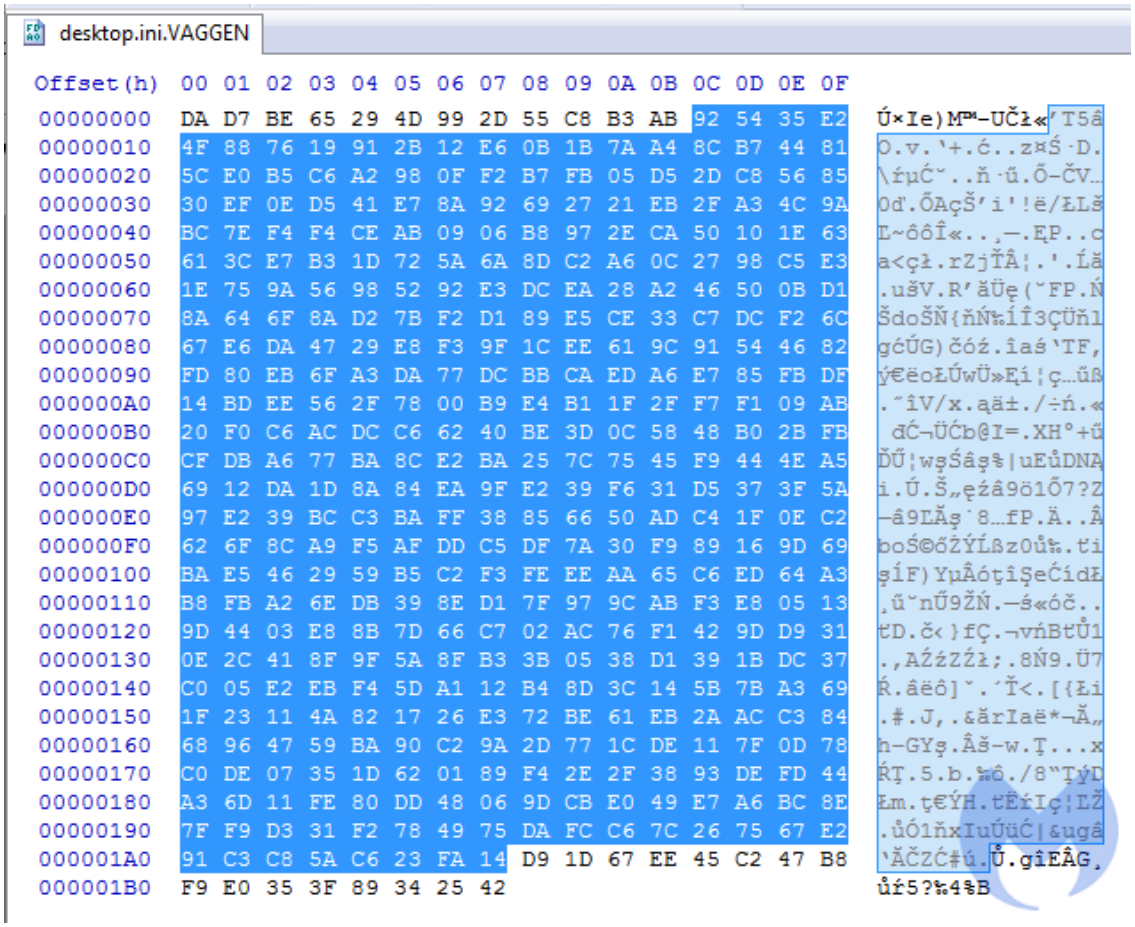
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



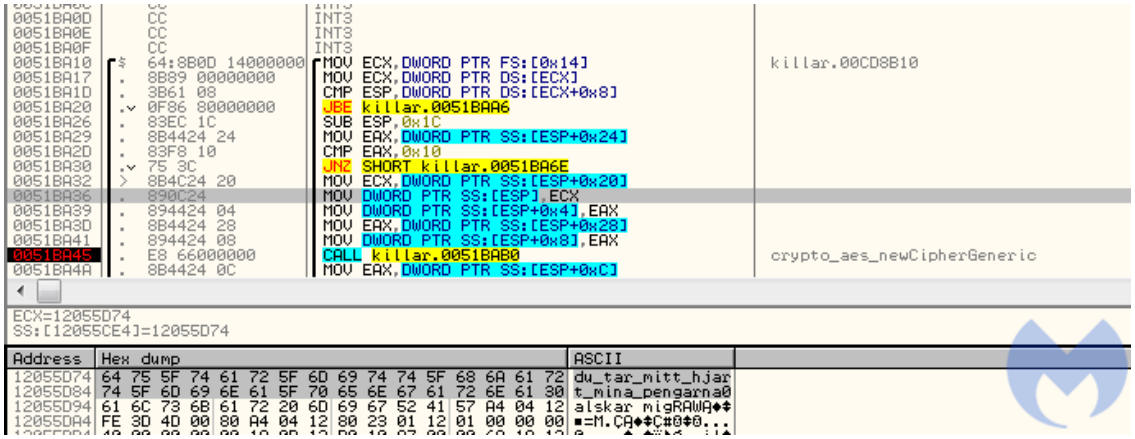
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

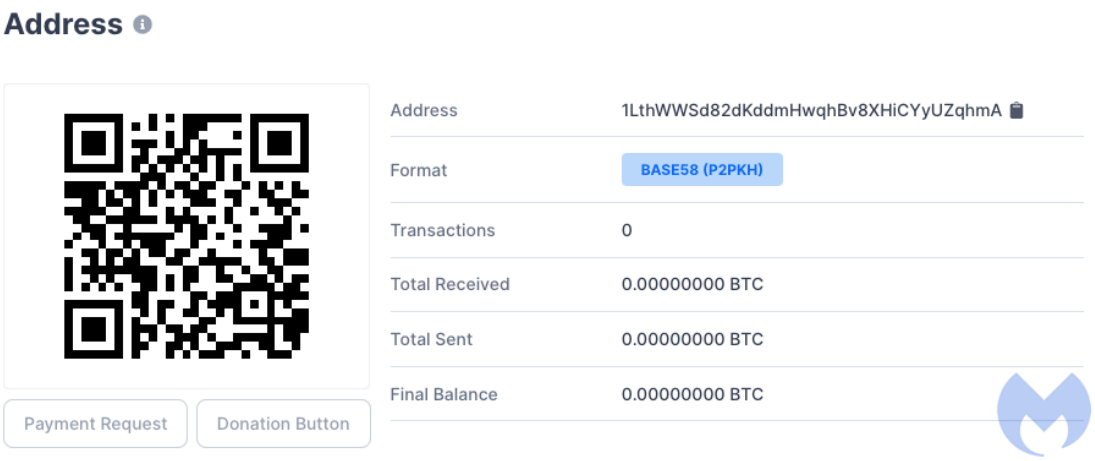
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

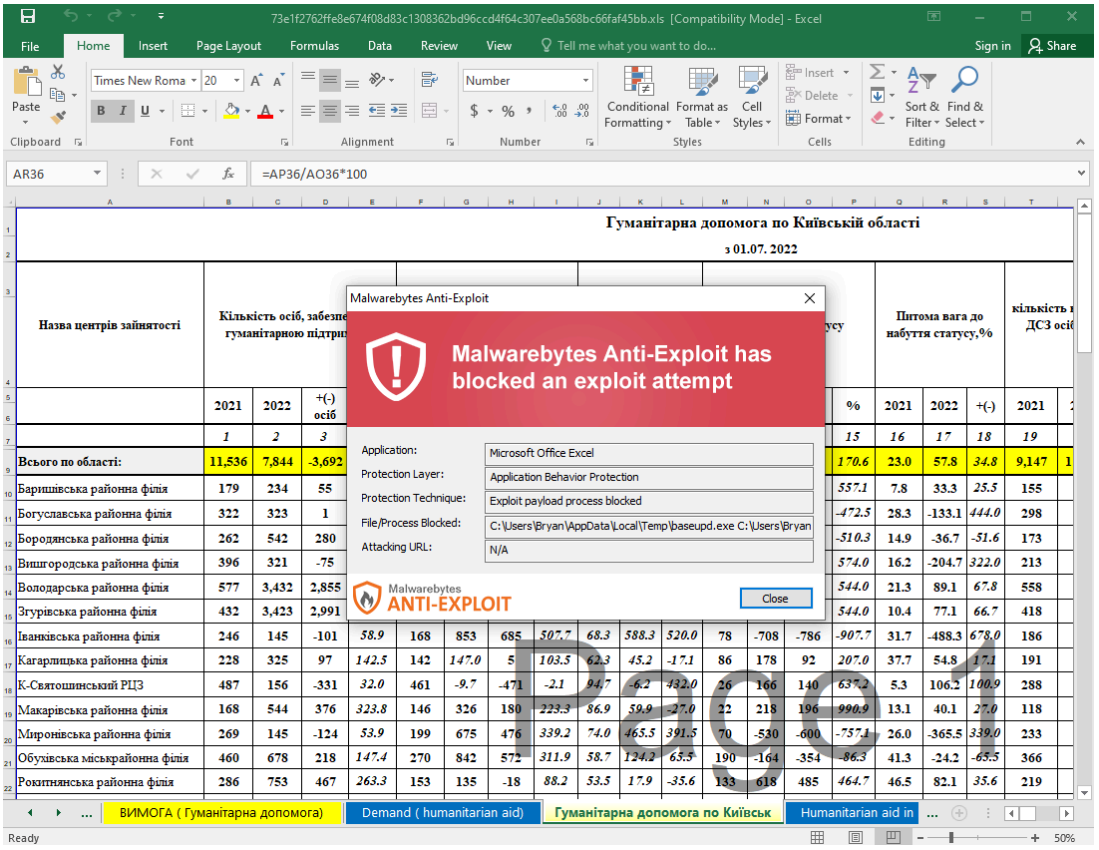
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

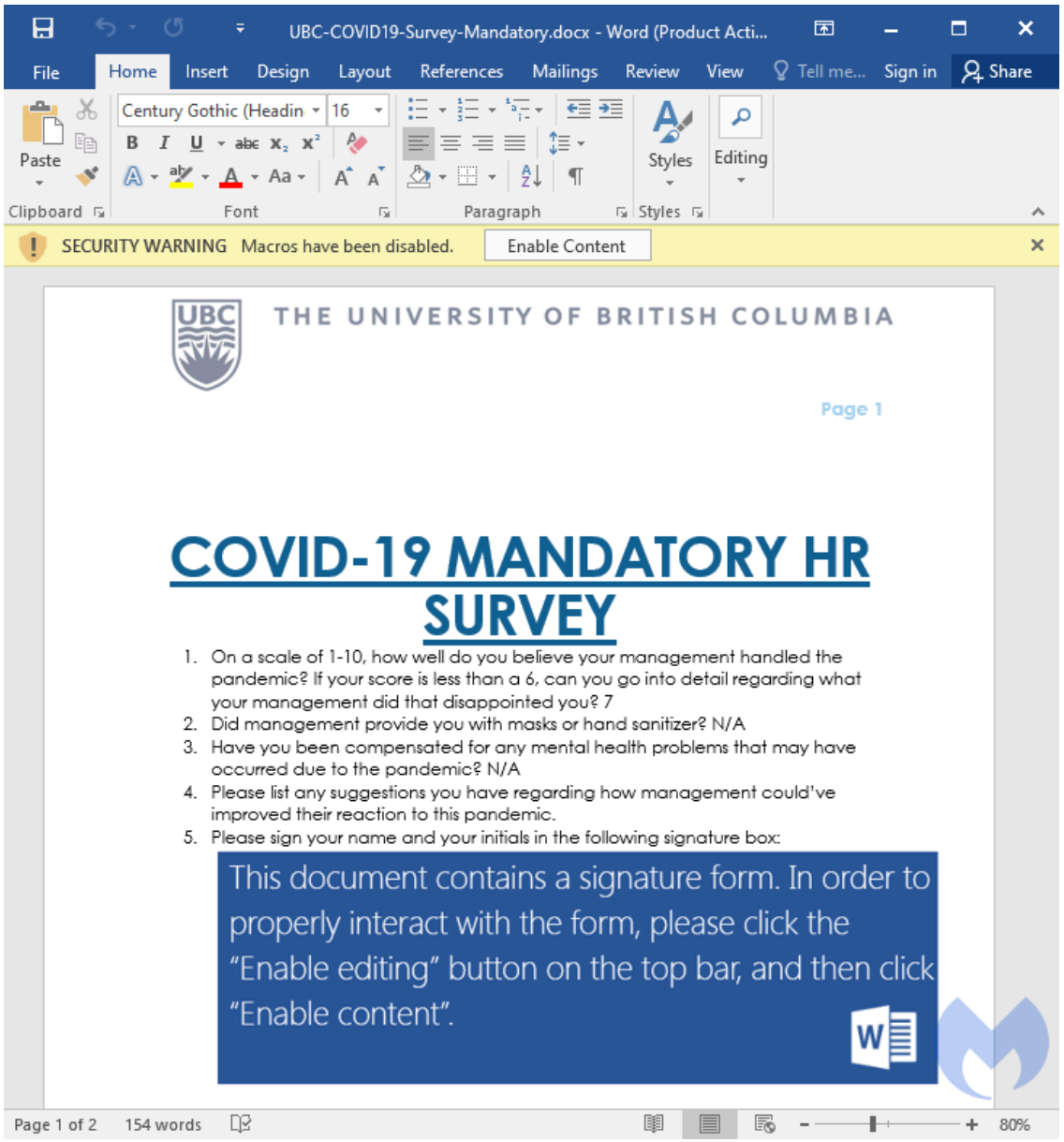
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

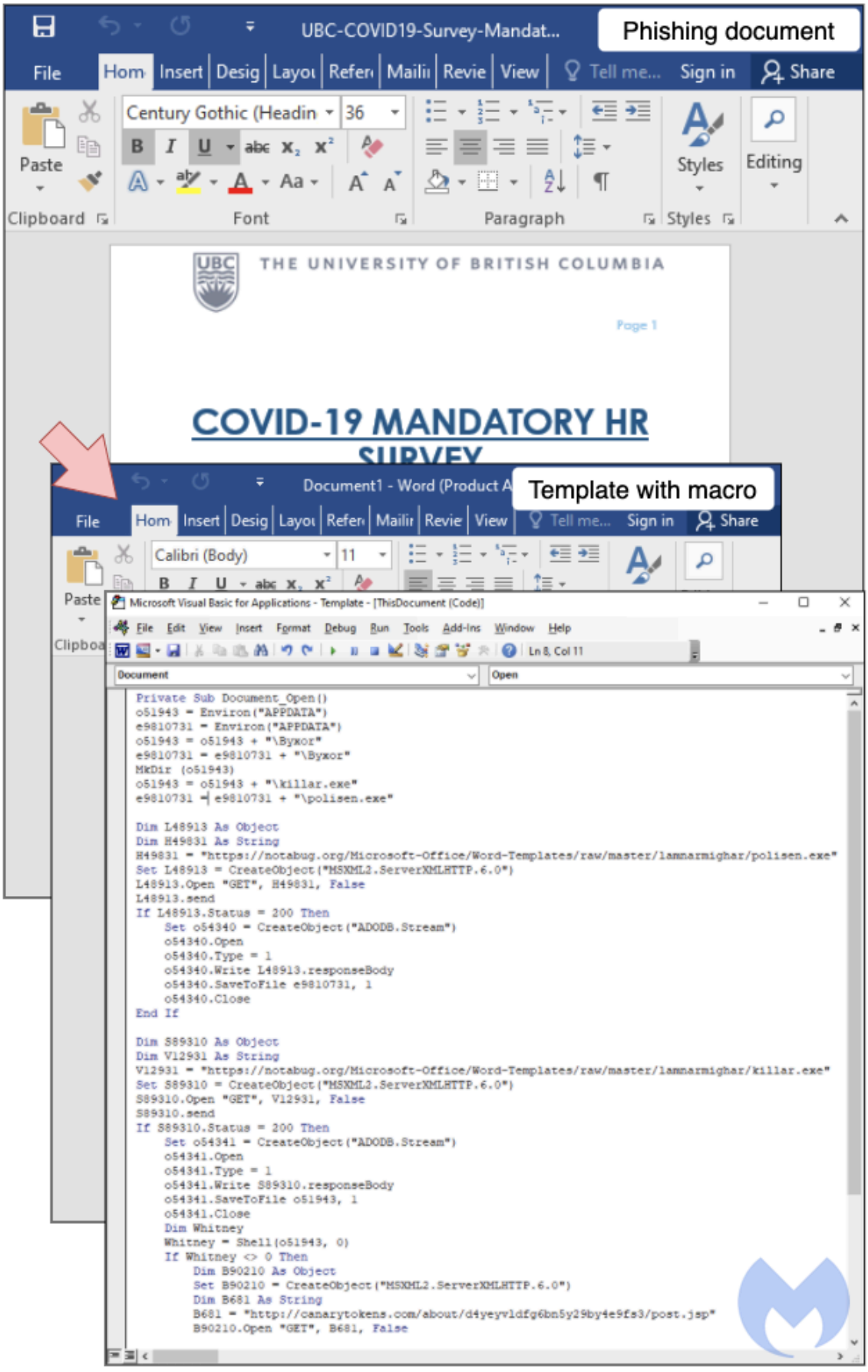
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

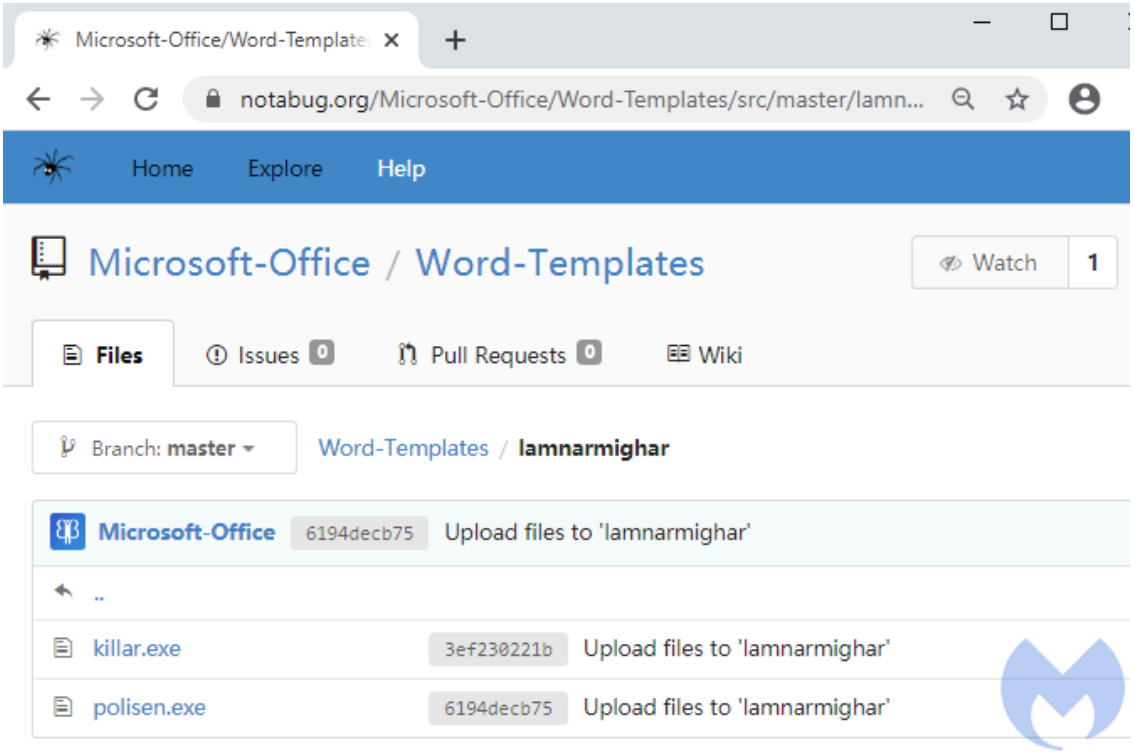
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



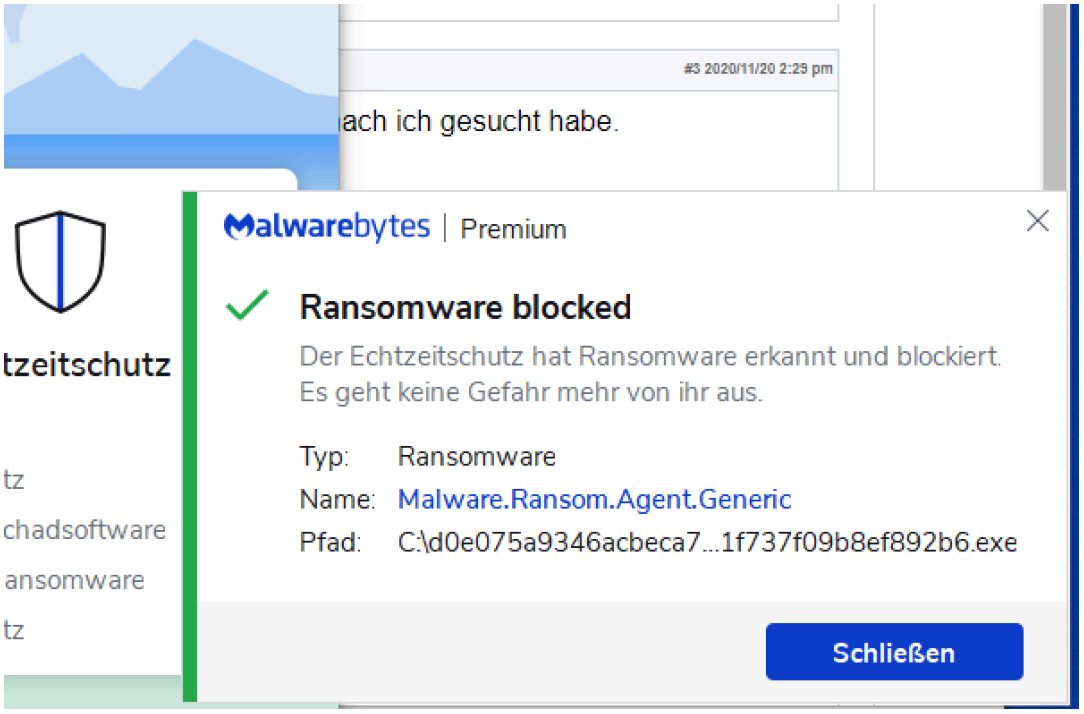
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

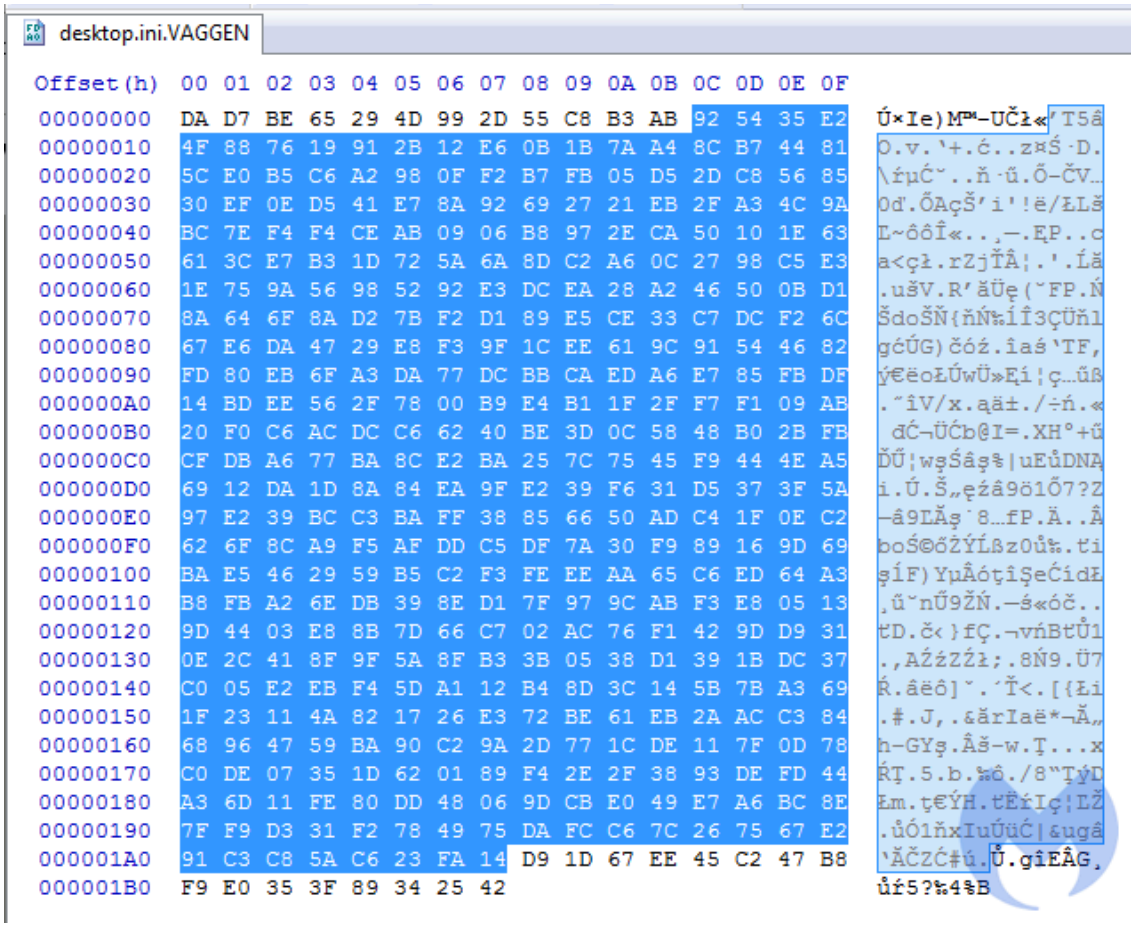
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



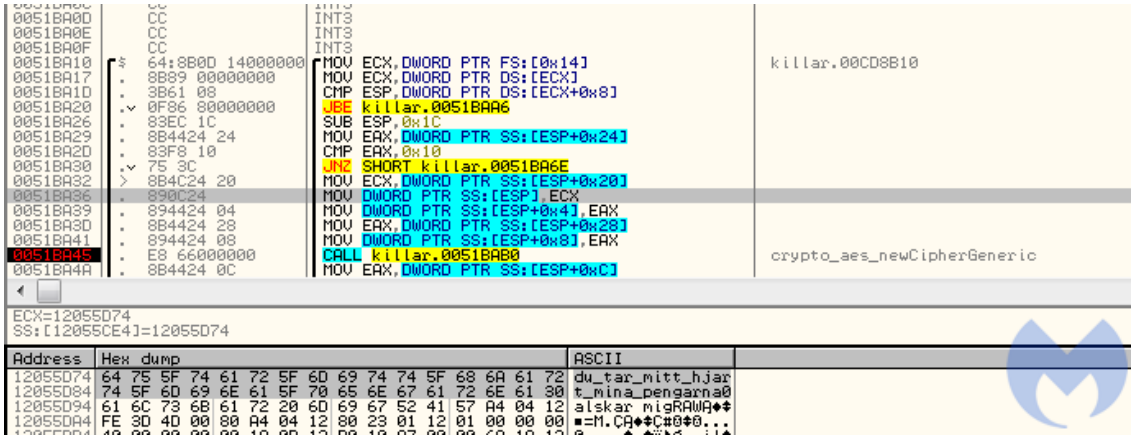
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

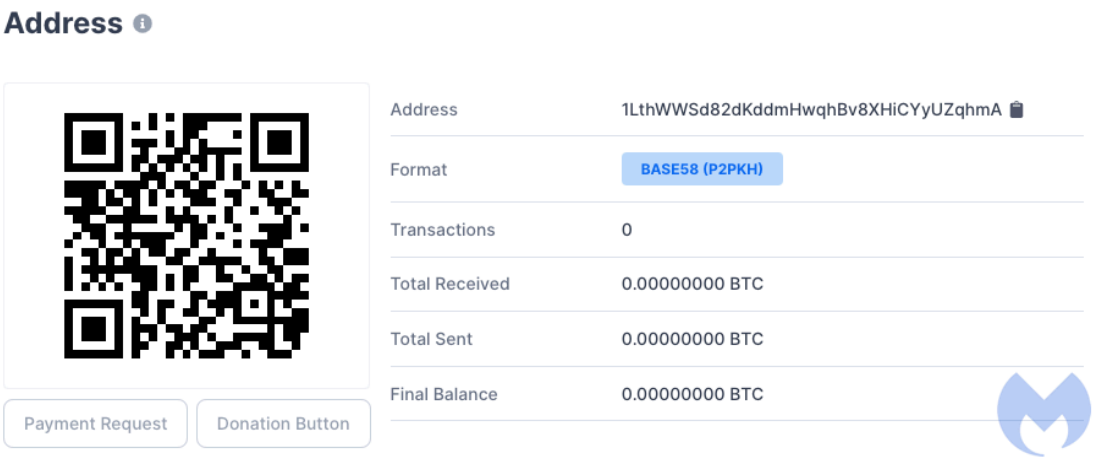
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

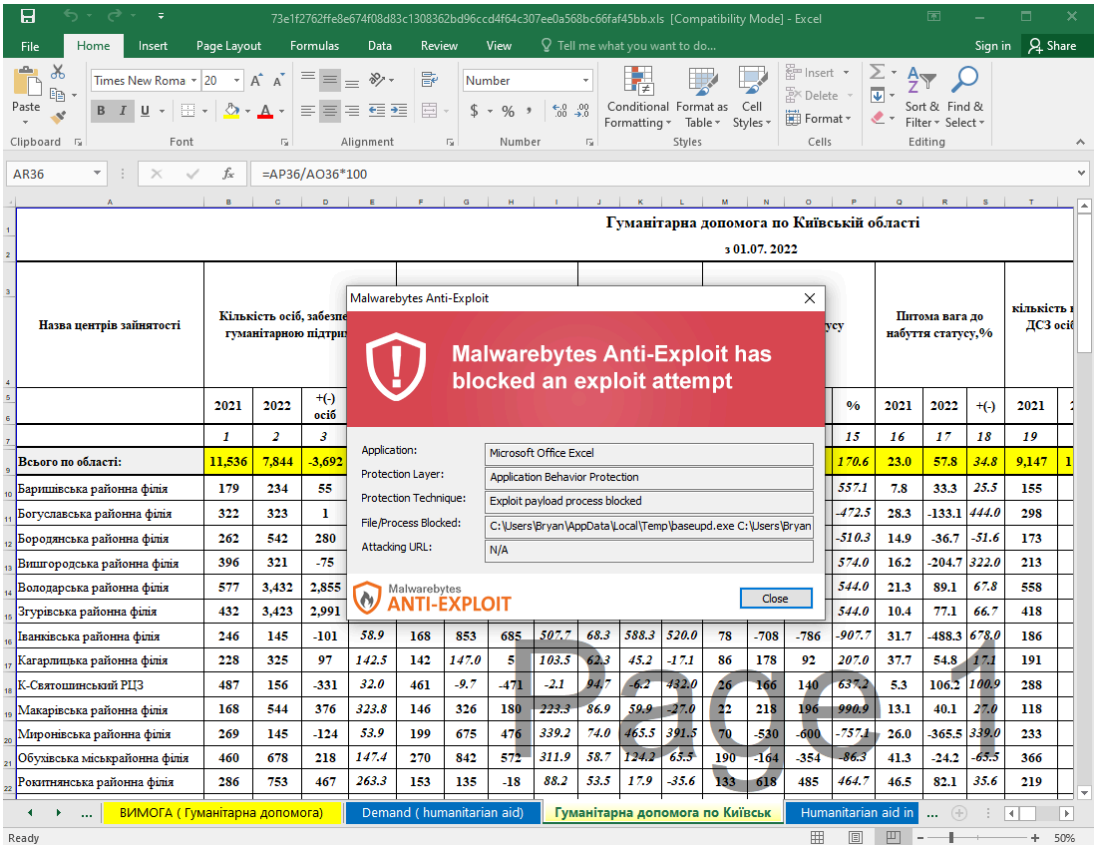
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

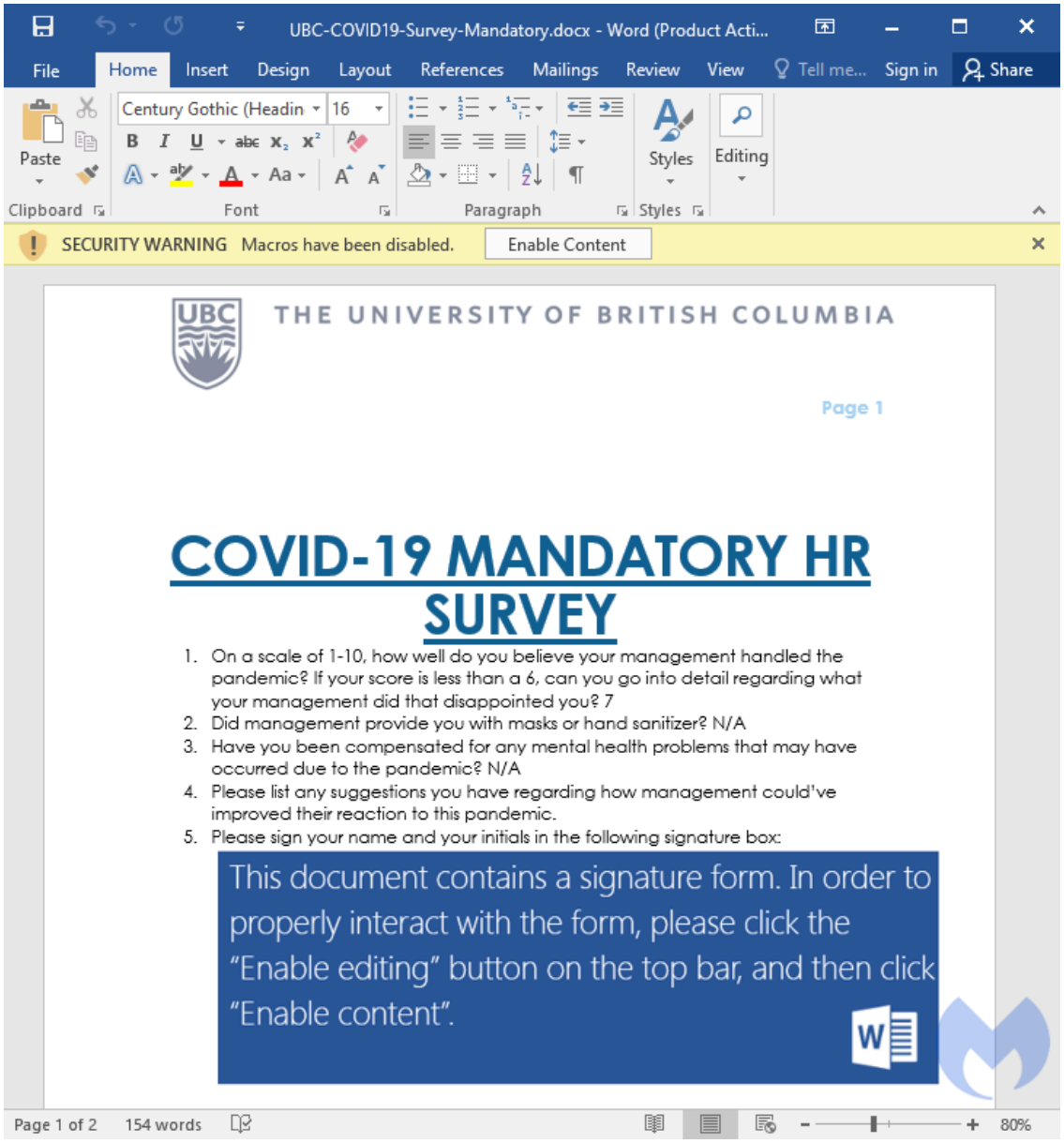
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

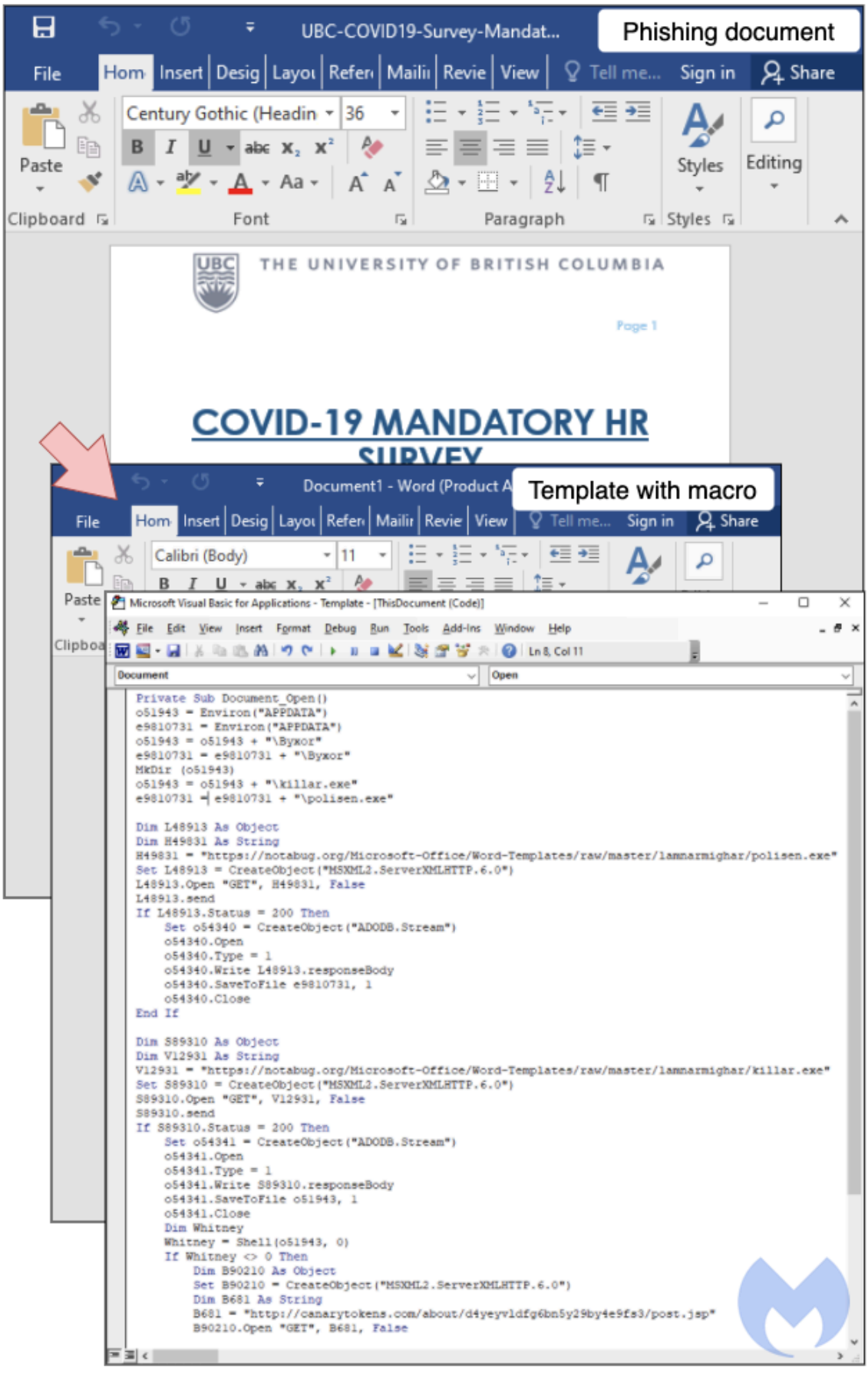
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

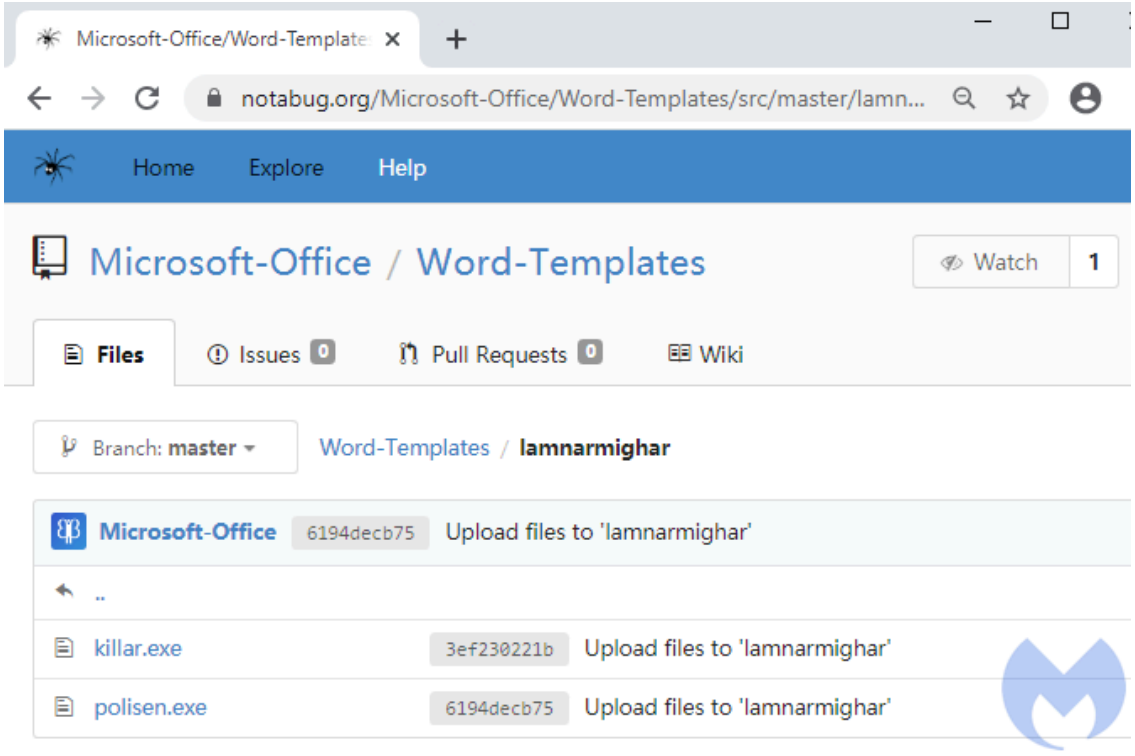
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



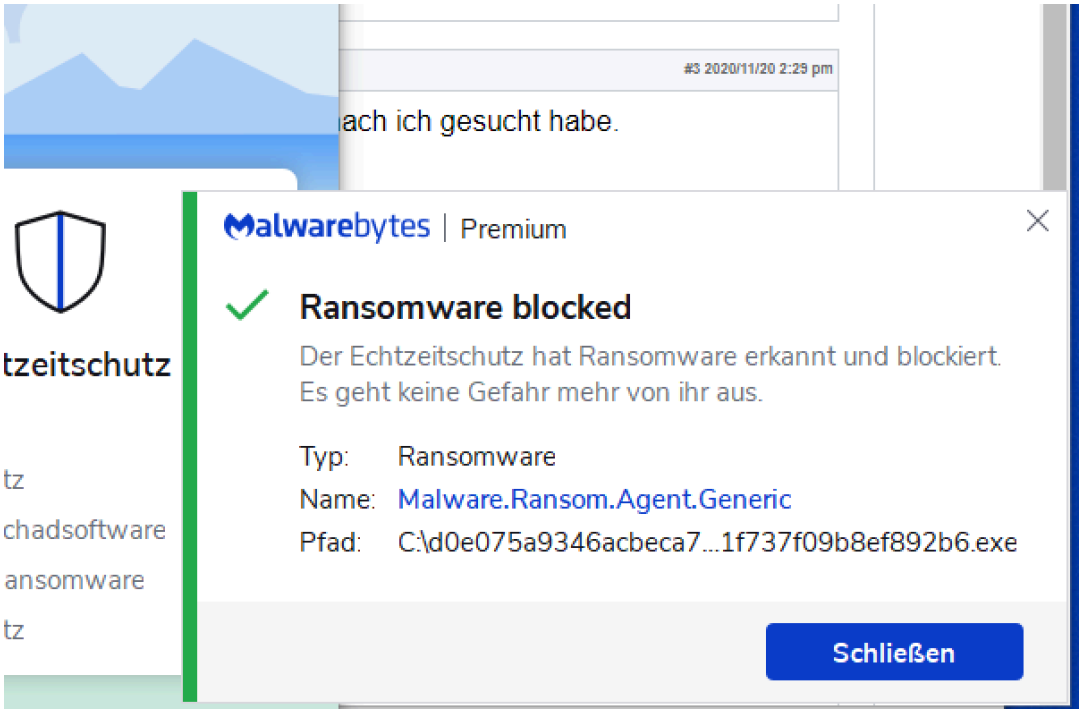
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

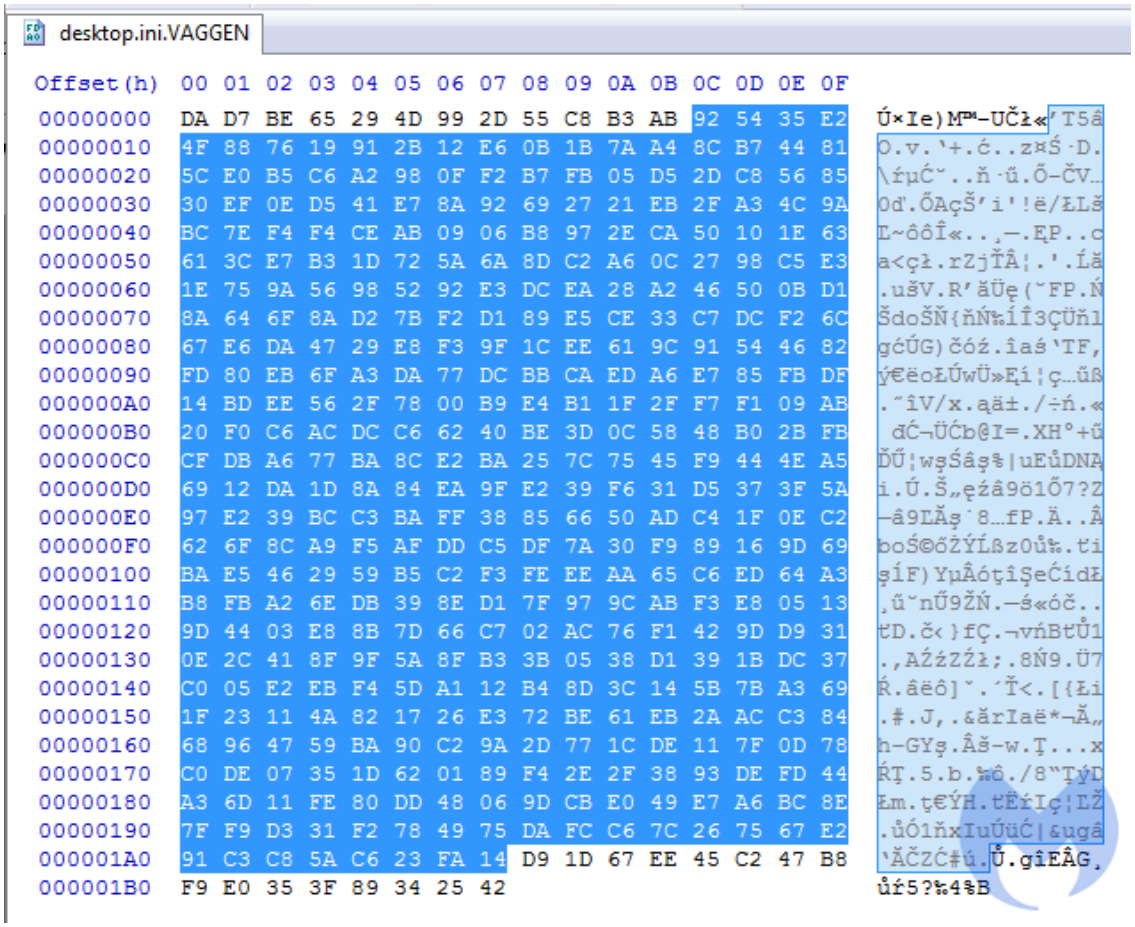
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



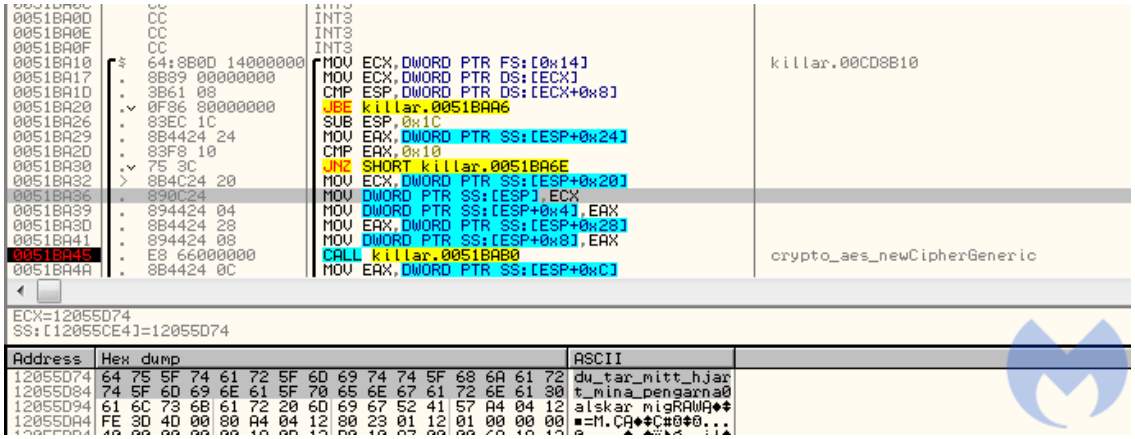
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

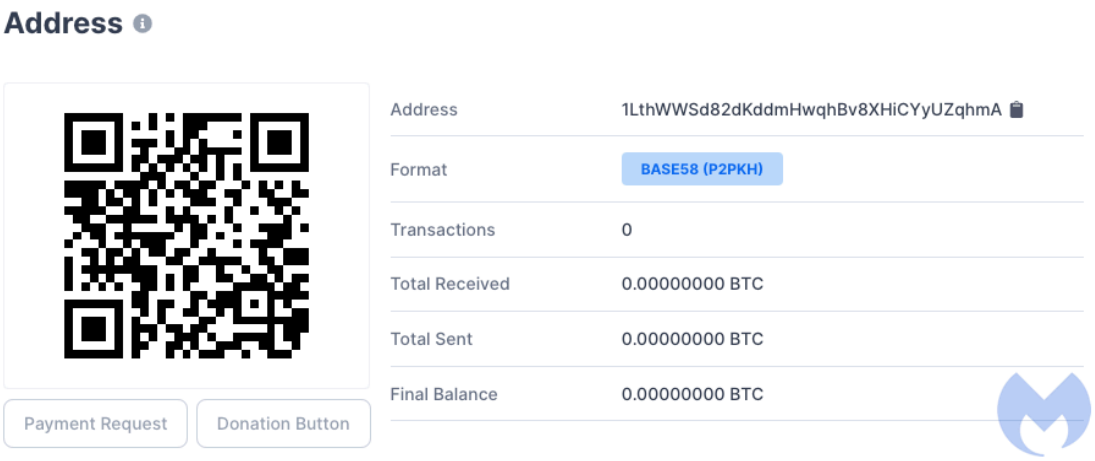
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

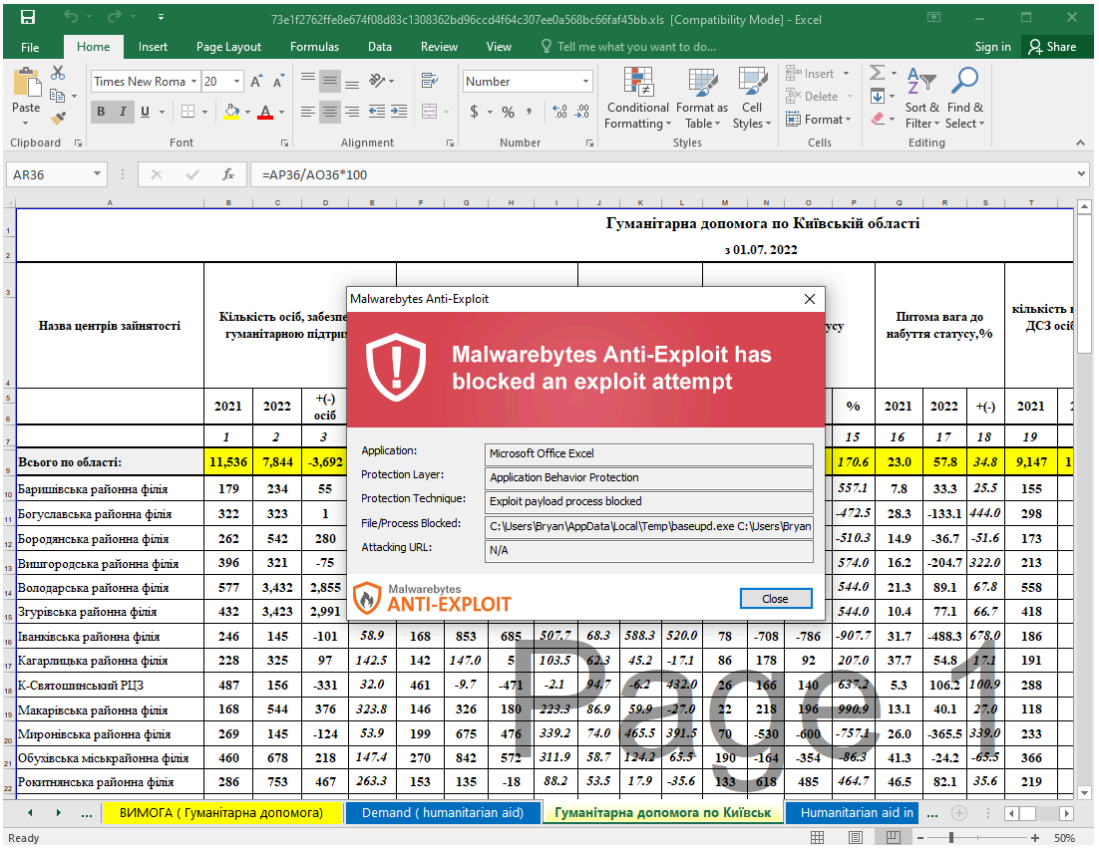
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

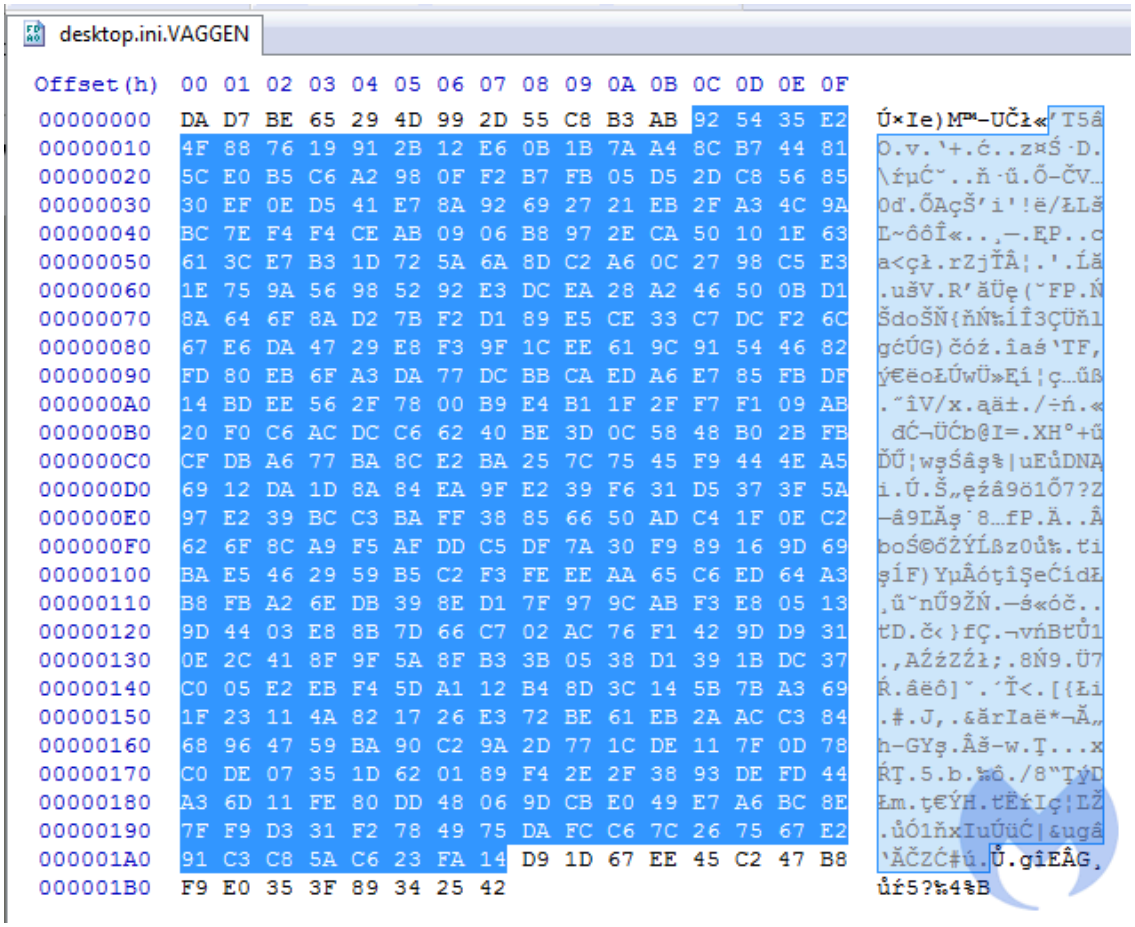
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

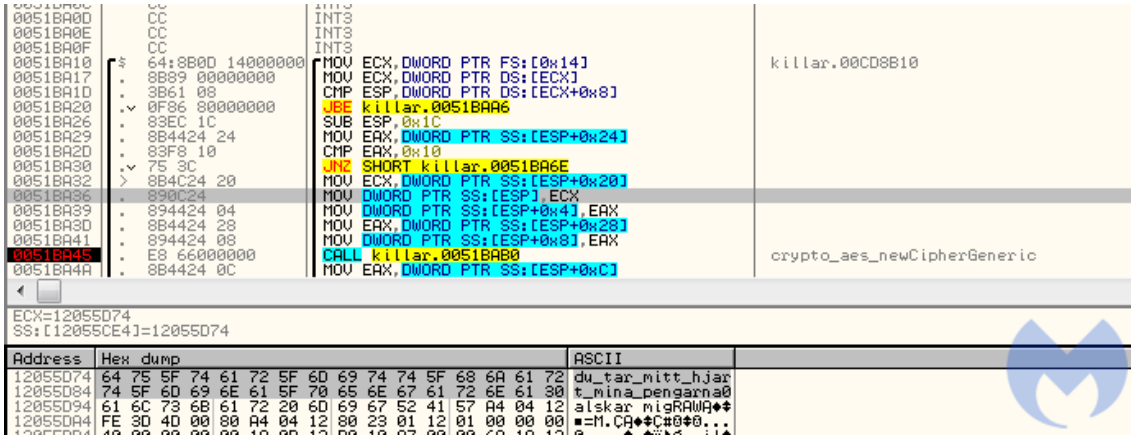


The content of the output file (with .VAGGEN extension) contains:

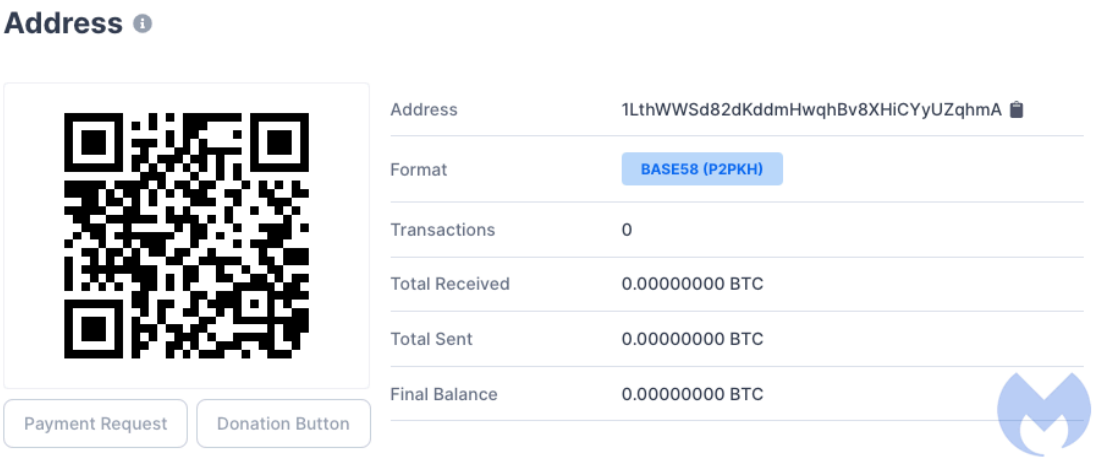
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

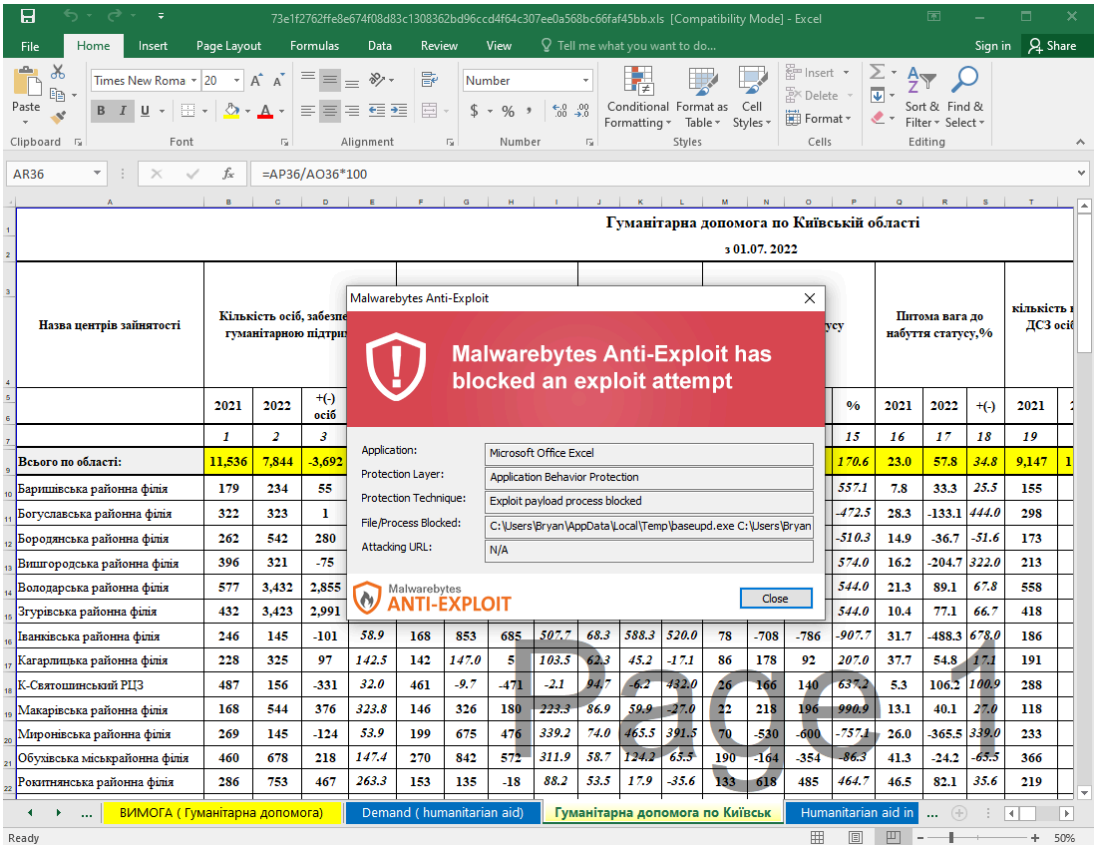
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

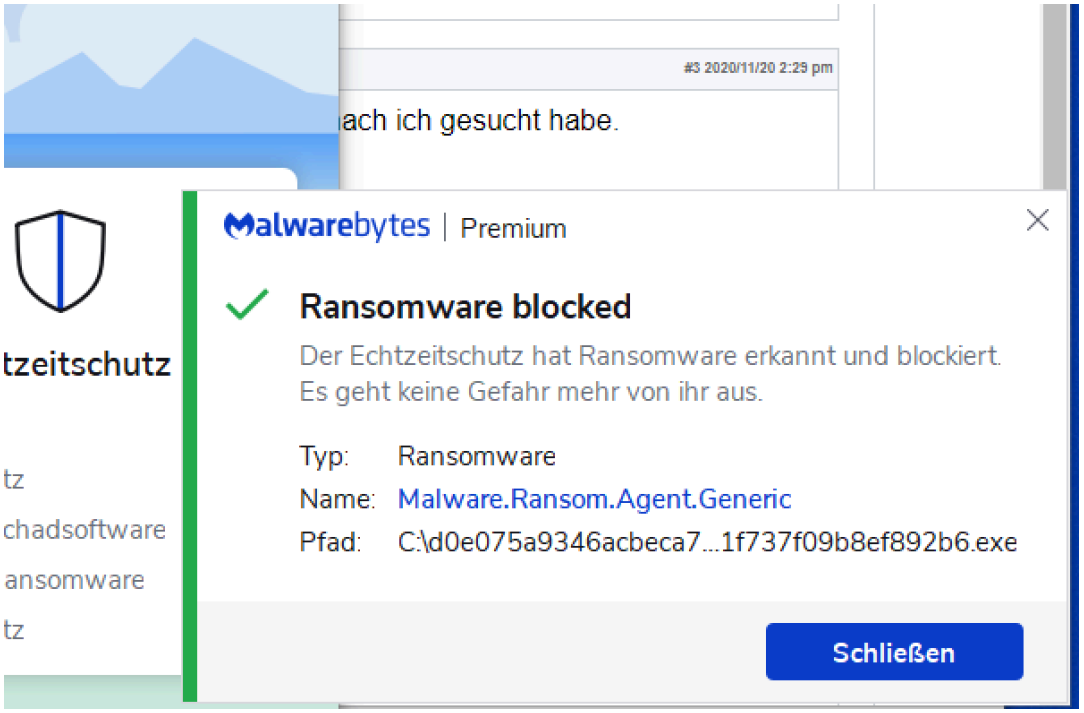
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

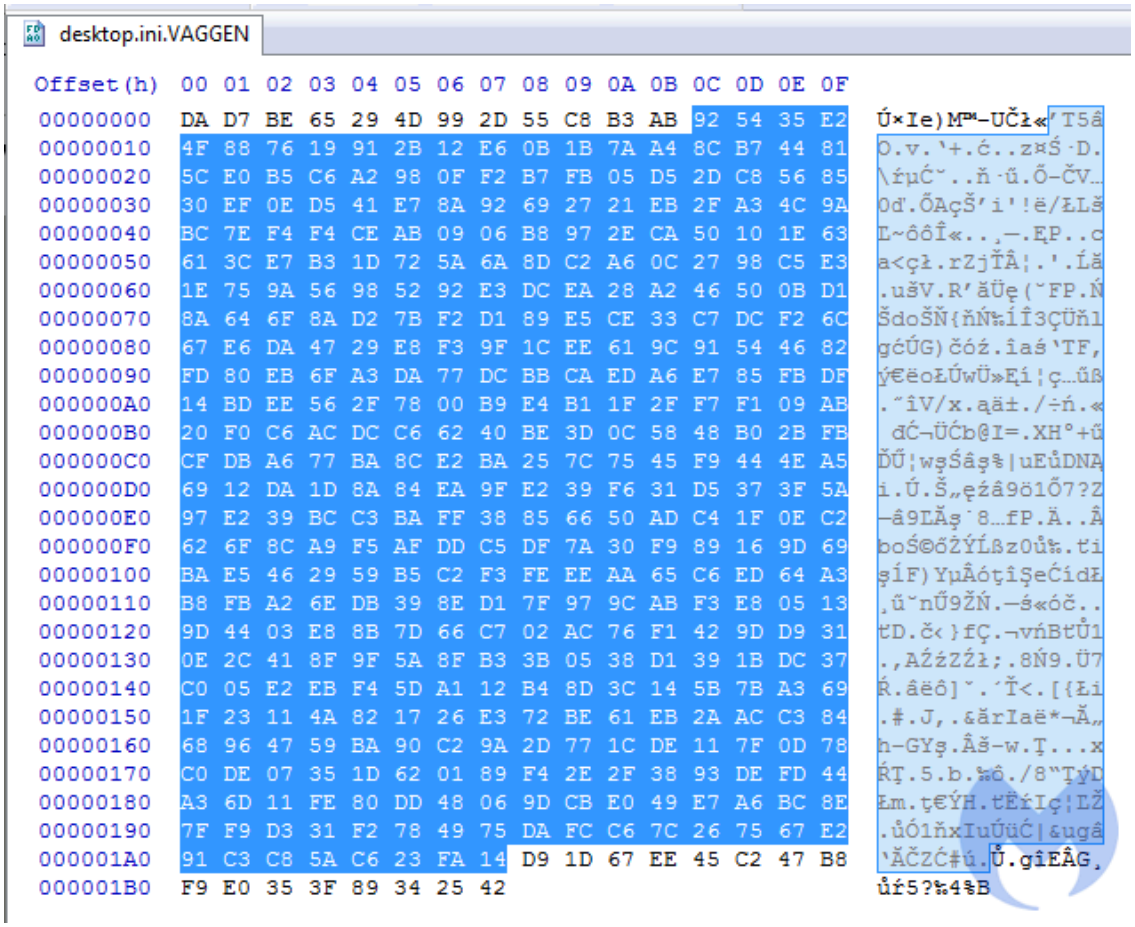
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

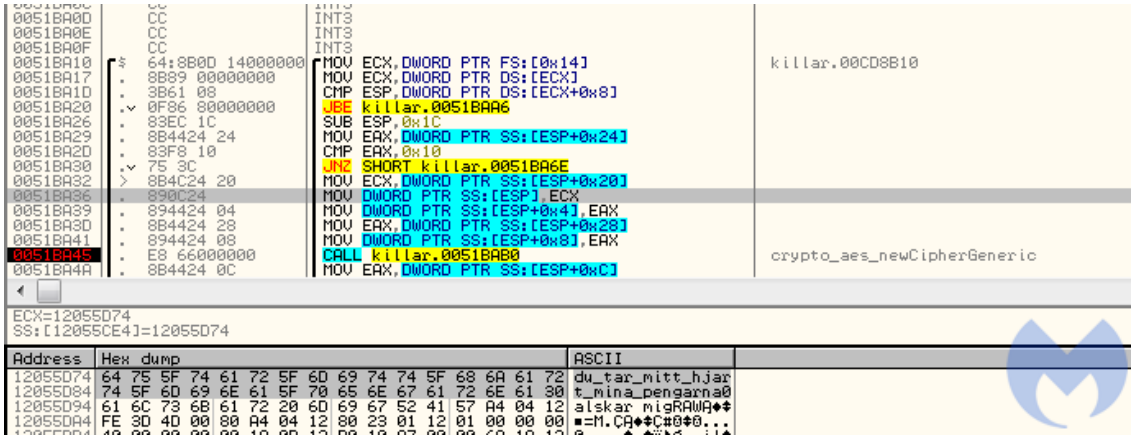


The content of the output file (with .VAGGEN extension) contains:

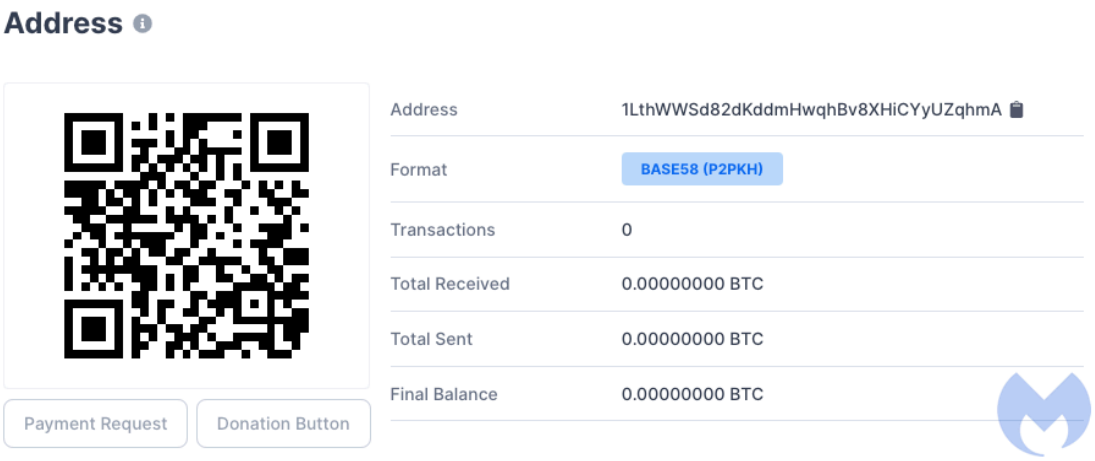
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

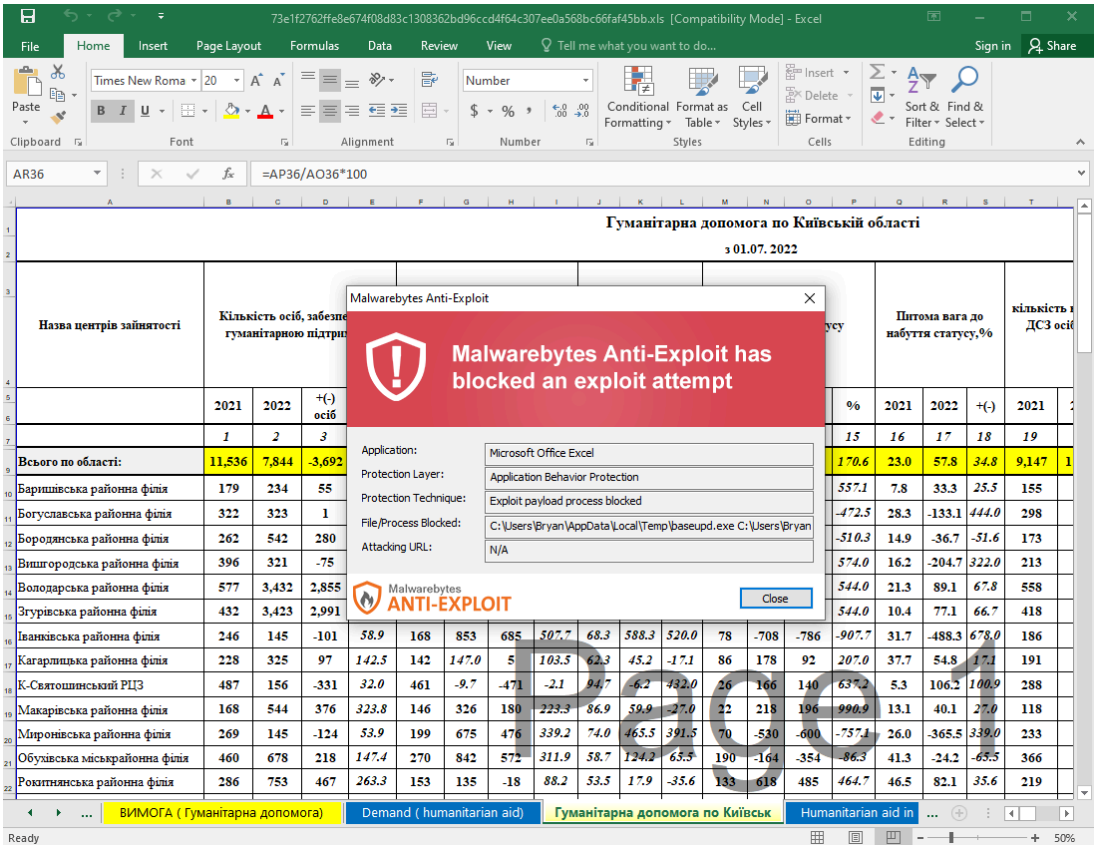
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

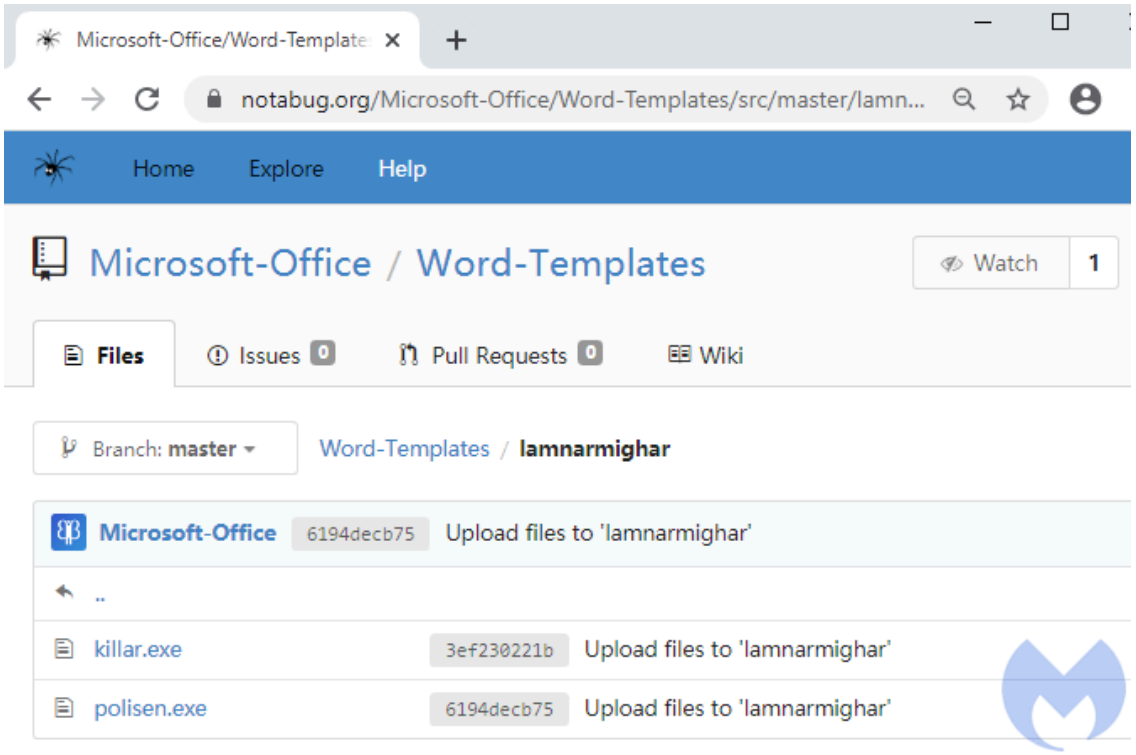
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



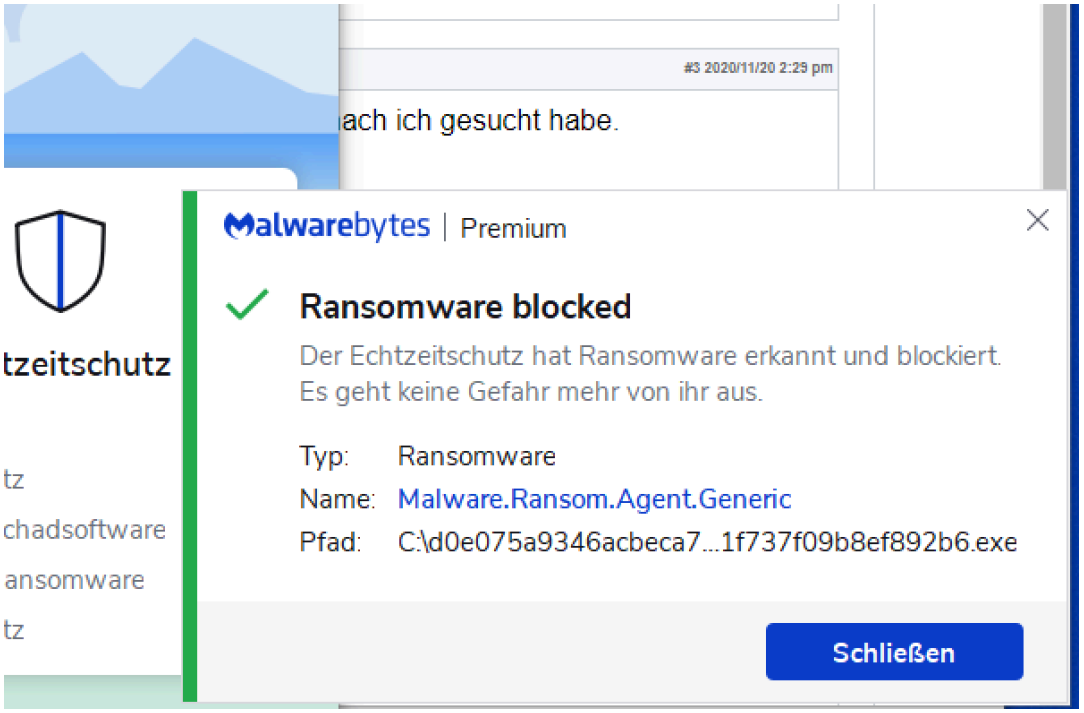
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

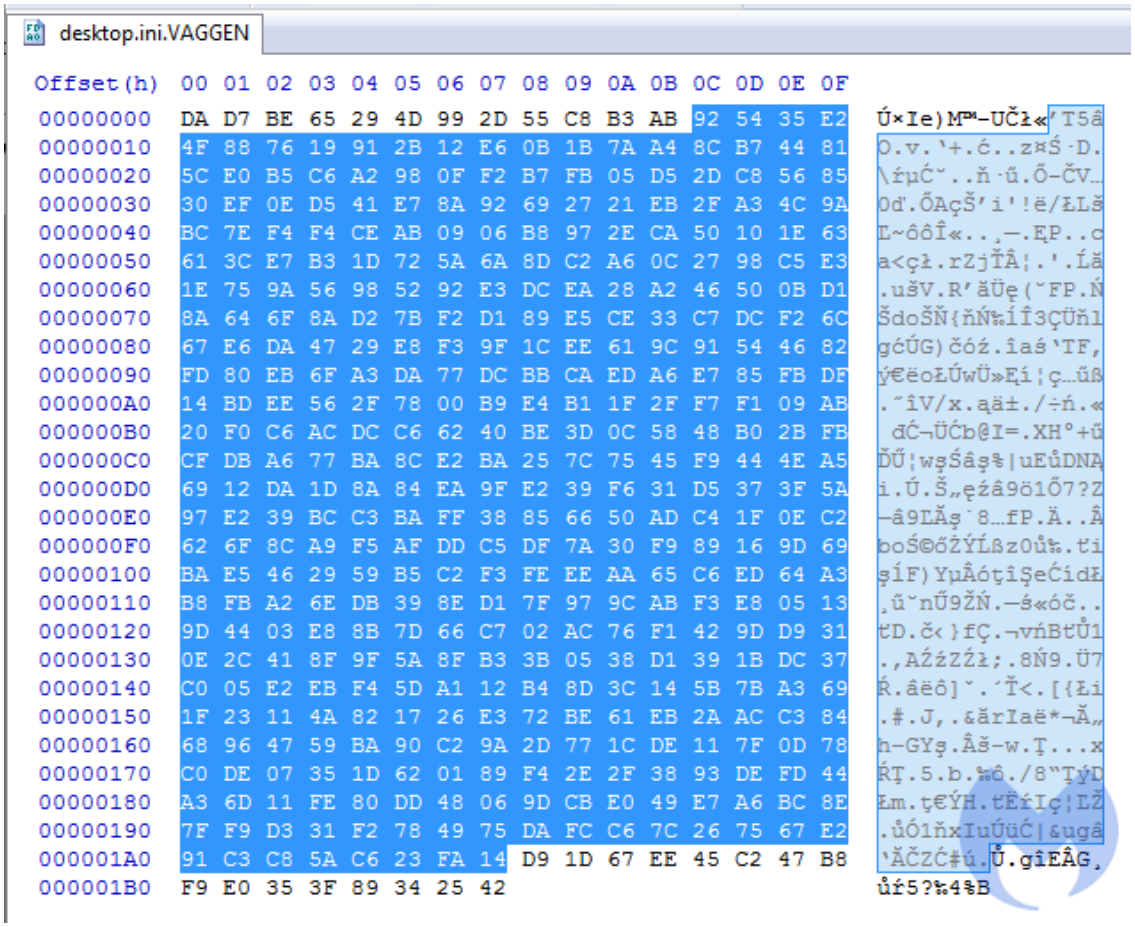
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

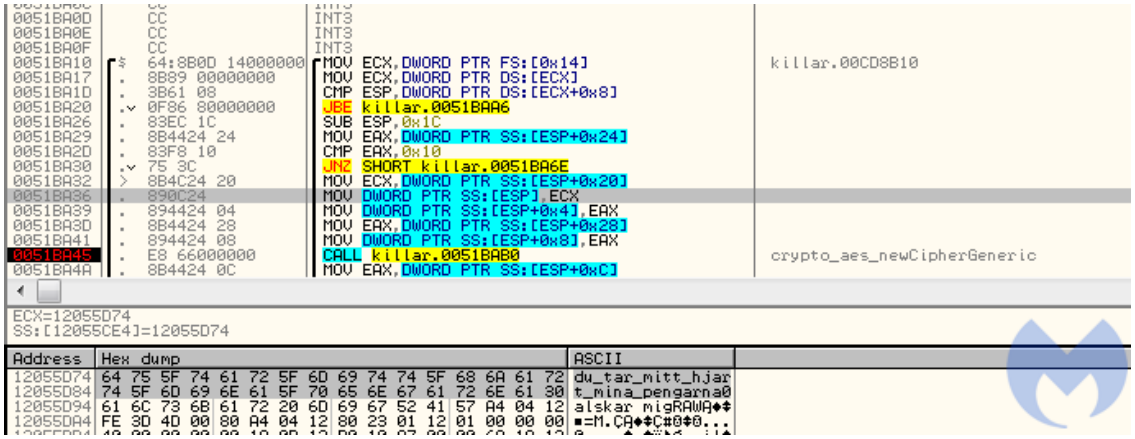


The content of the output file (with .VAGGEN extension) contains:

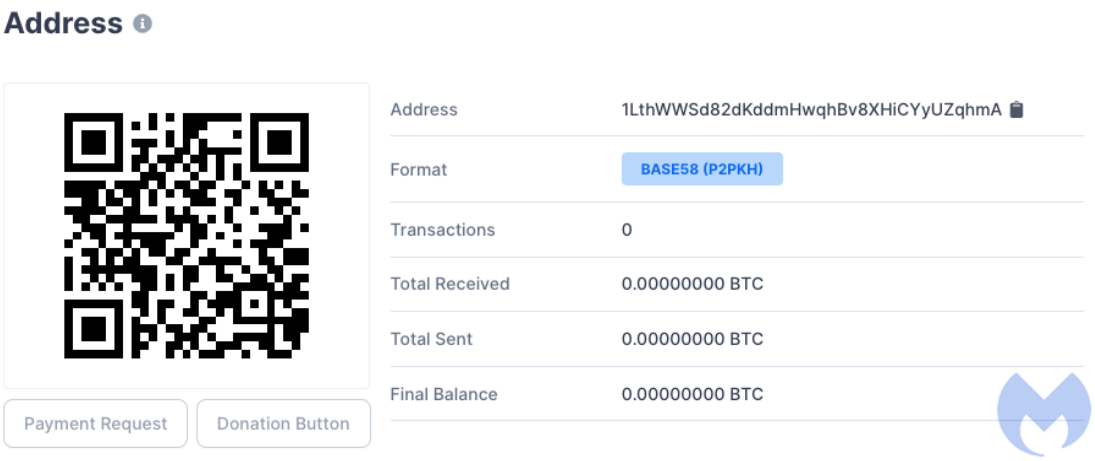
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

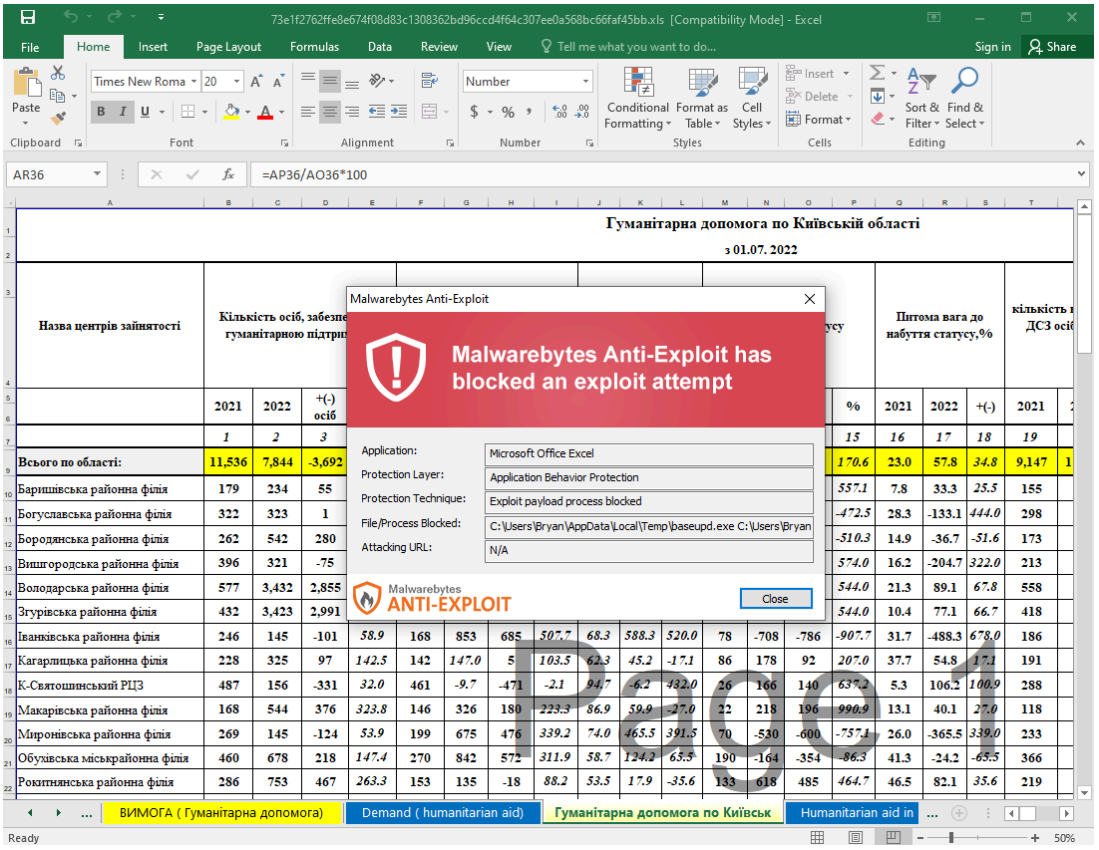
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

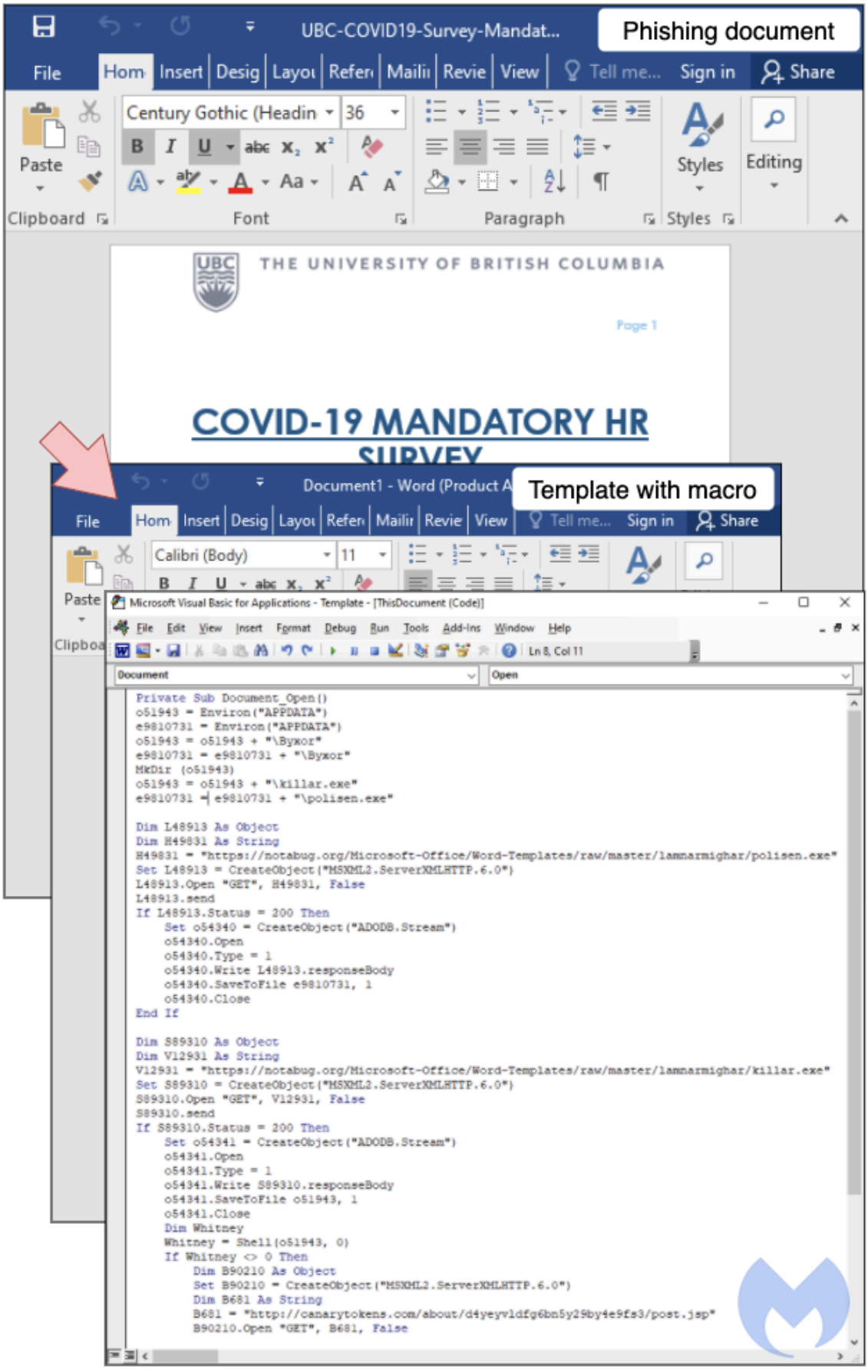
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

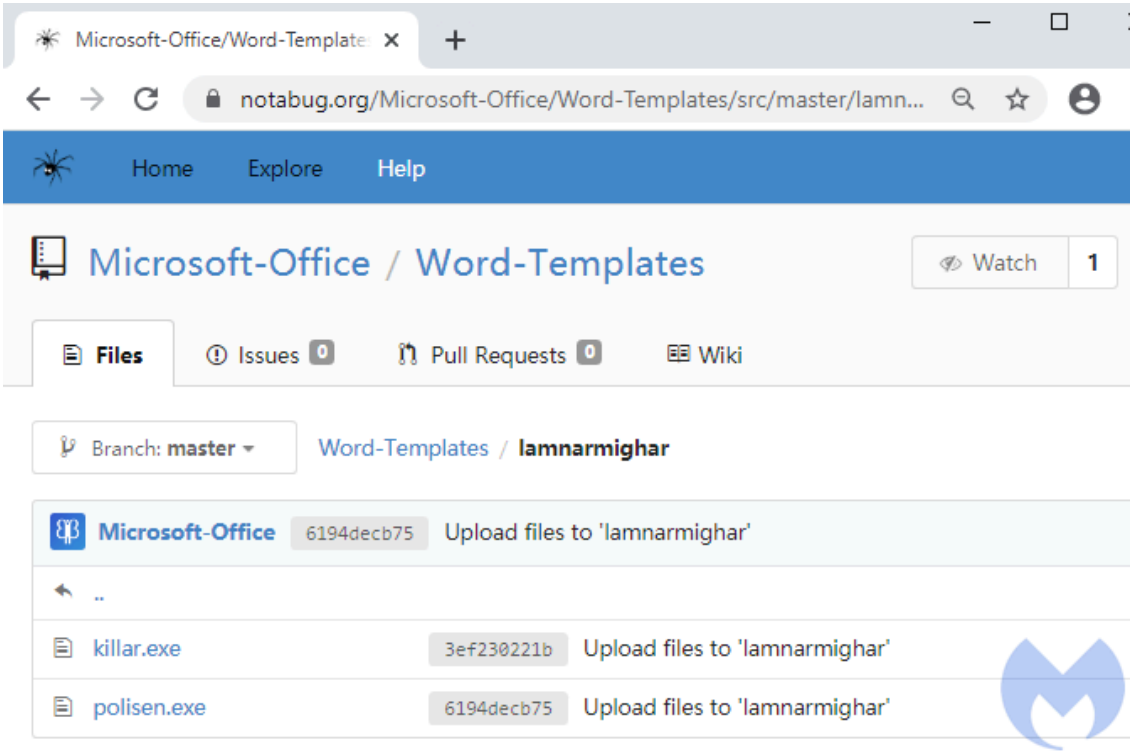
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



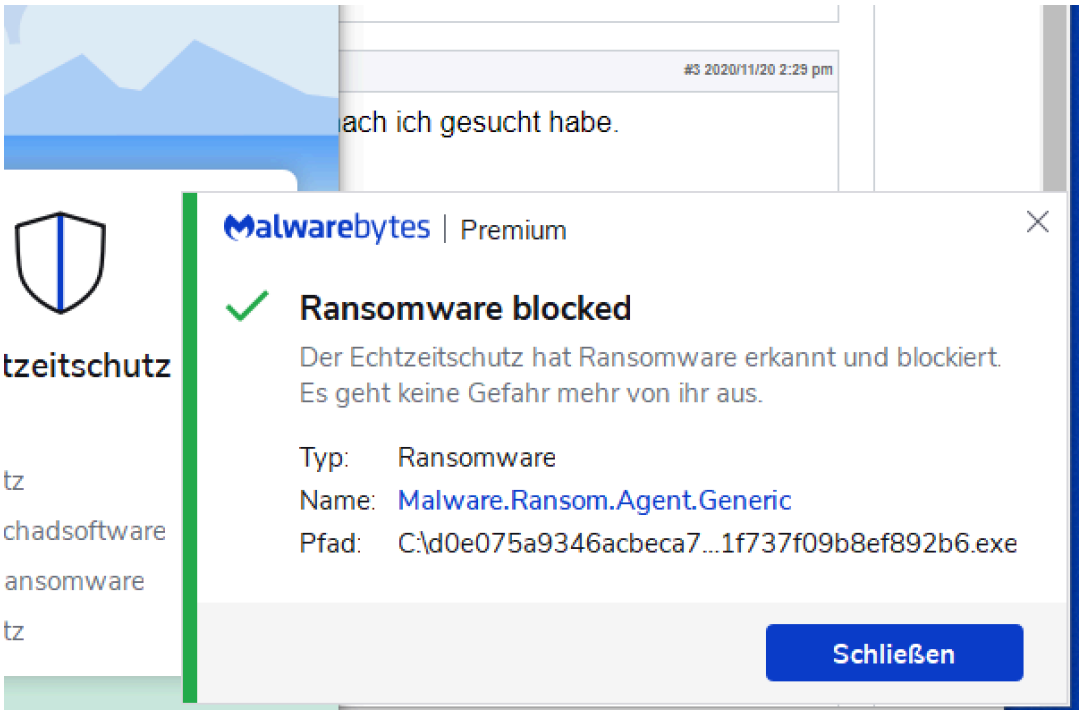
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

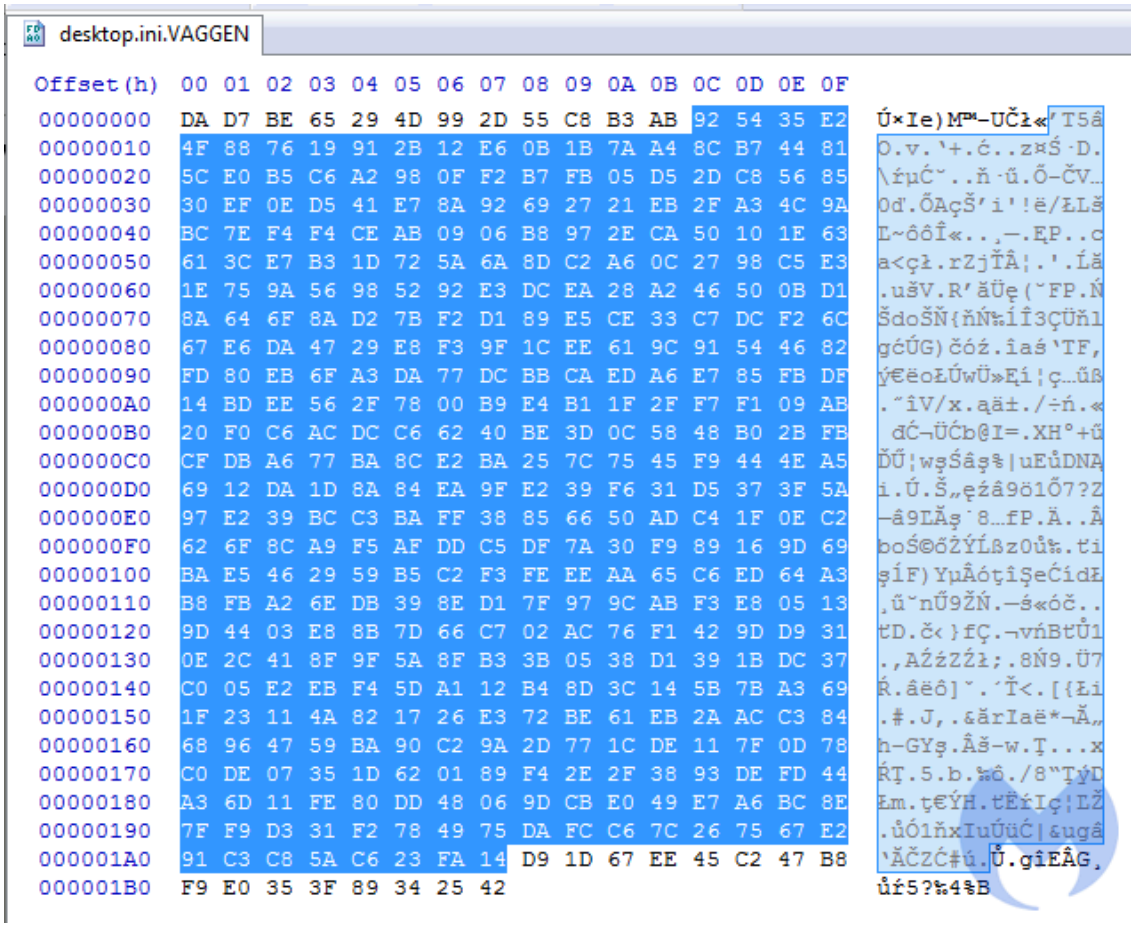
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



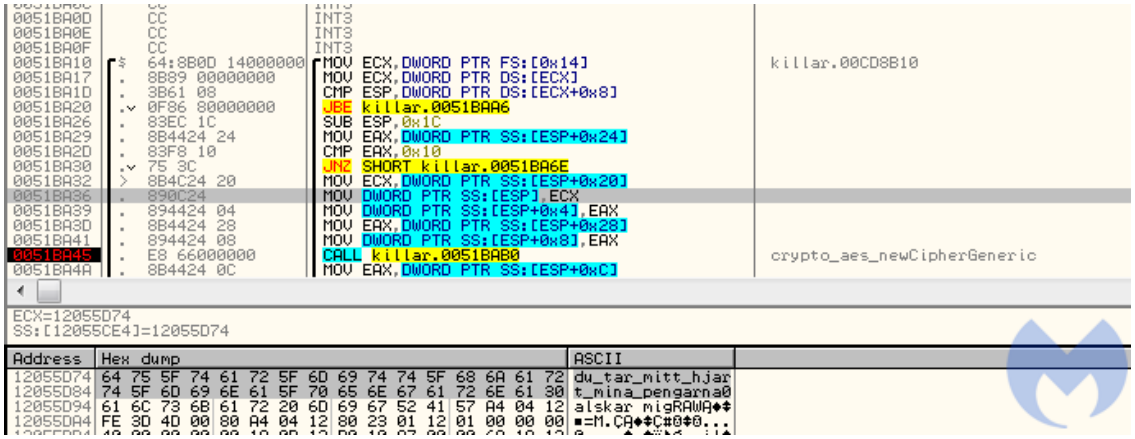
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

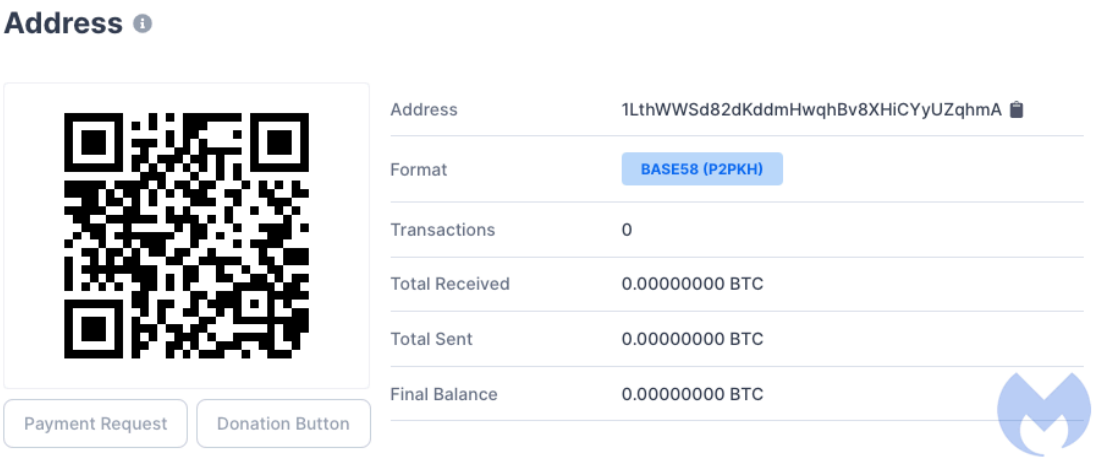
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

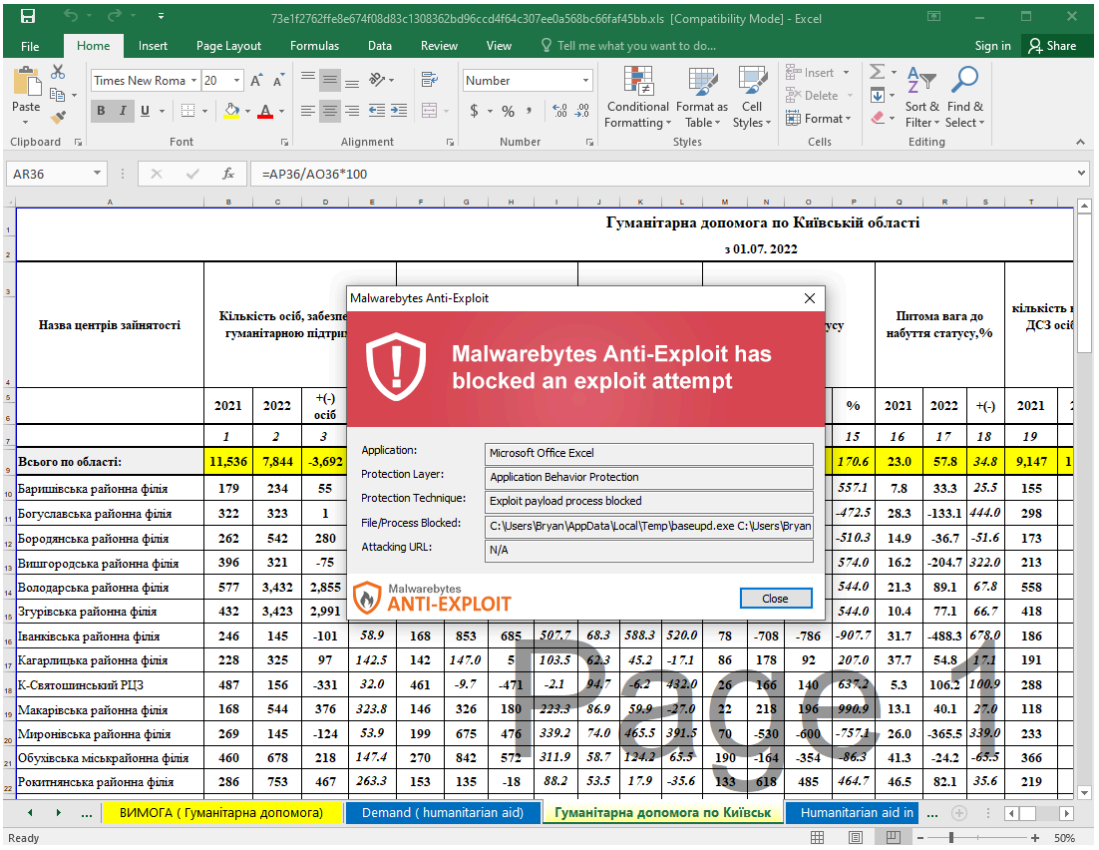
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

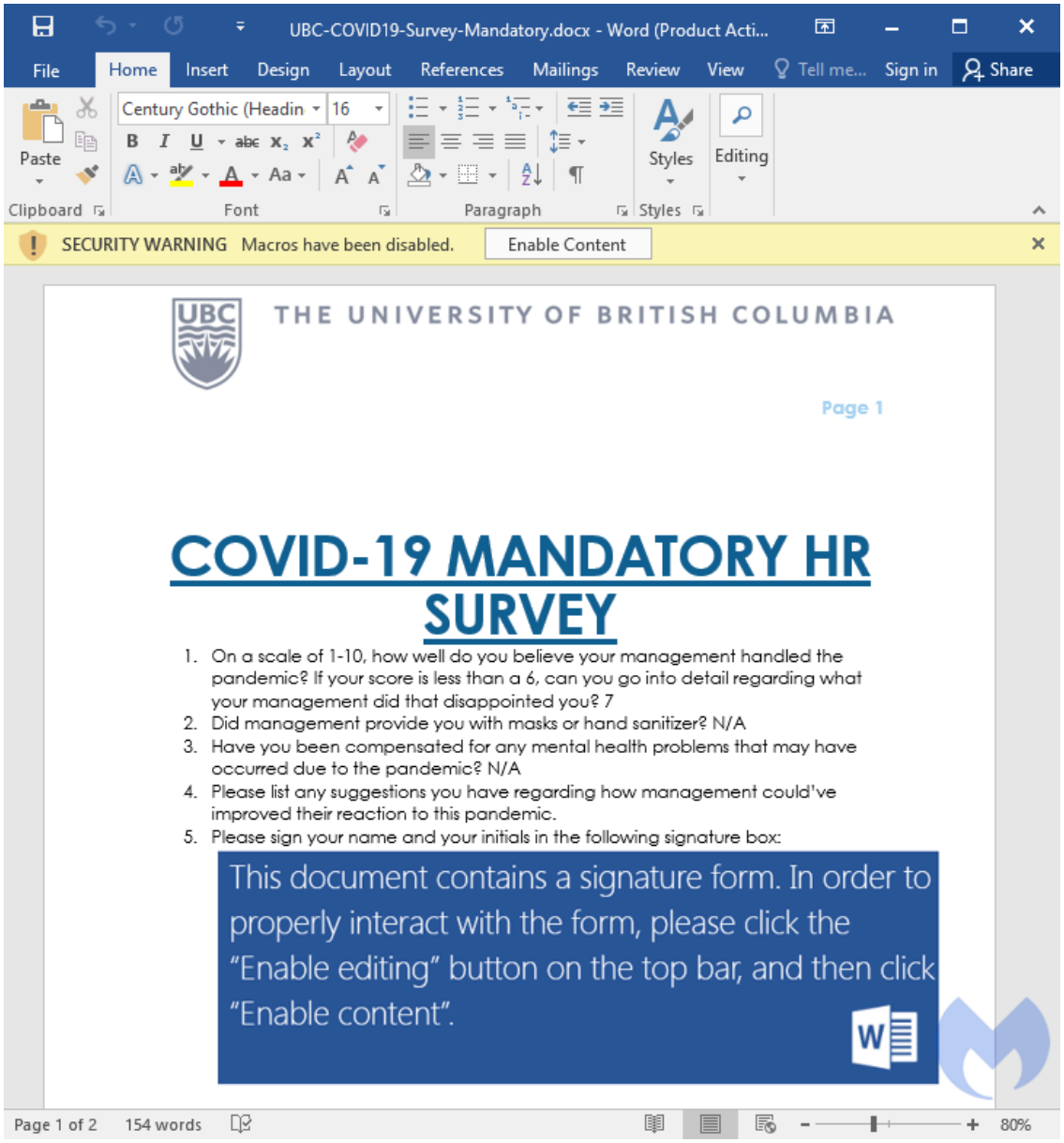
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

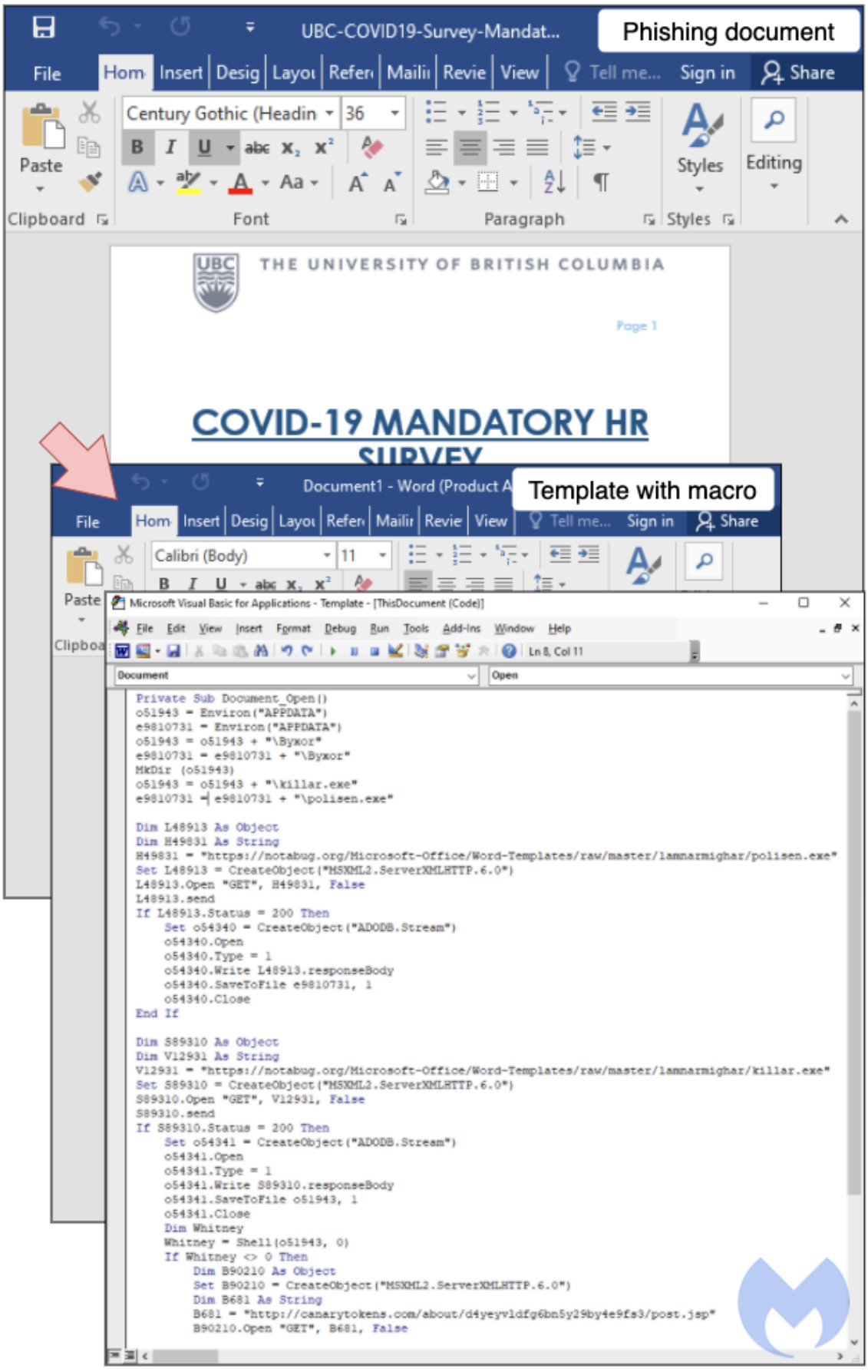
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

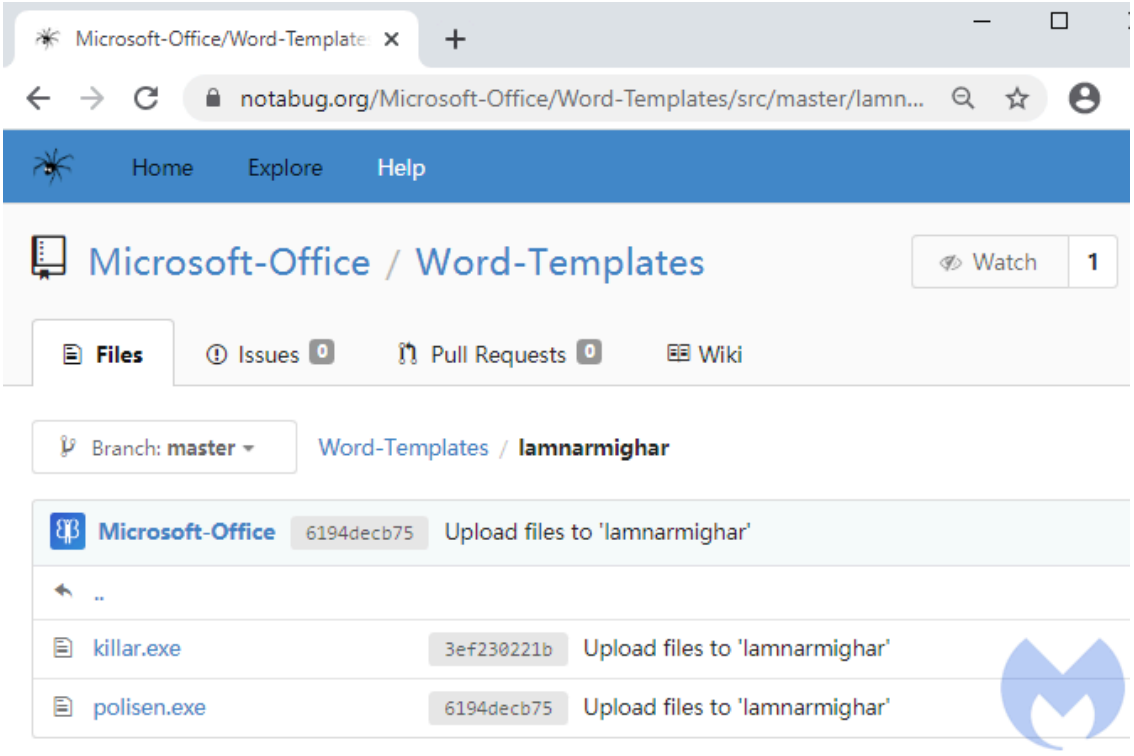
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



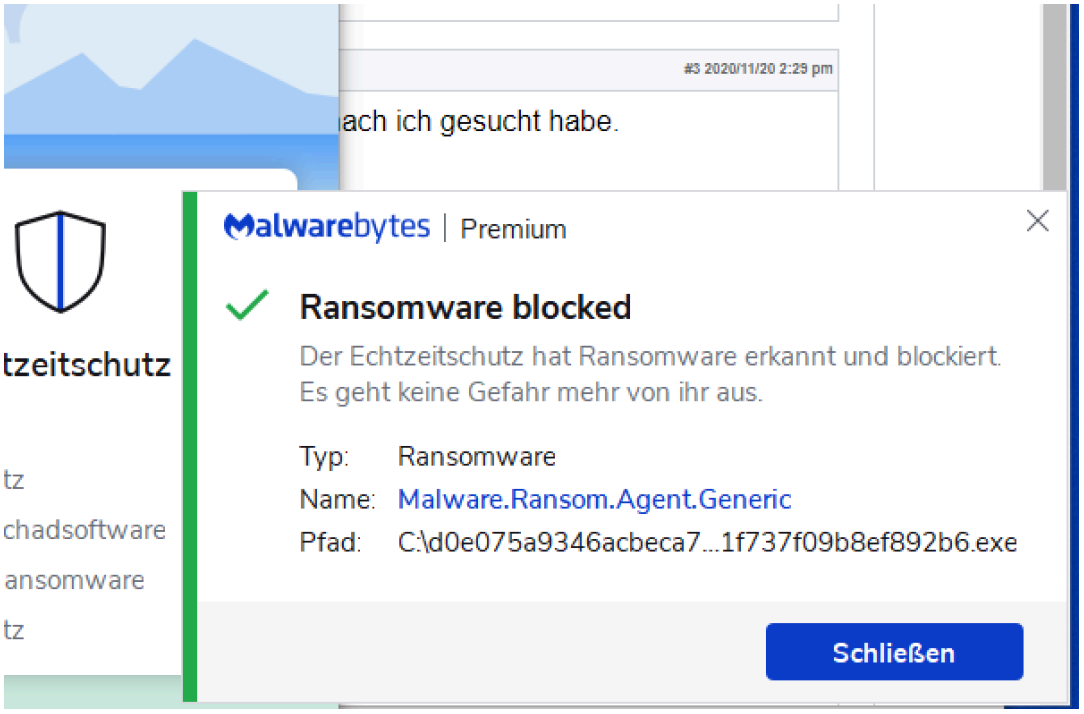
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

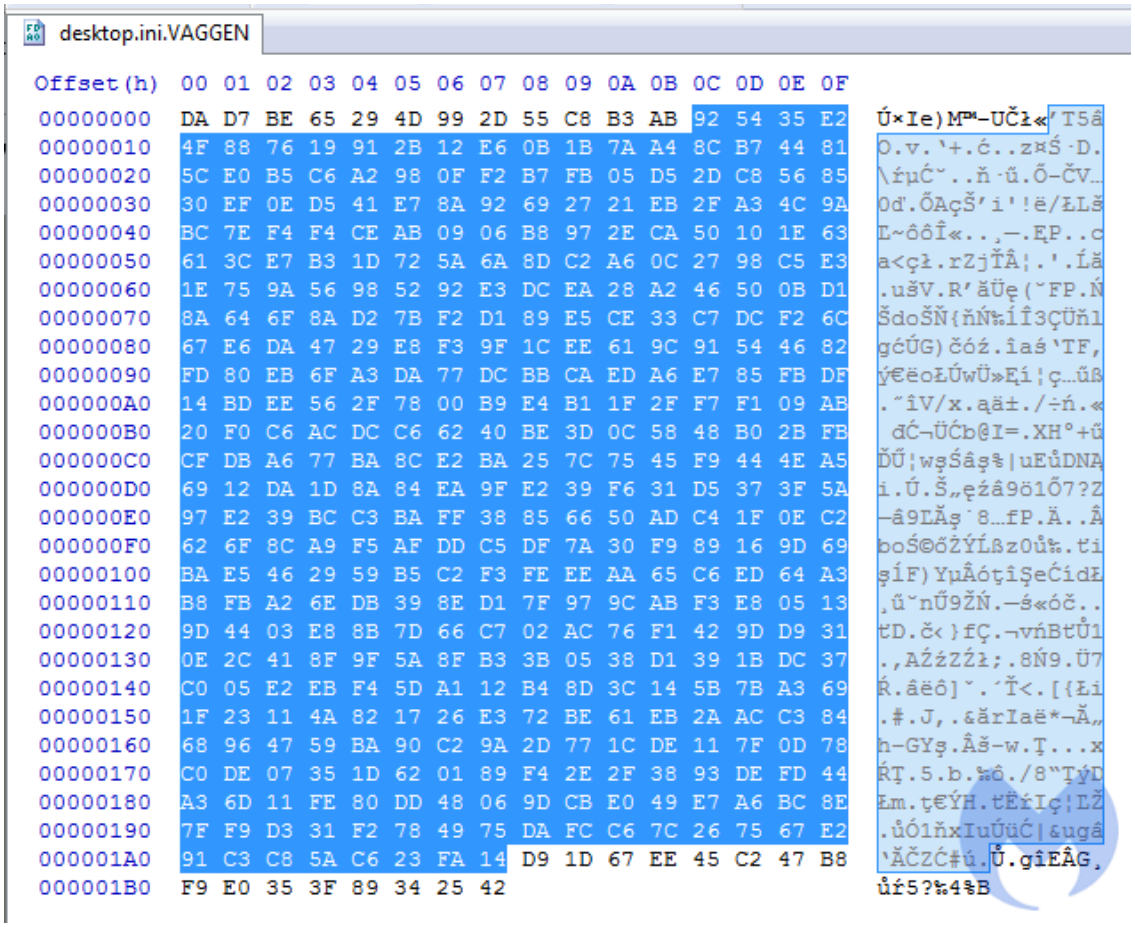
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



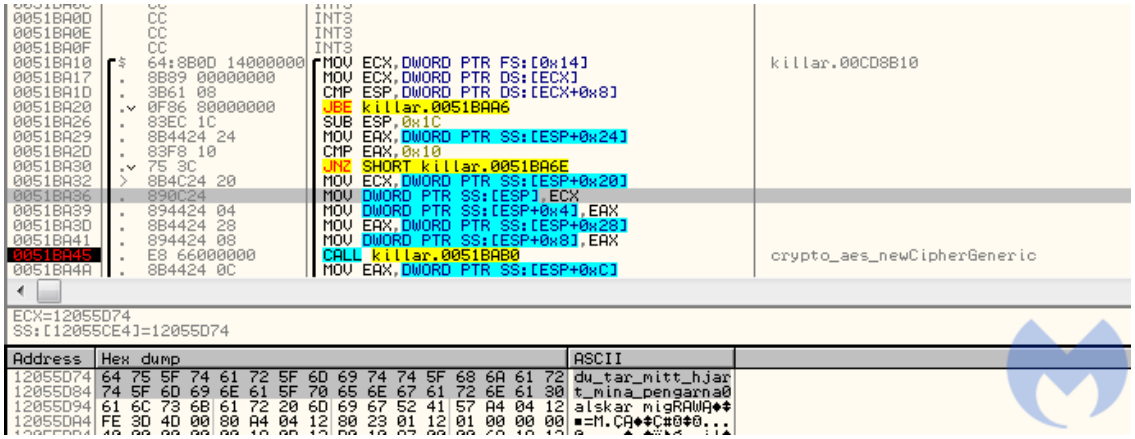
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

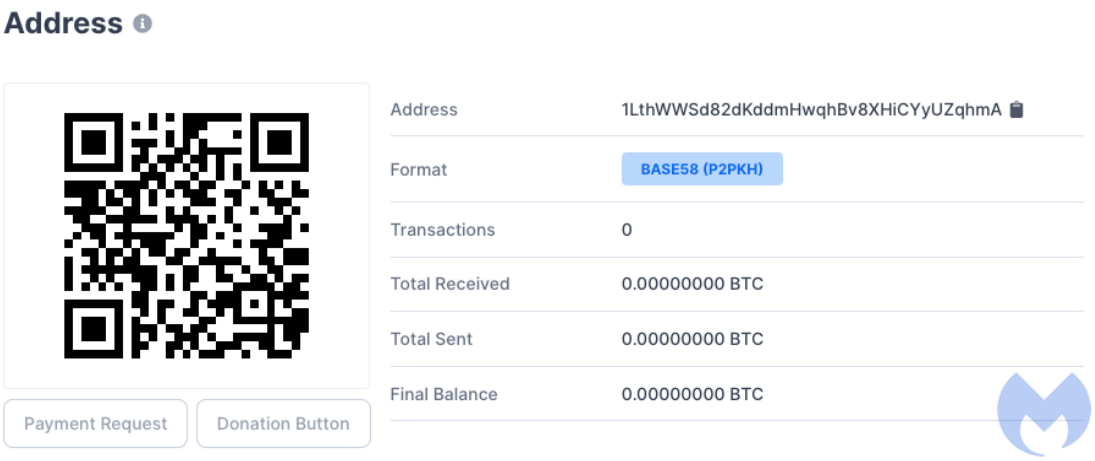
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

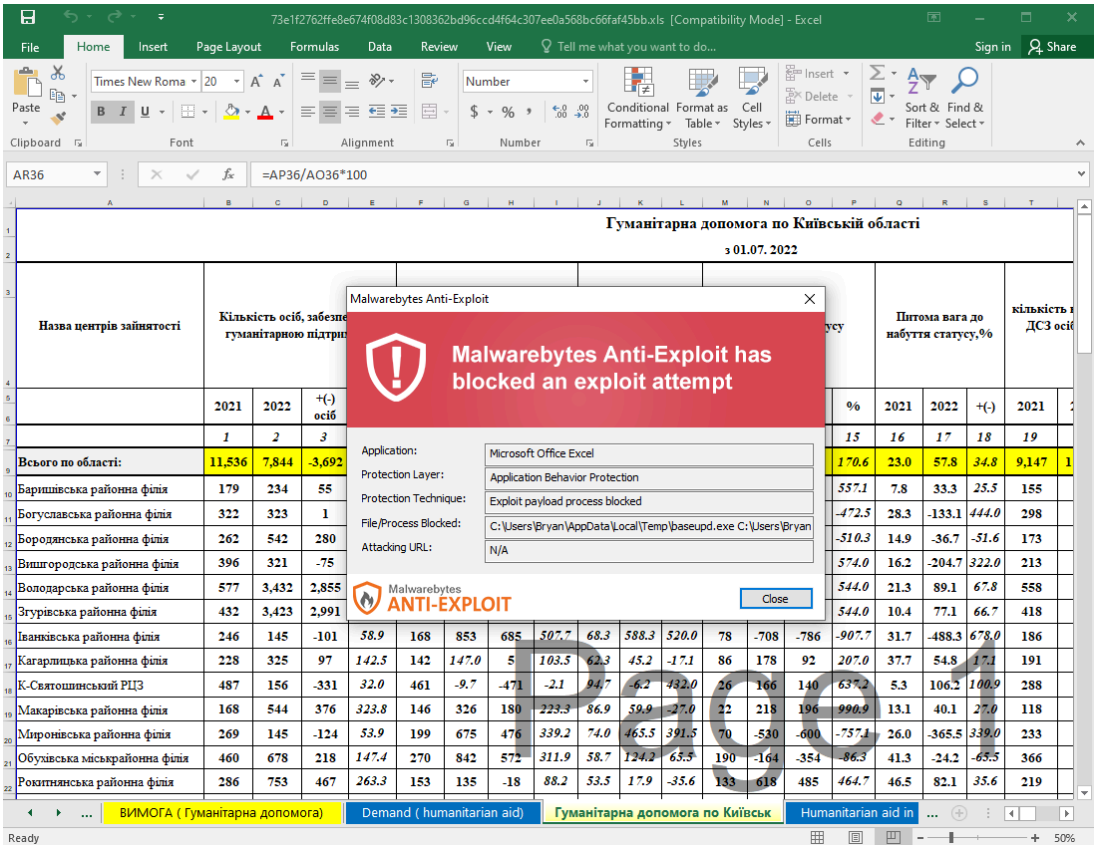
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

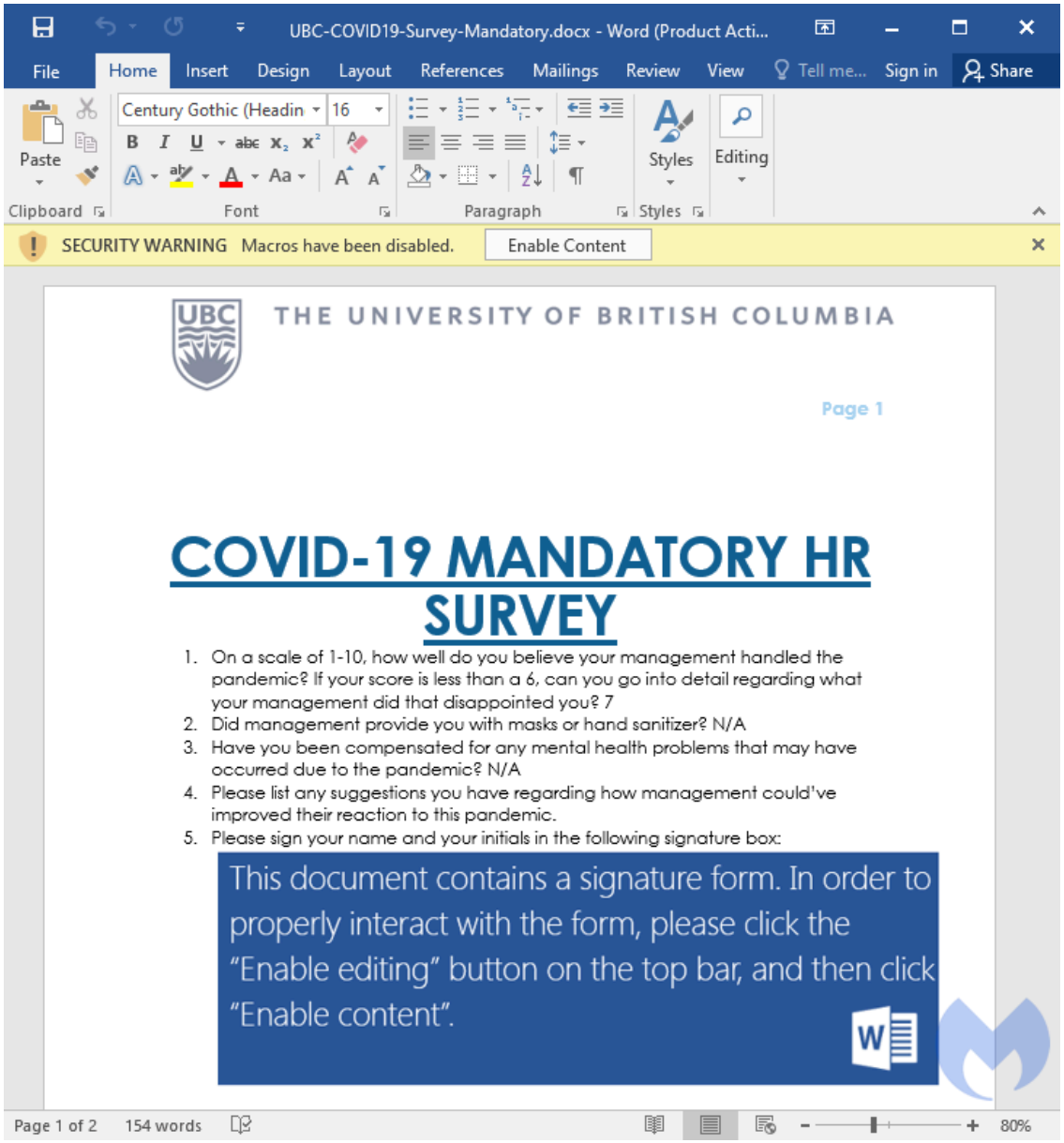
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

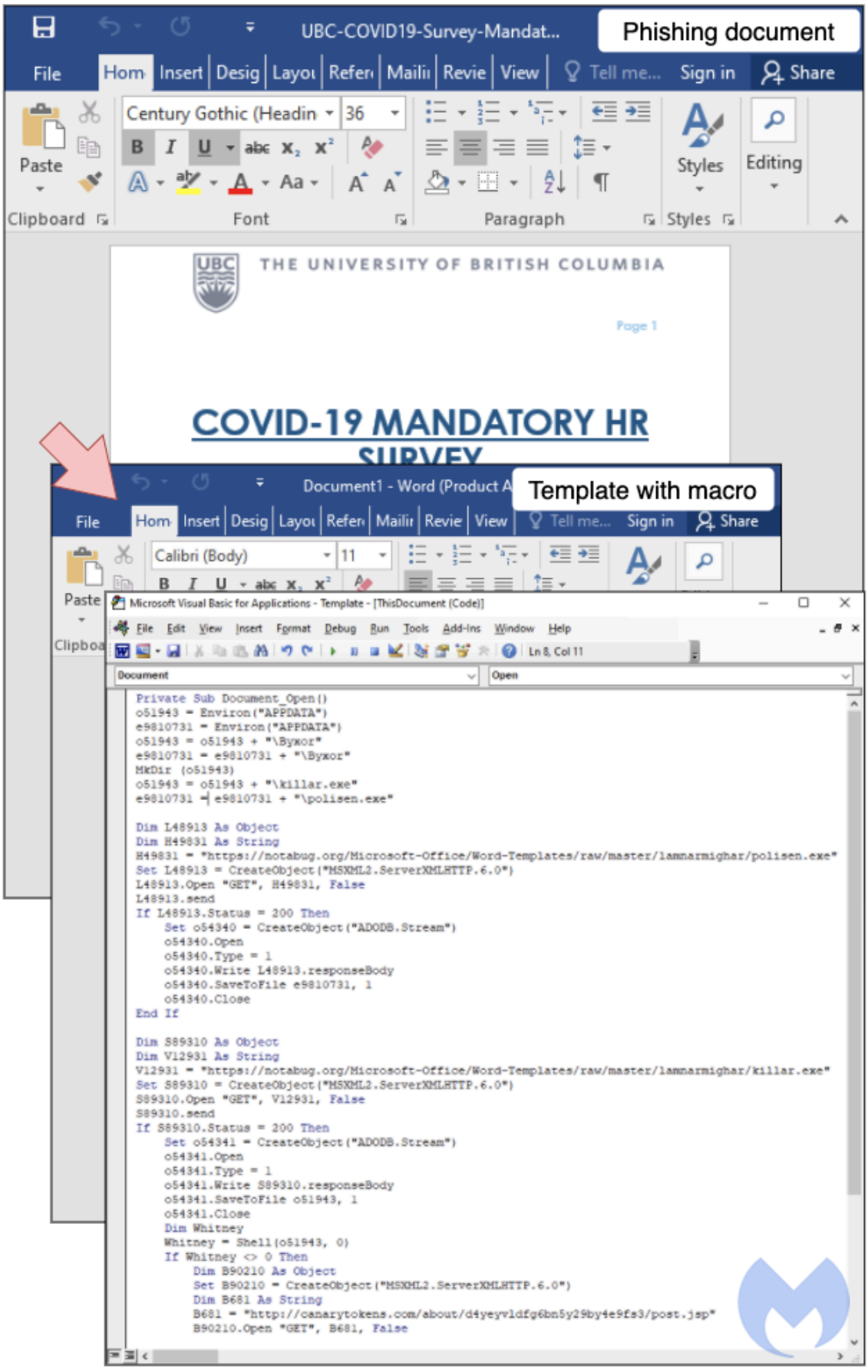
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

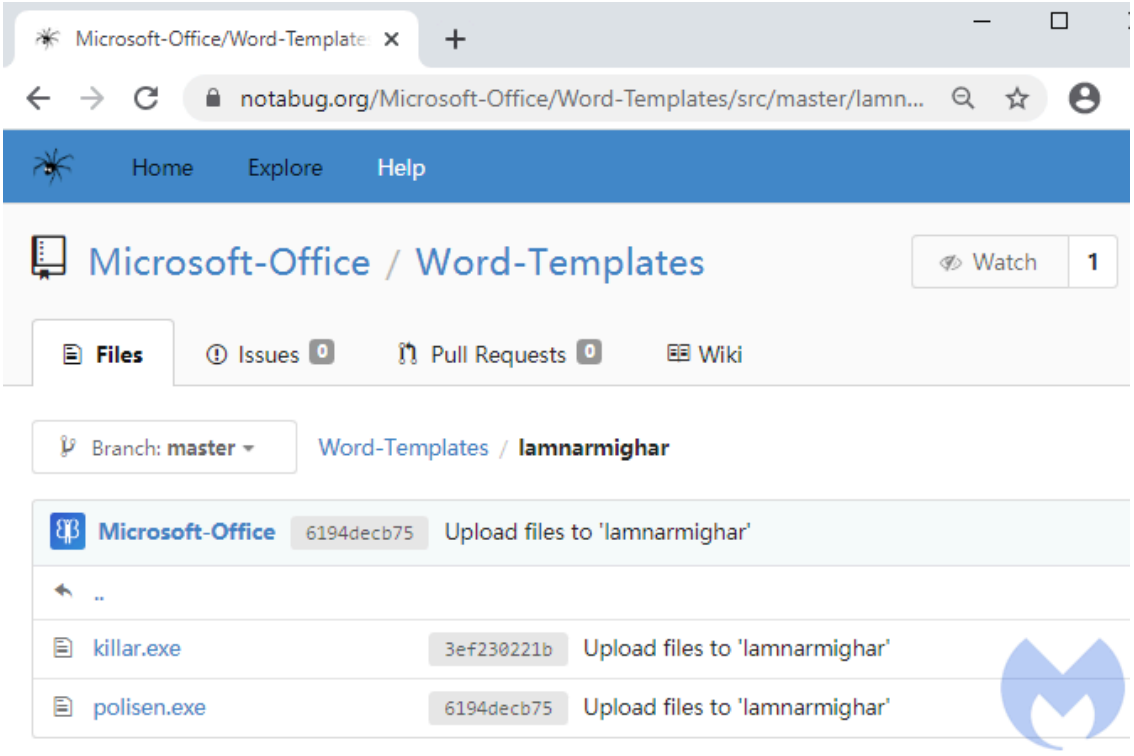
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



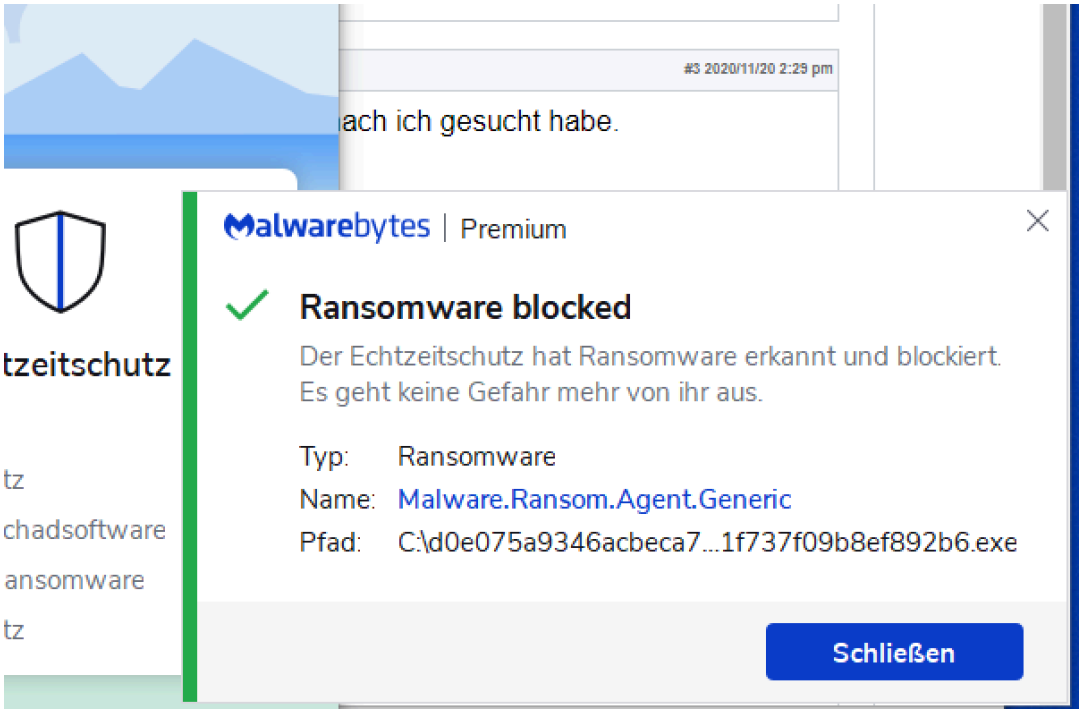
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

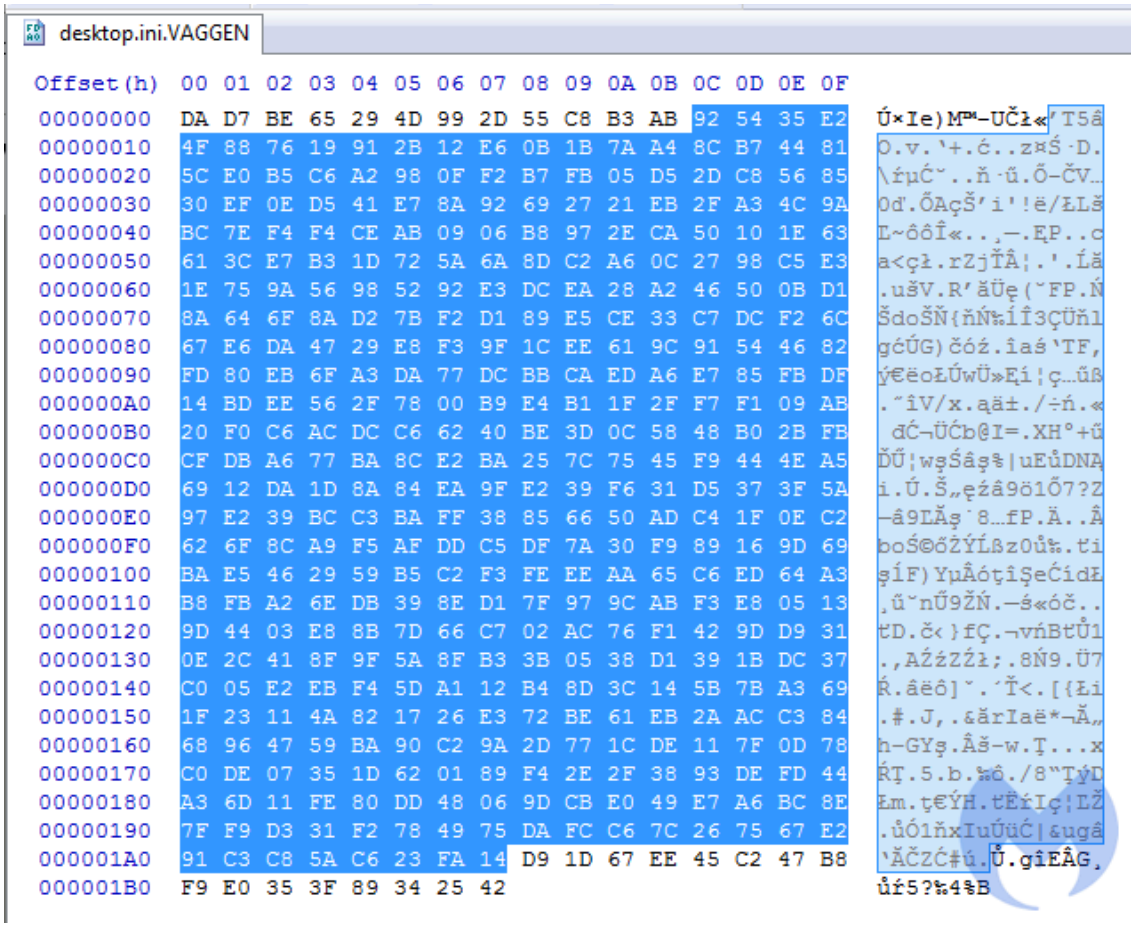
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

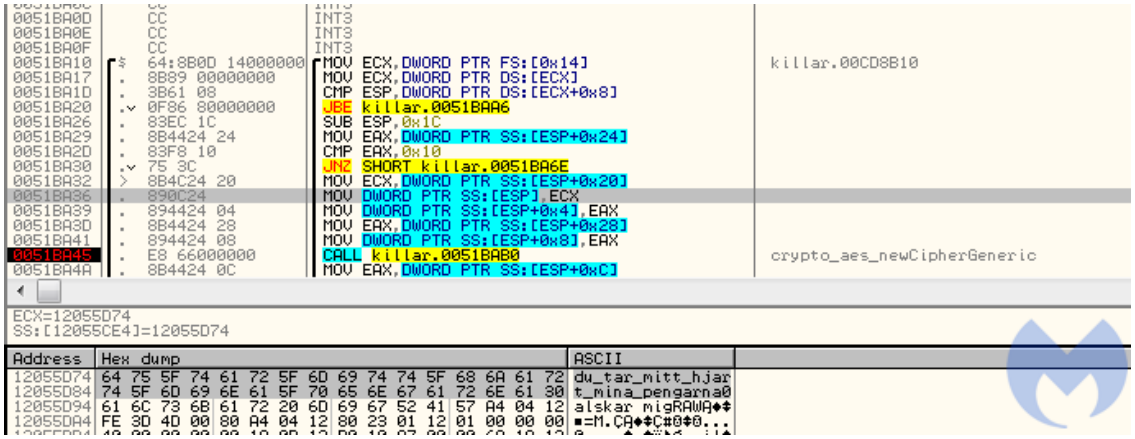


The content of the output file (with .VAGGEN extension) contains:

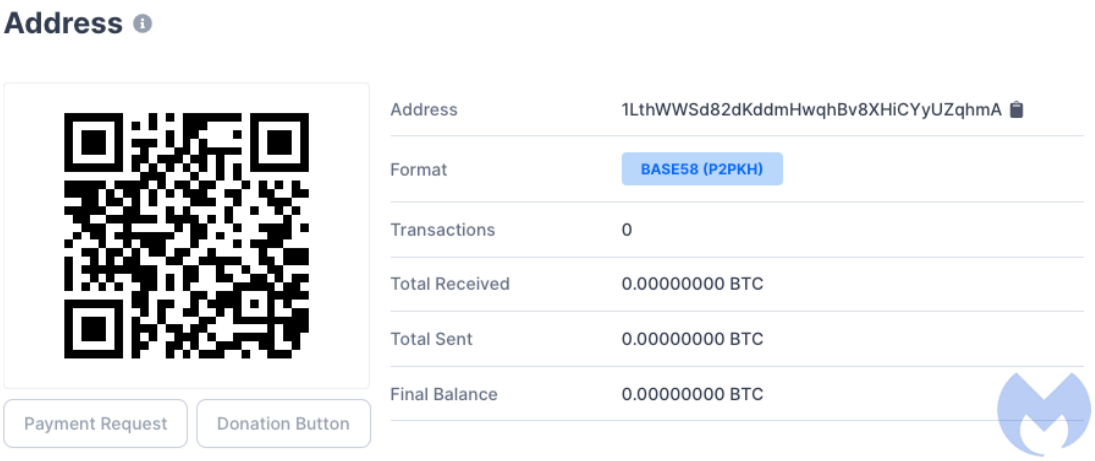
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

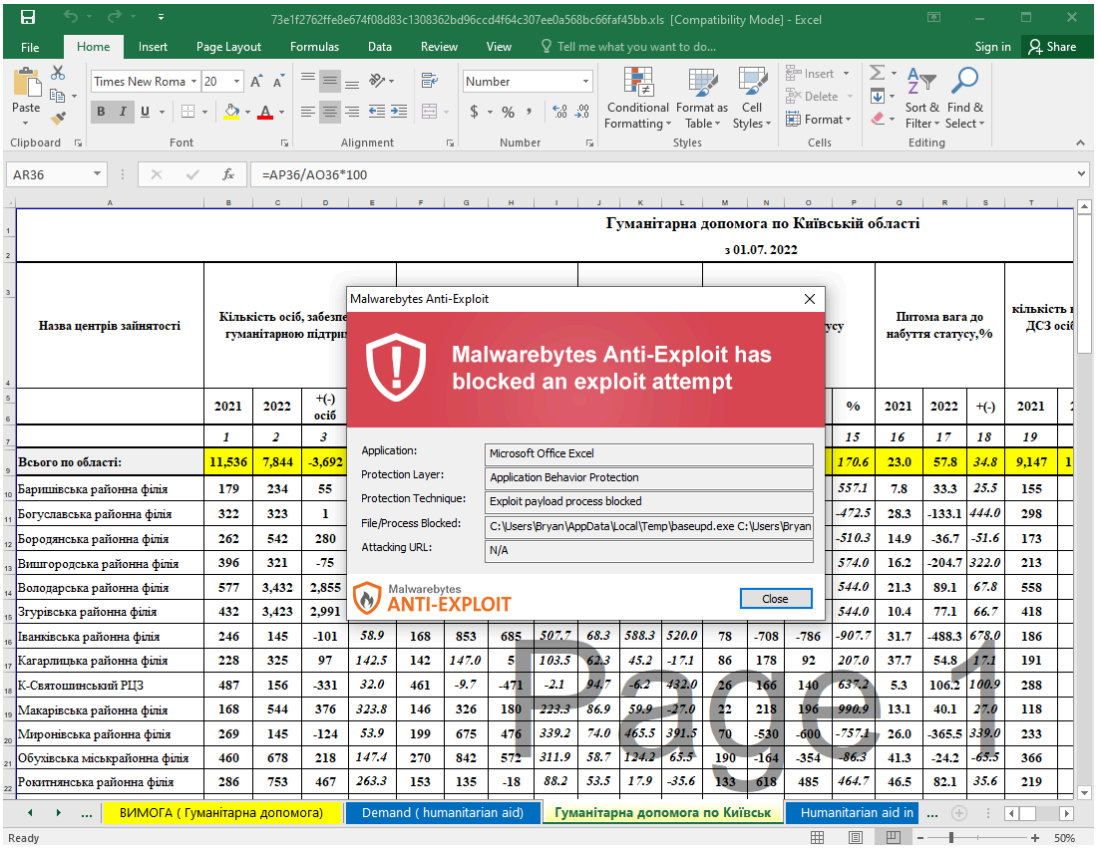
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

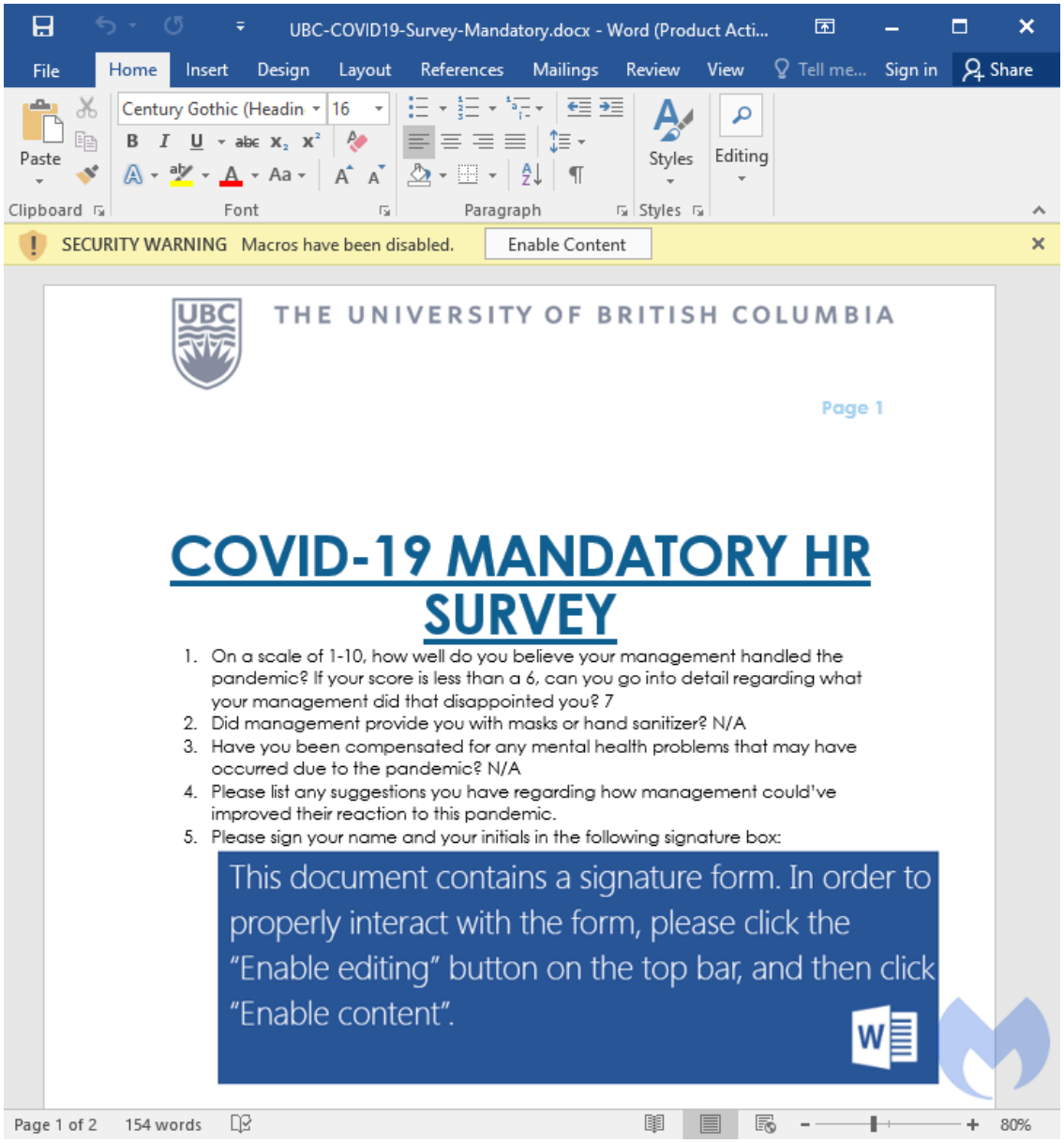
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

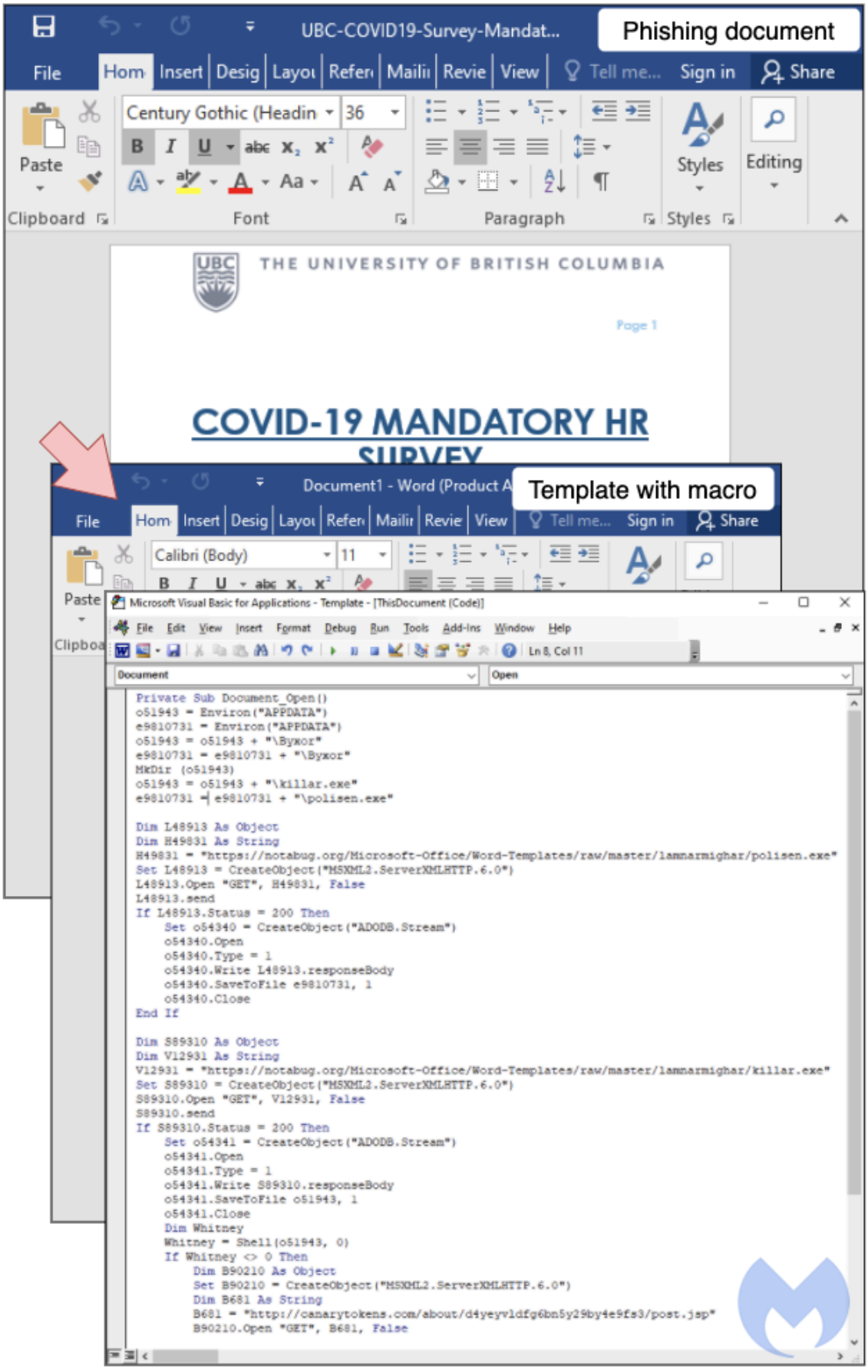
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

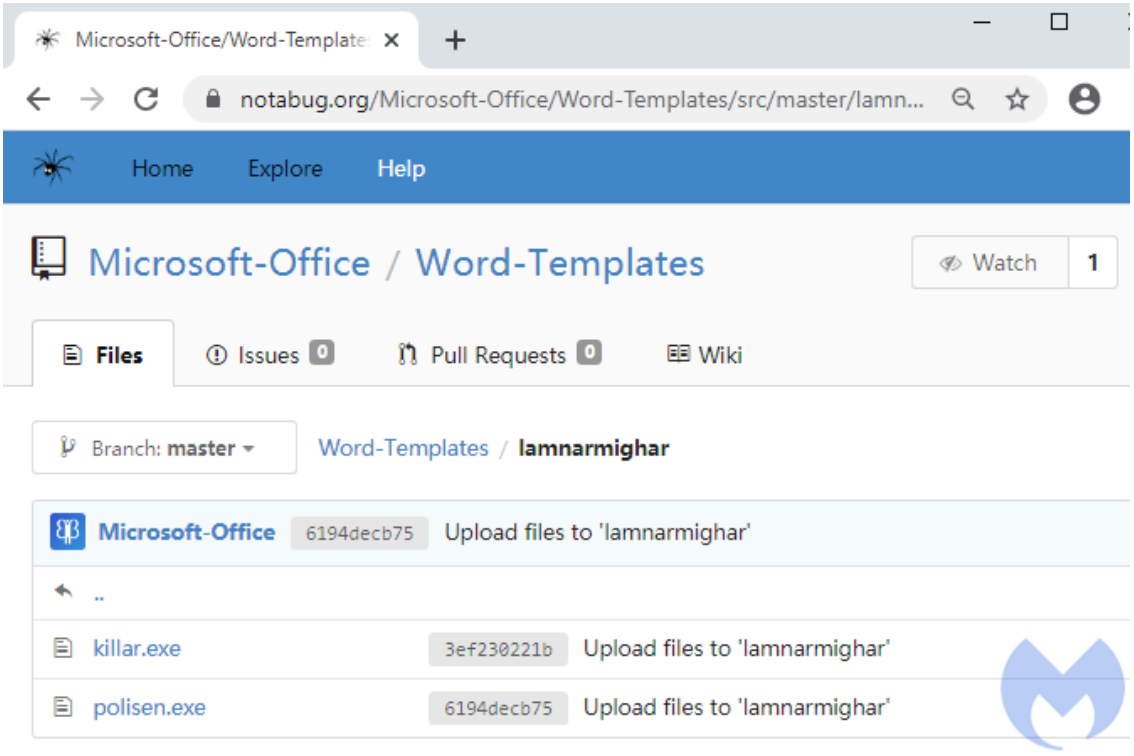
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



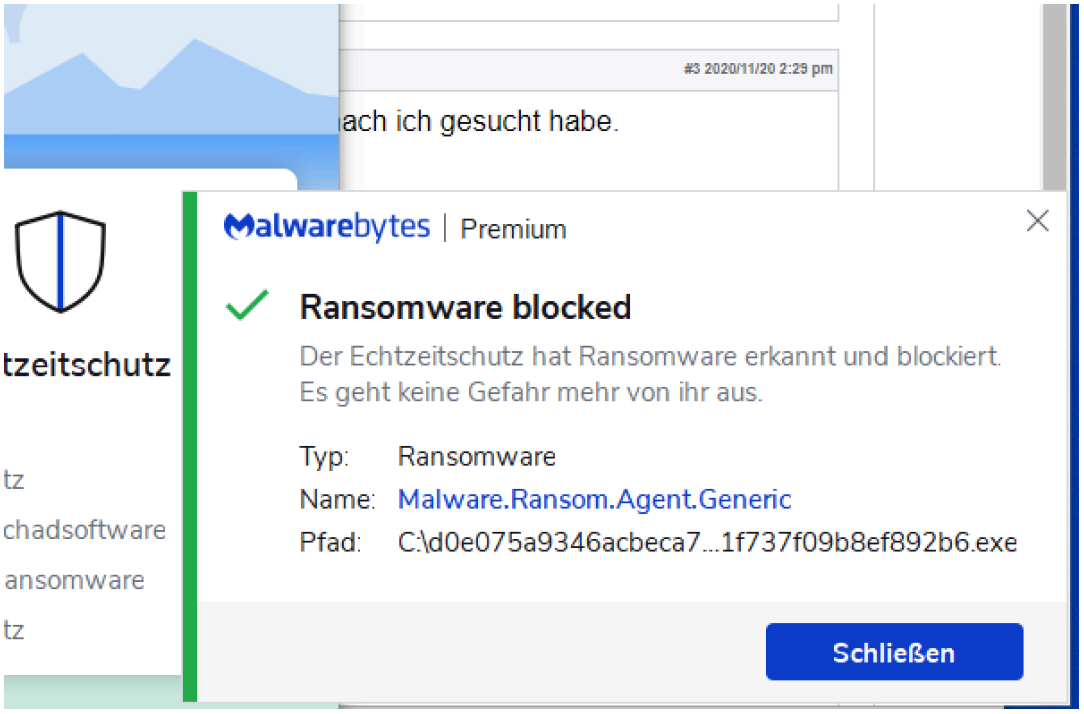
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

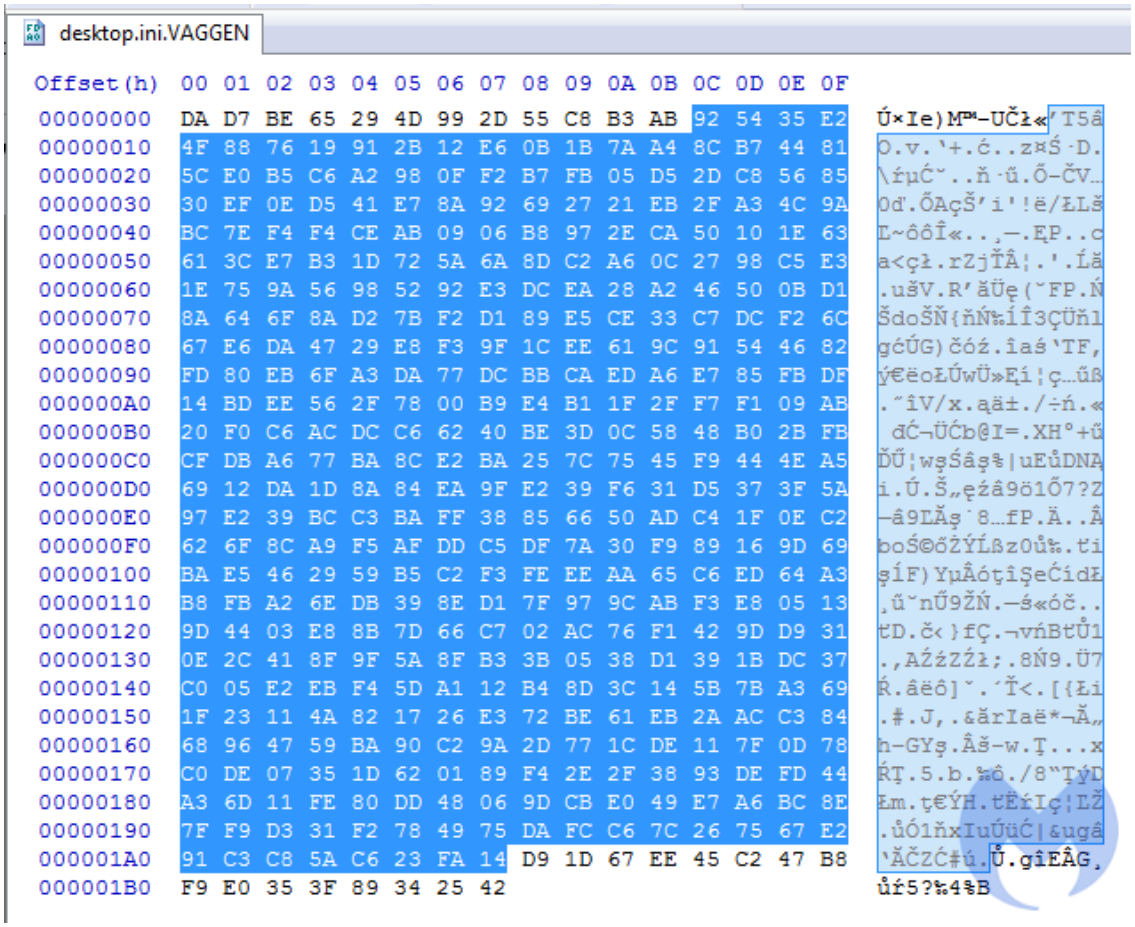
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

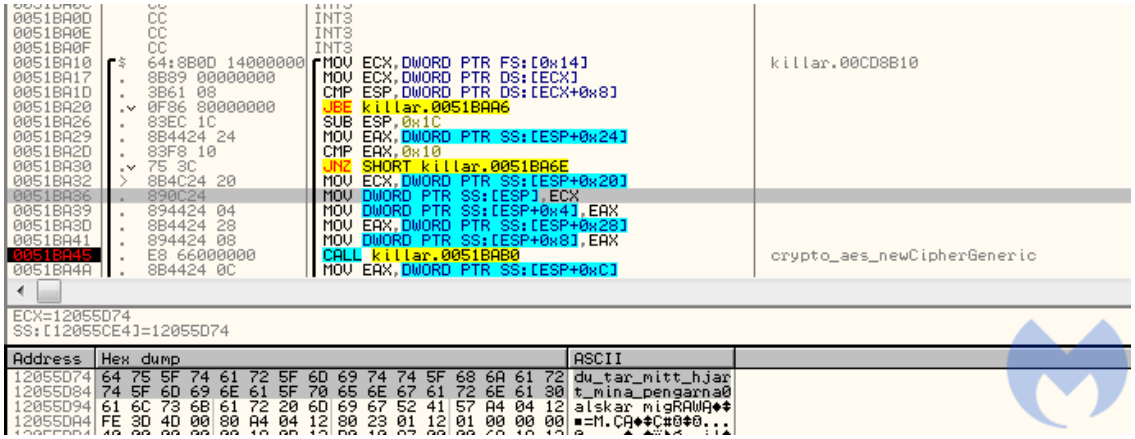


The content of the output file (with .VAGGEN extension) contains:

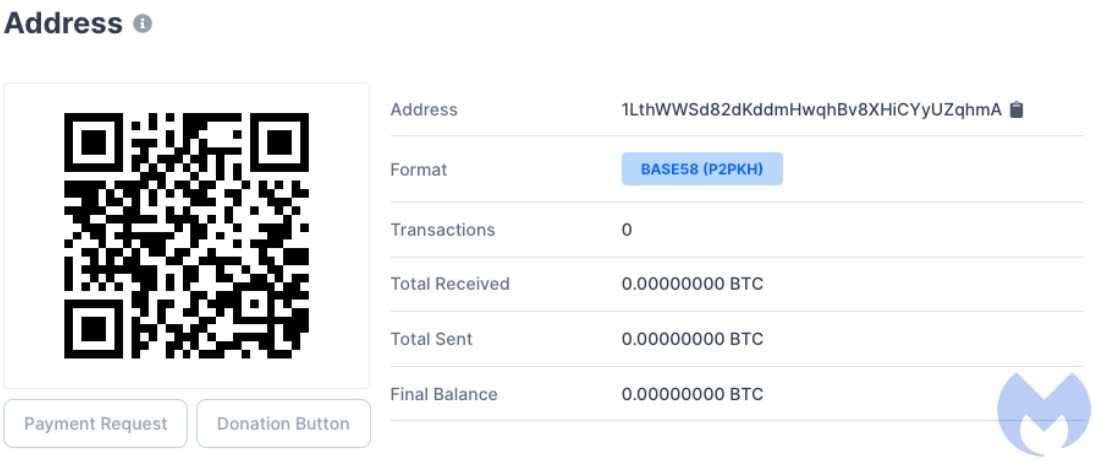
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

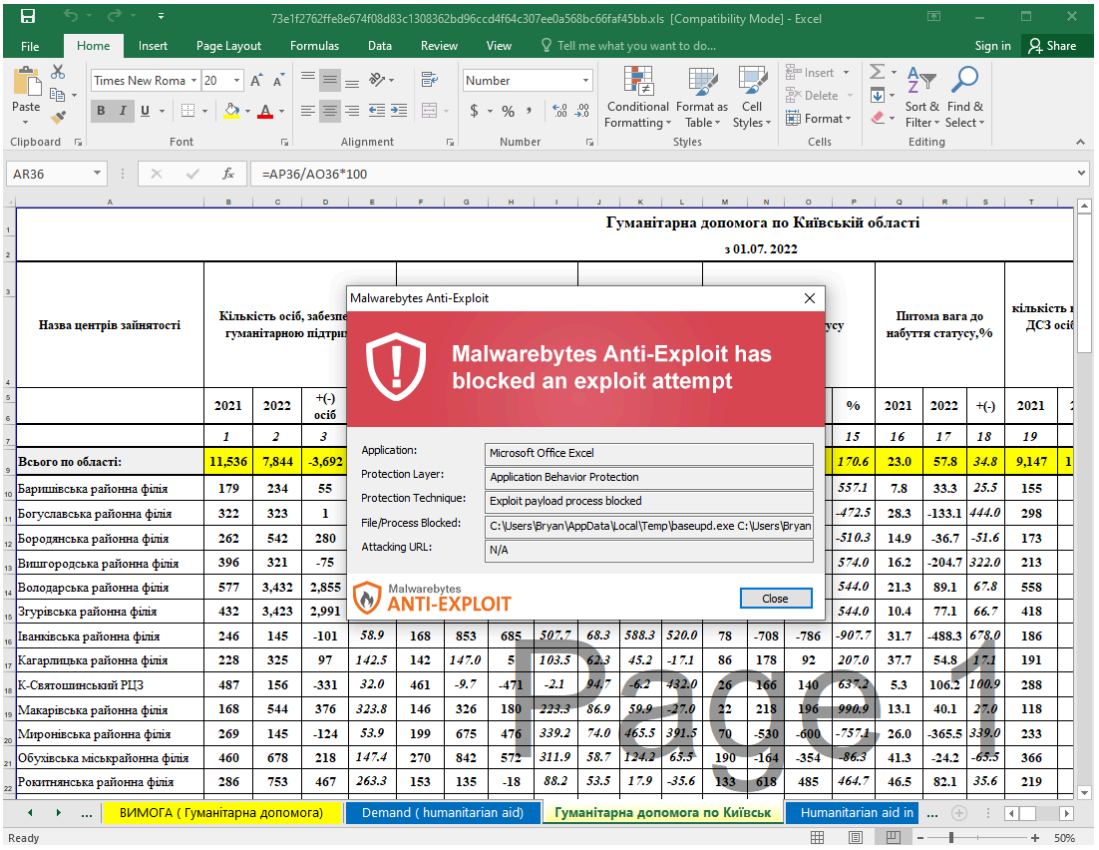
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

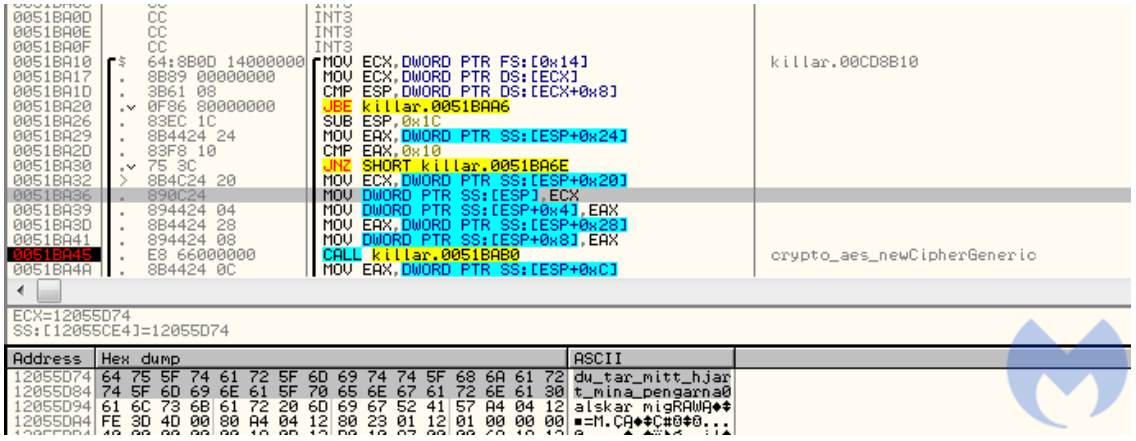
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

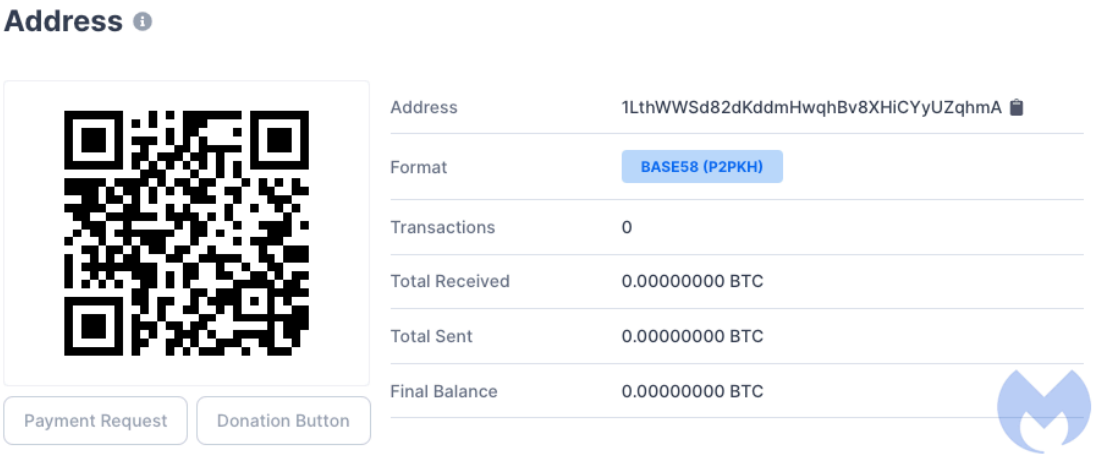
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

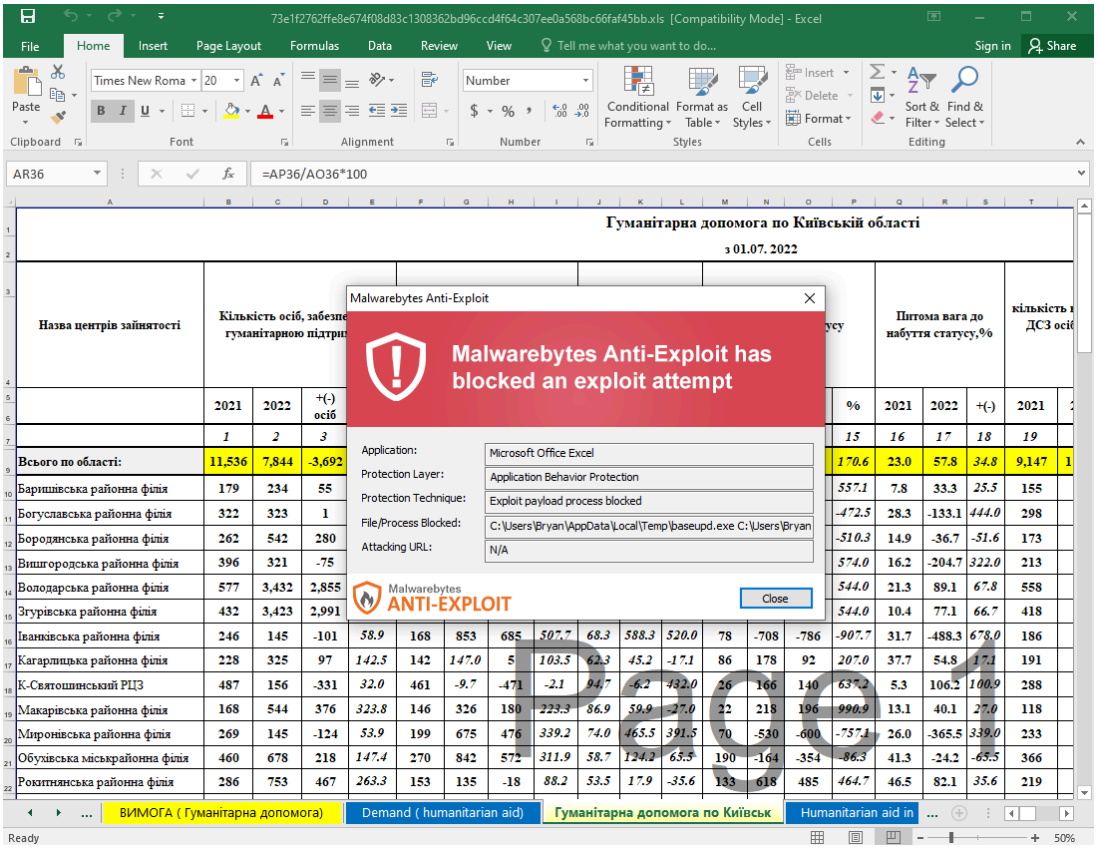
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

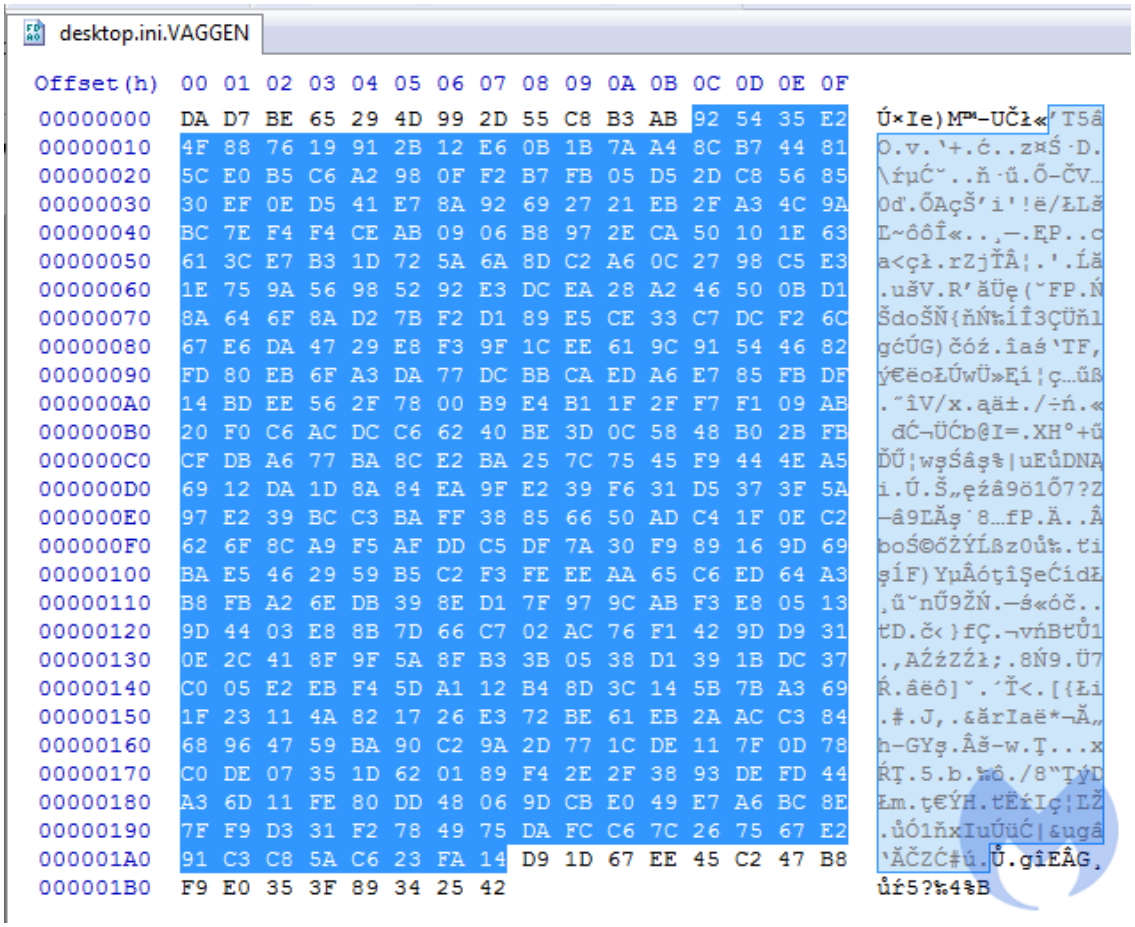
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



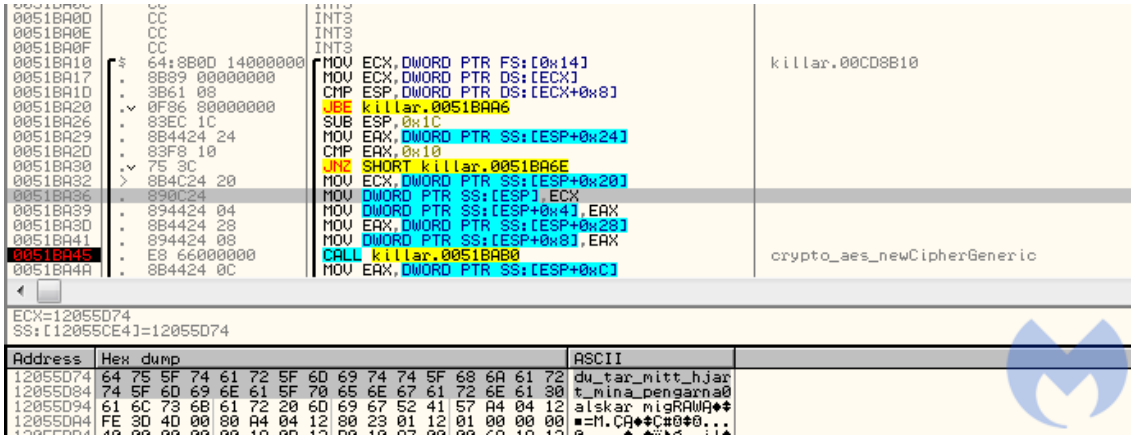
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

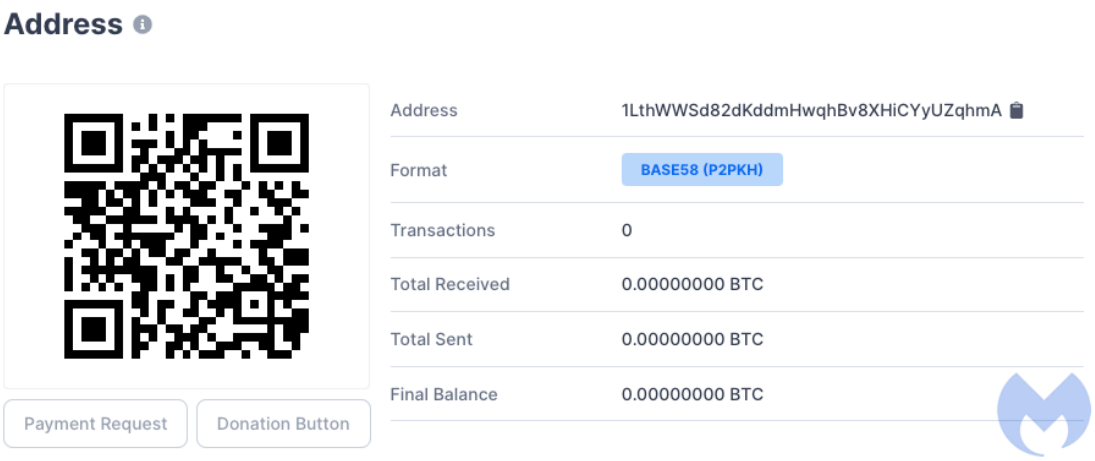
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

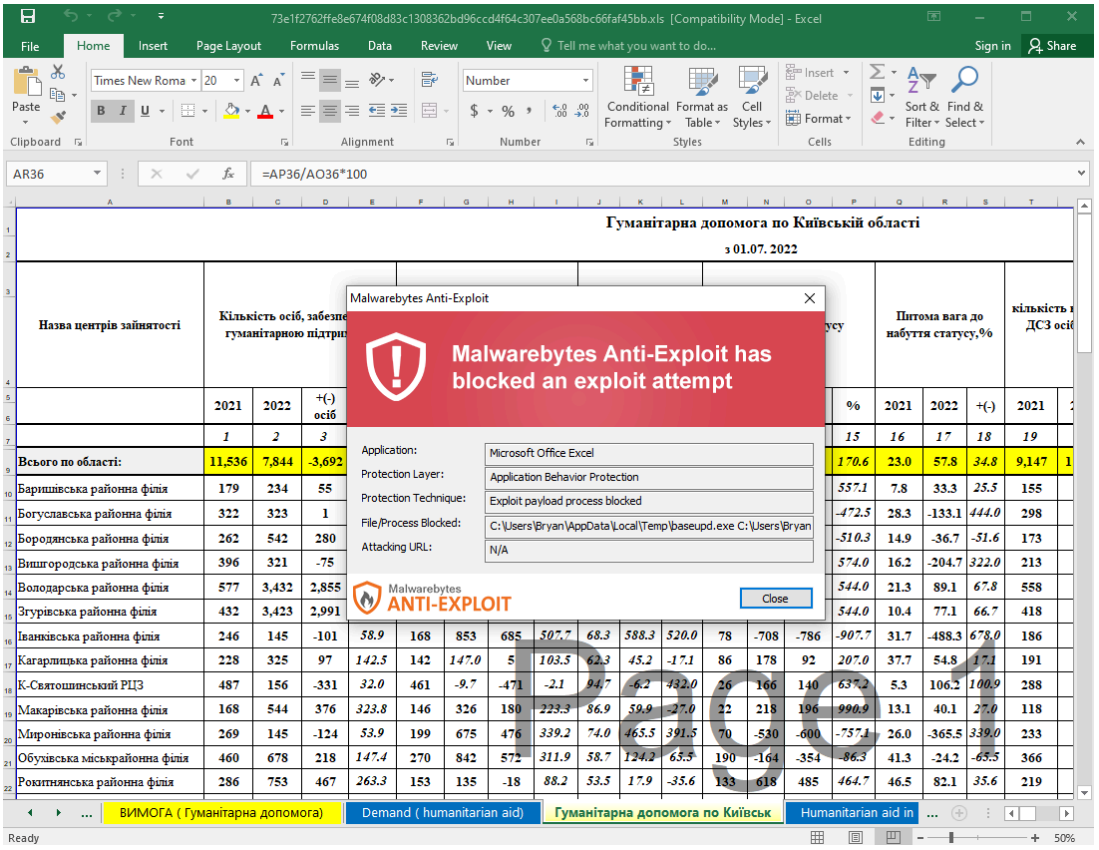
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

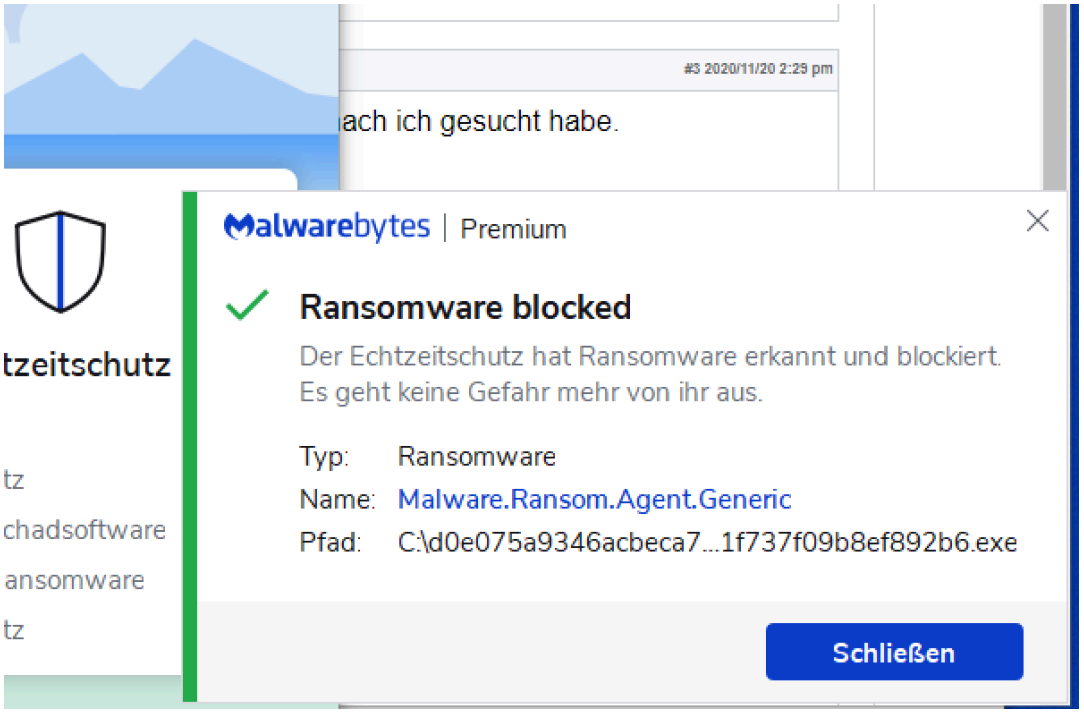
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

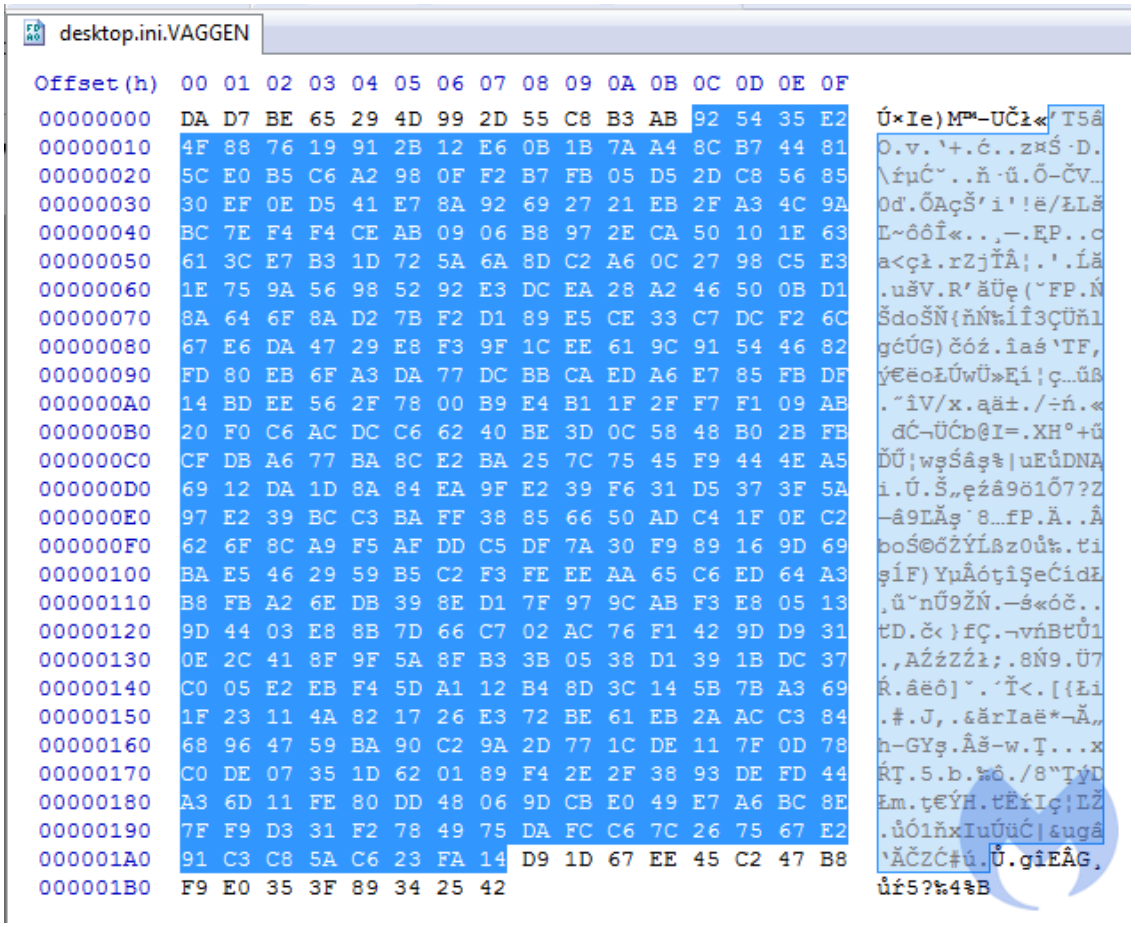
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

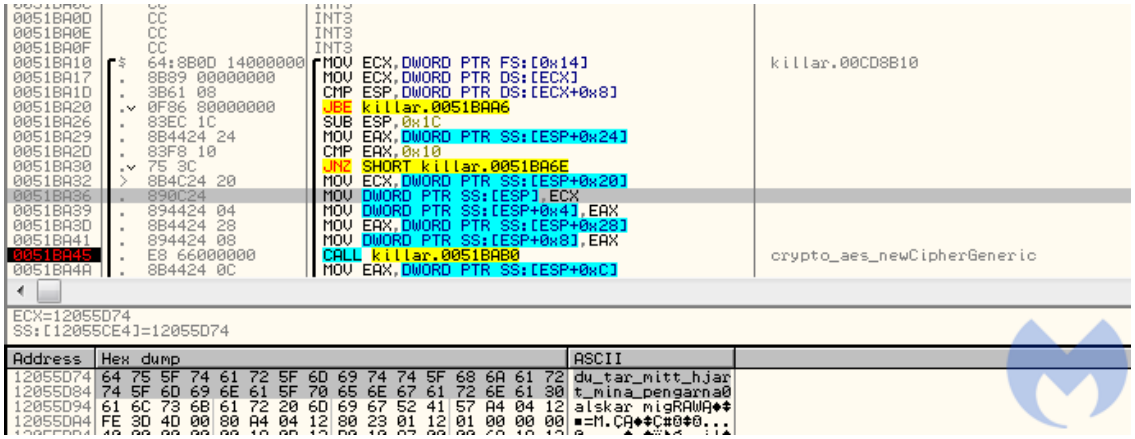


The content of the output file (with .VAGGEN extension) contains:

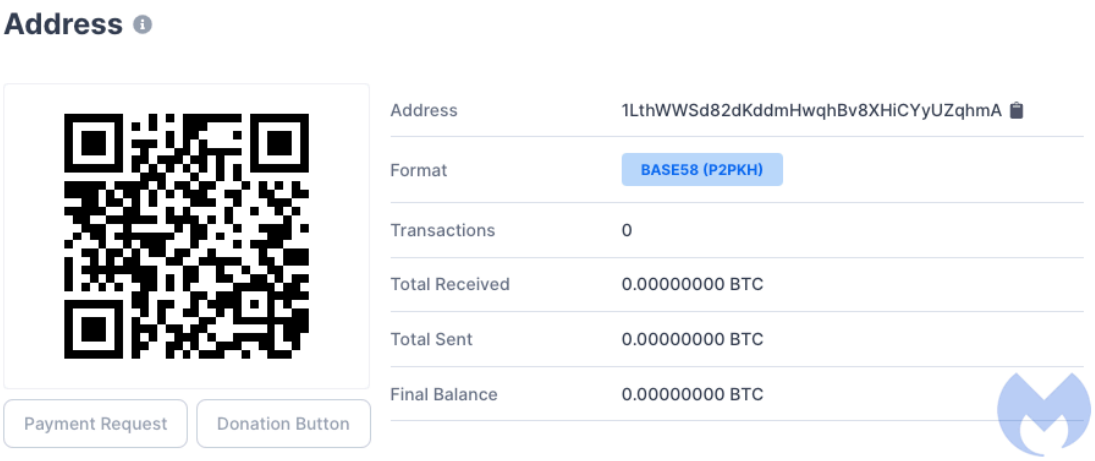
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

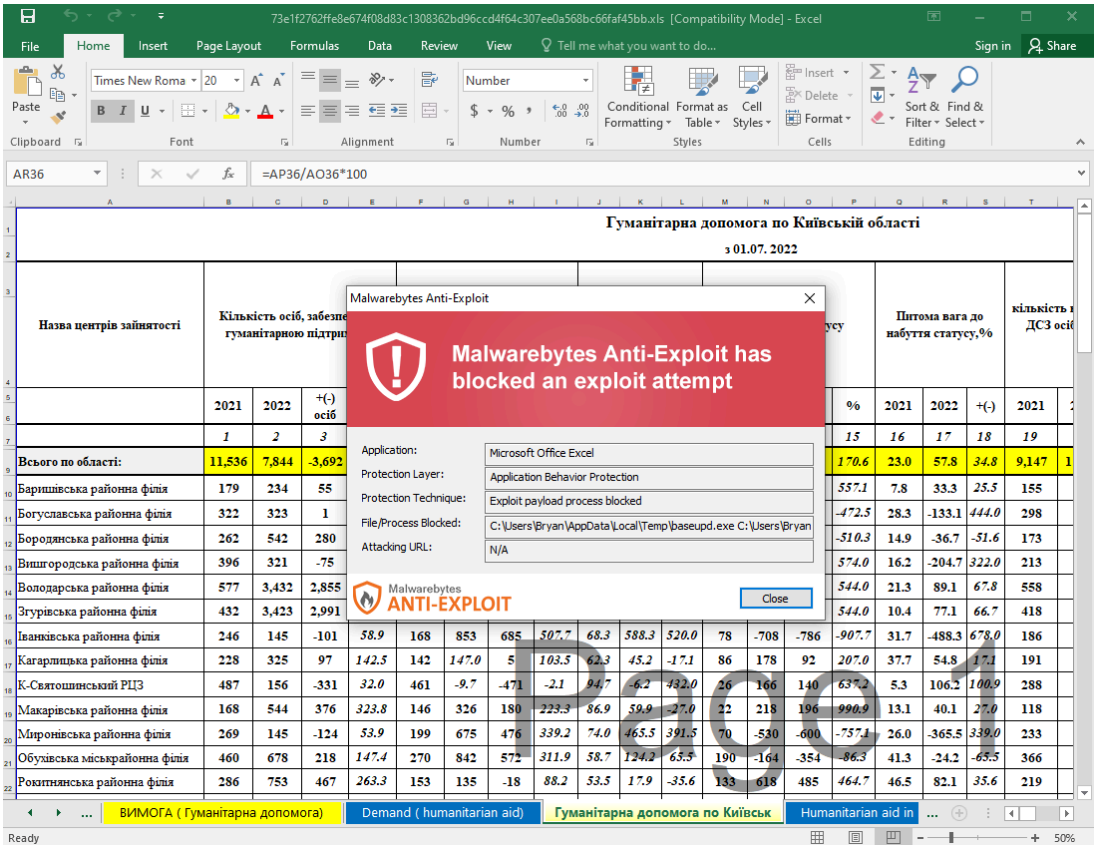
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

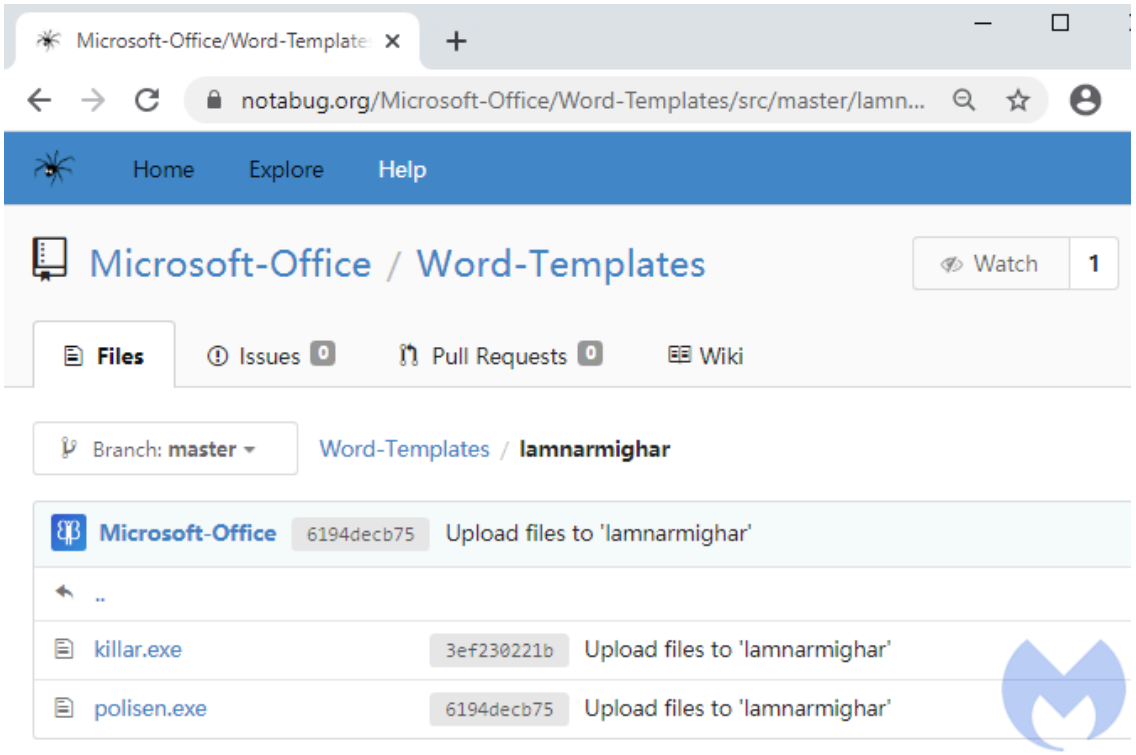
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



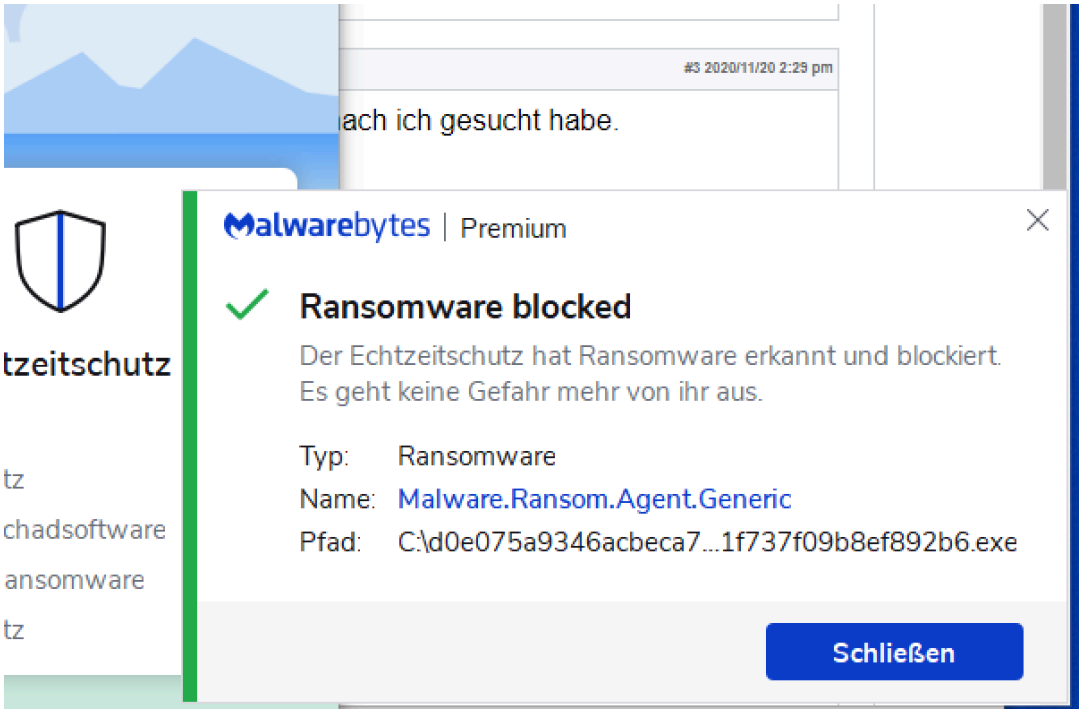
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

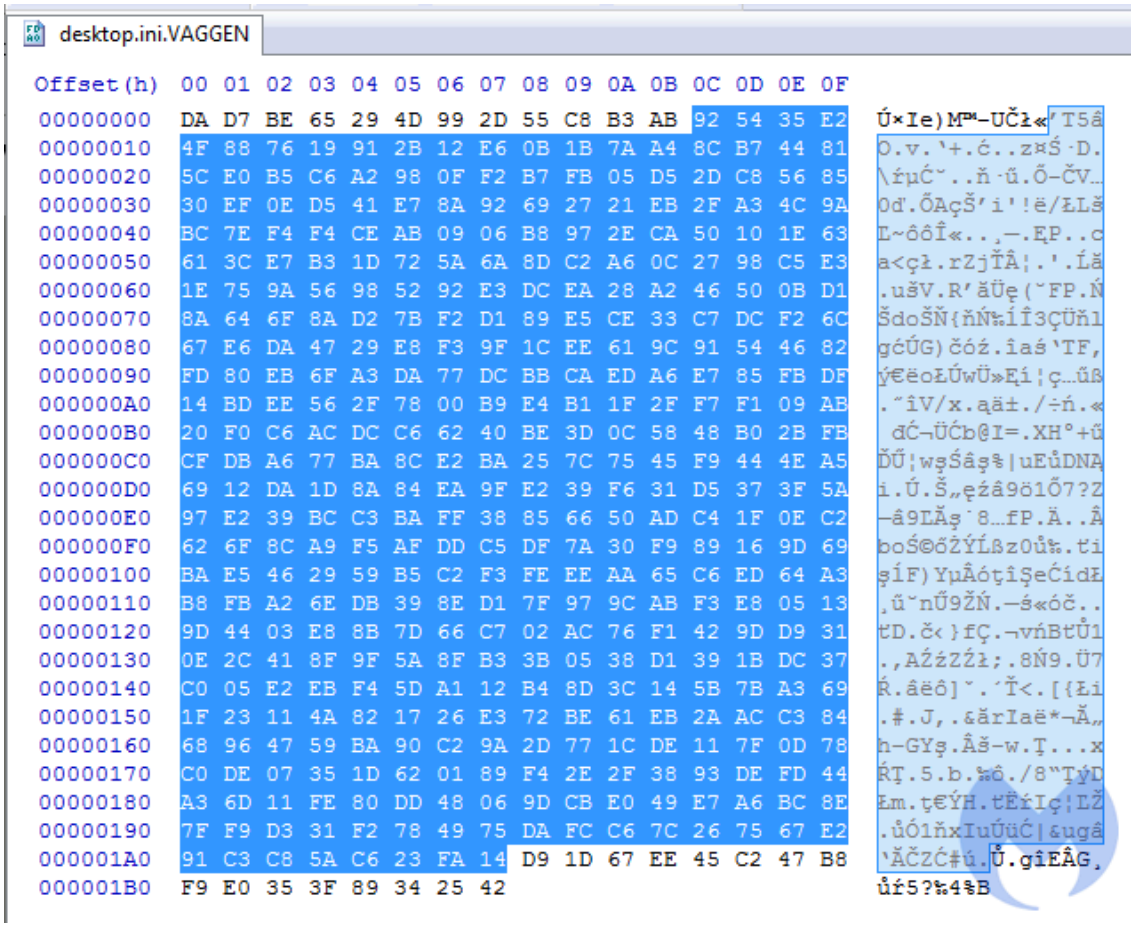
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



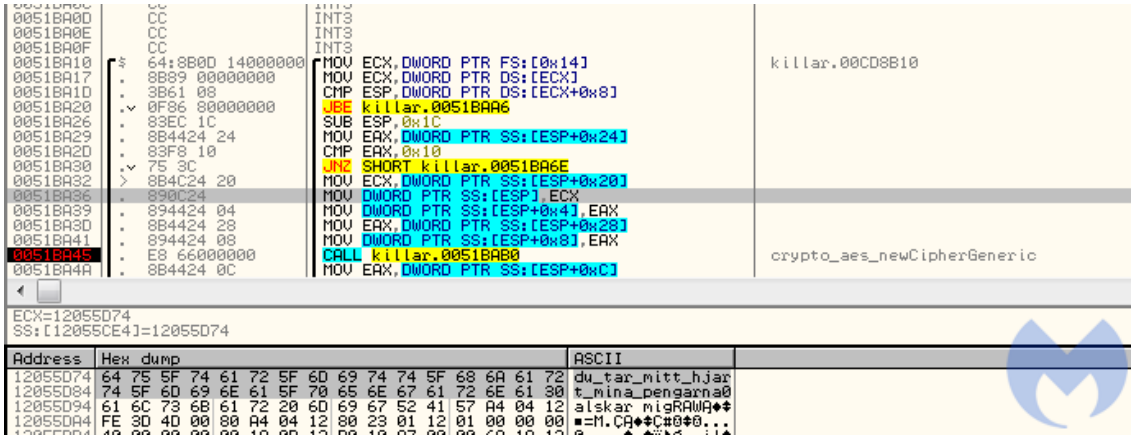
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

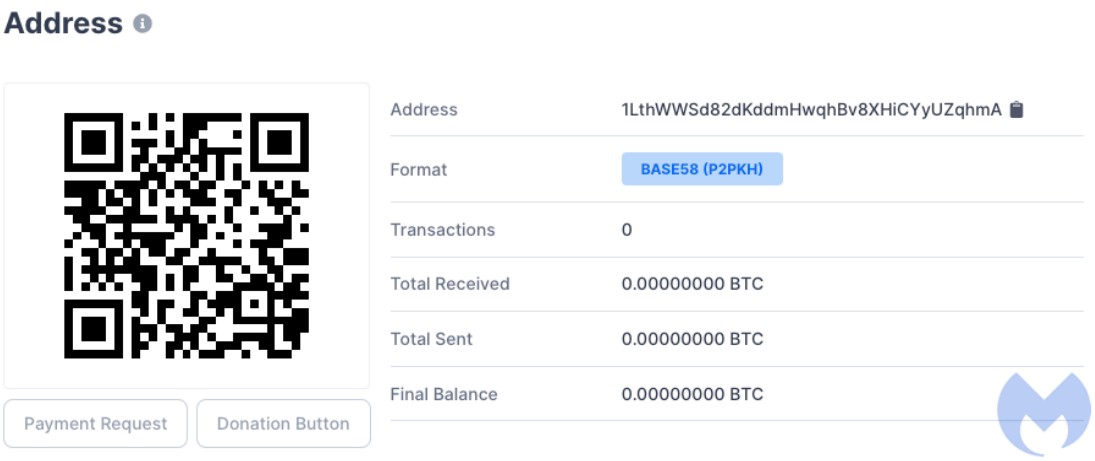
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

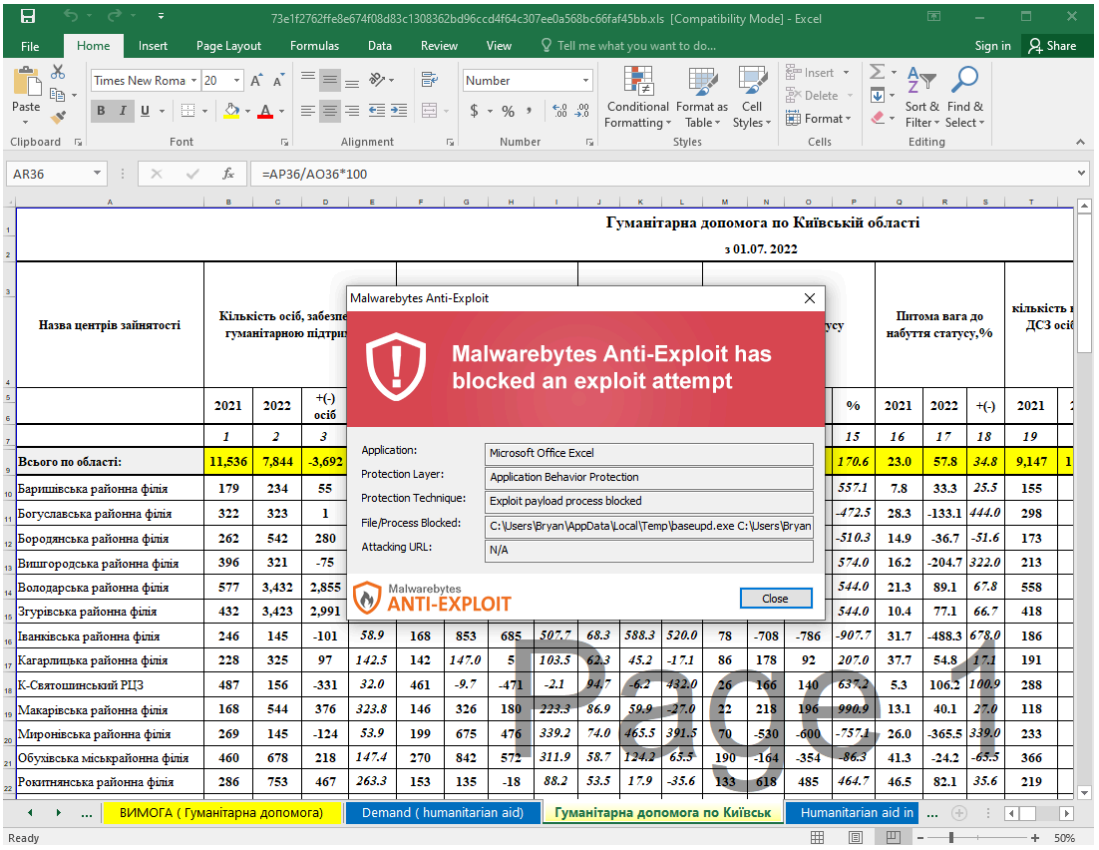
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

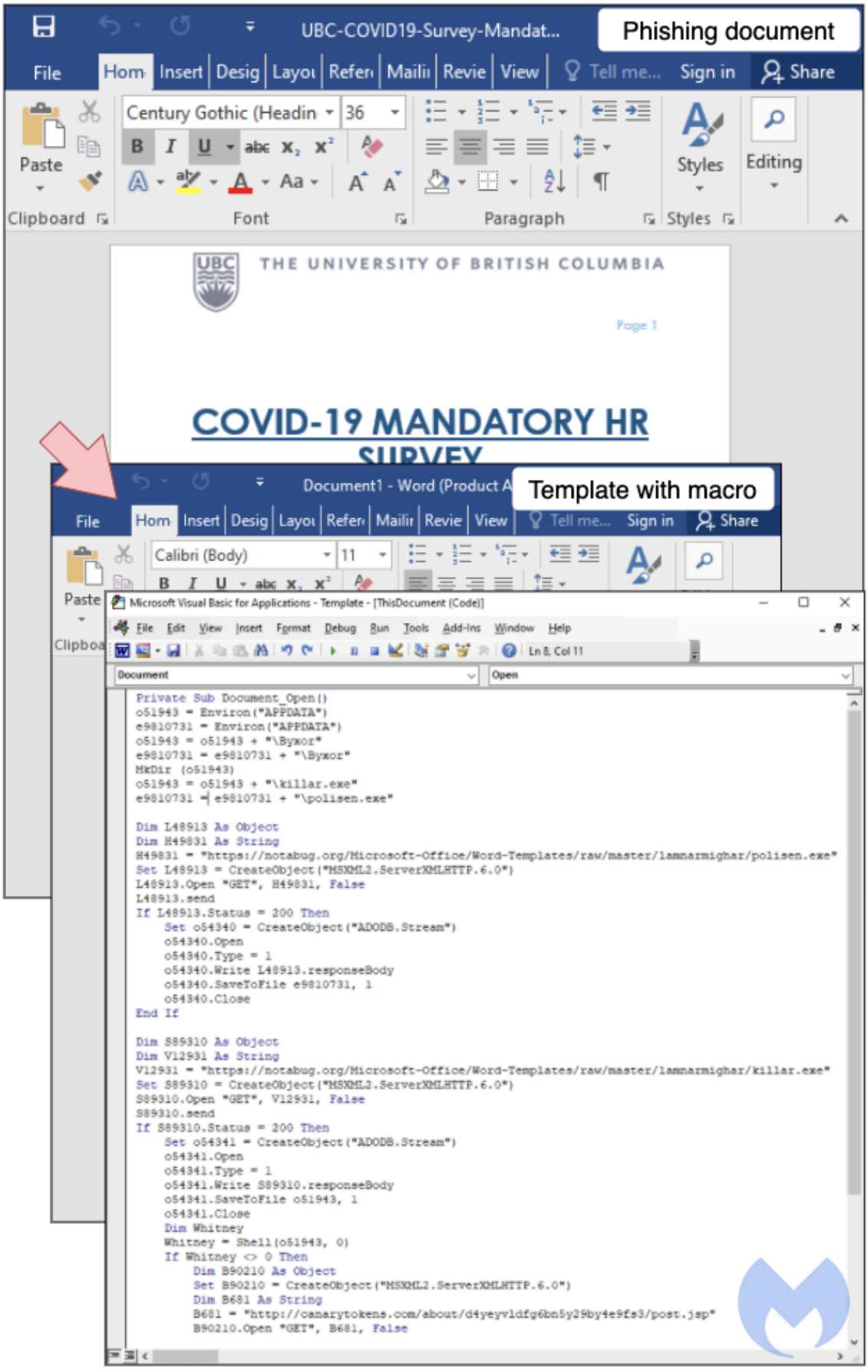
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

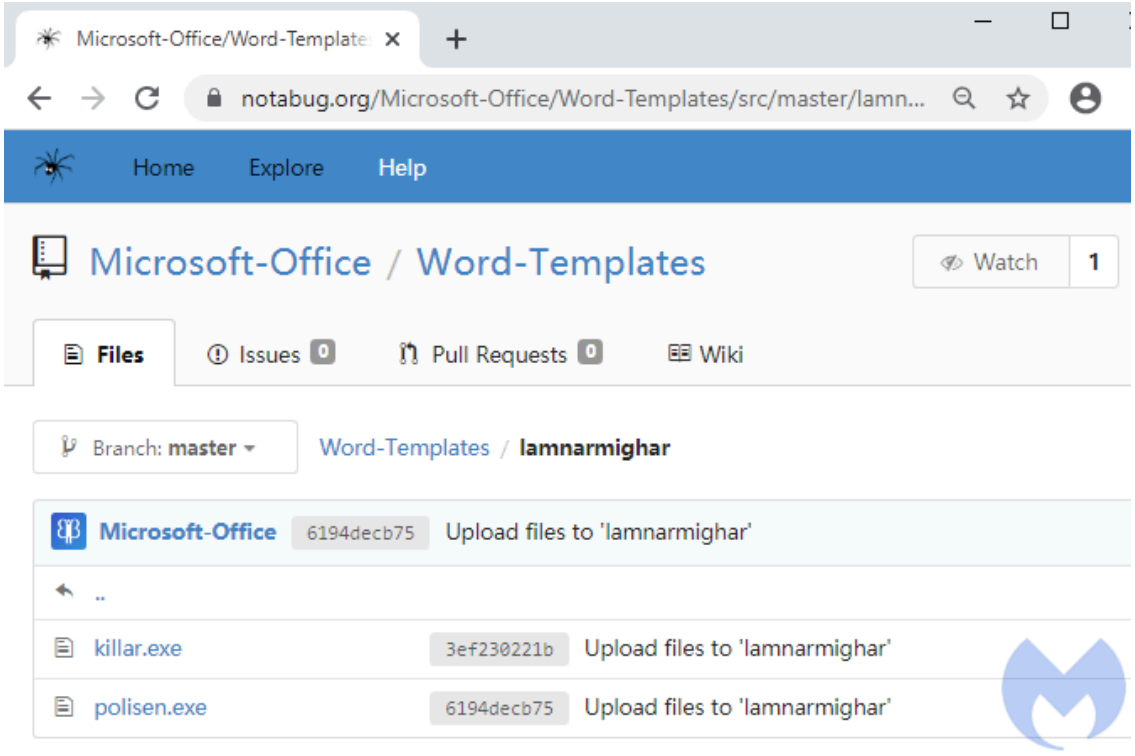
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



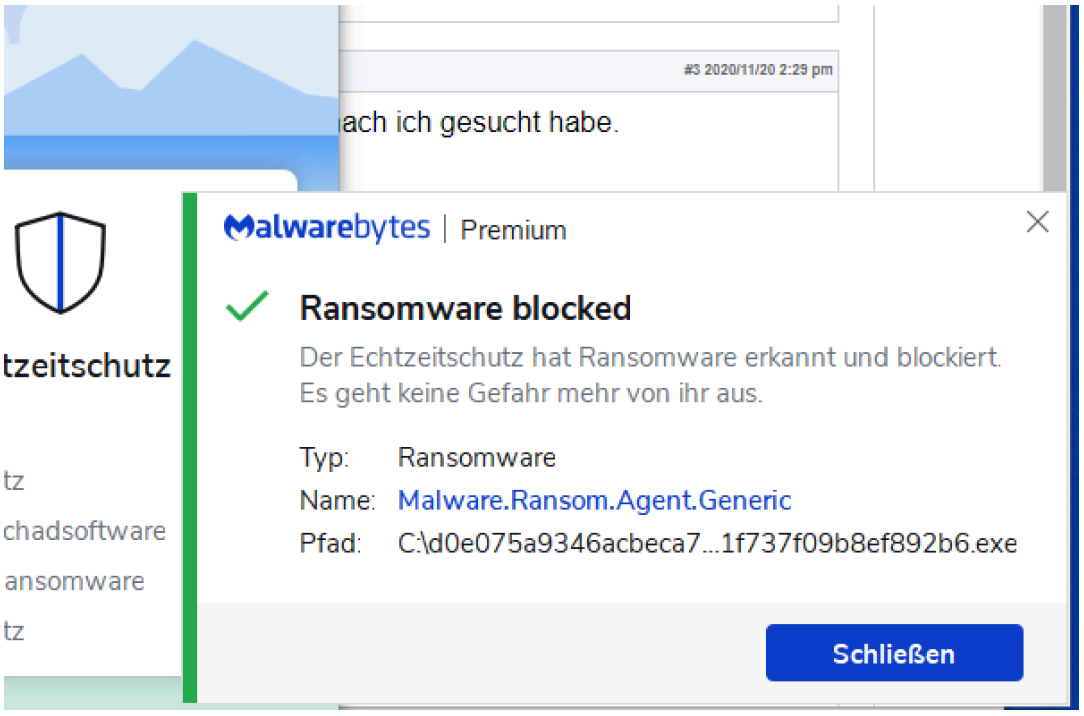
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

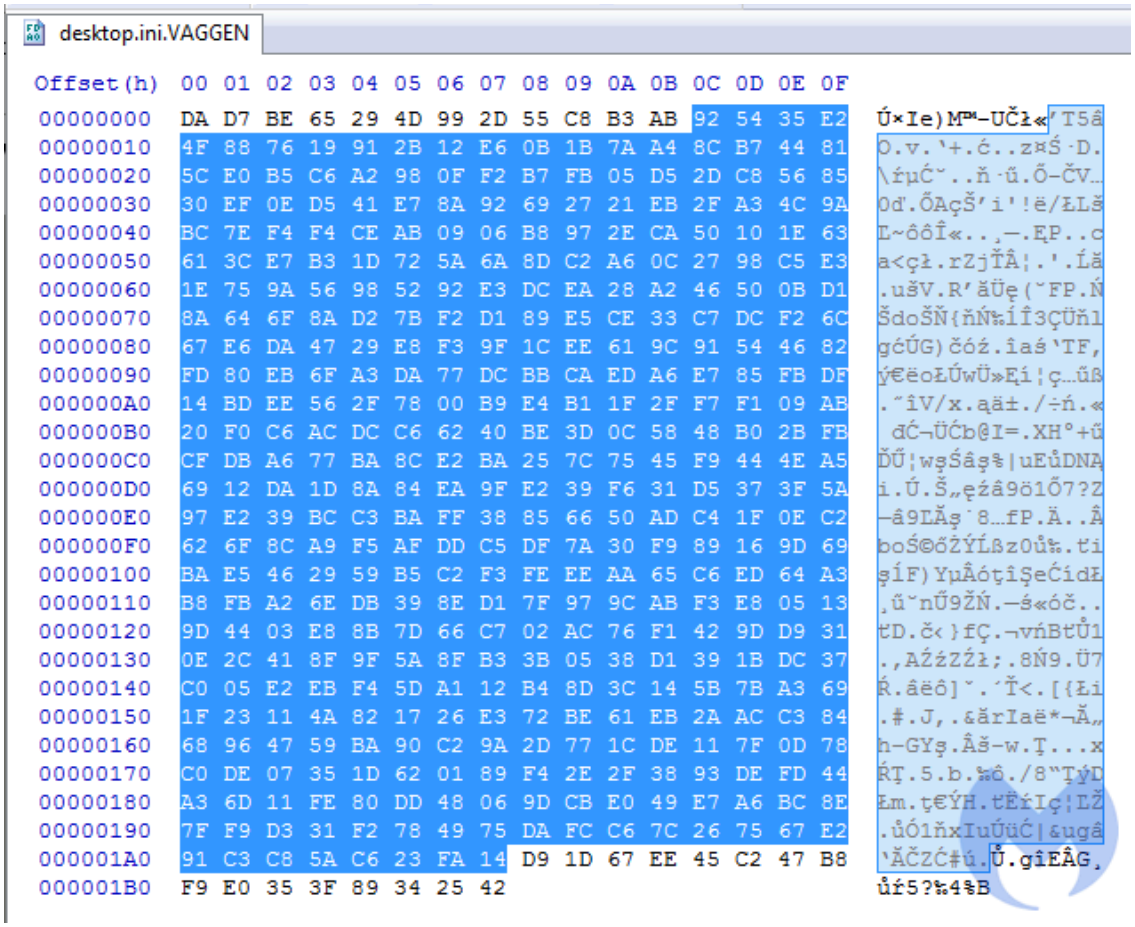
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

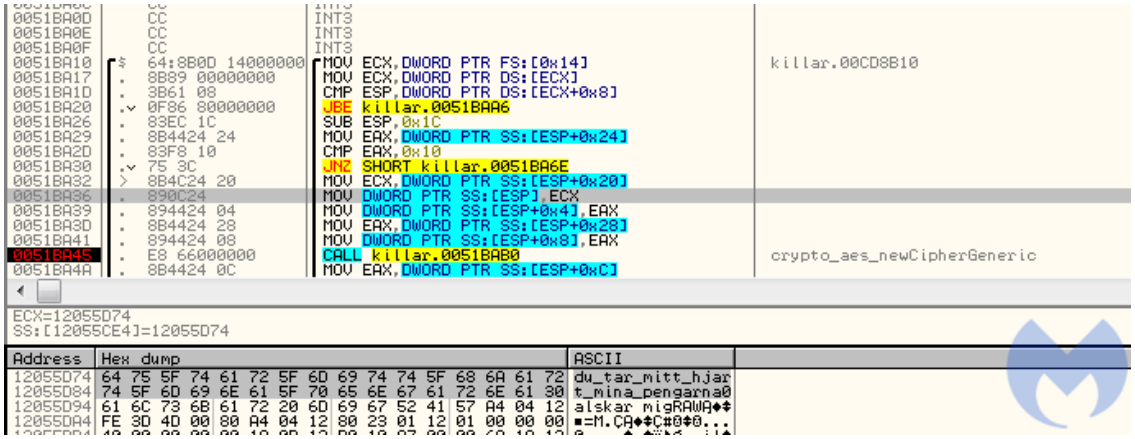


The content of the output file (with .VAGGEN extension) contains:

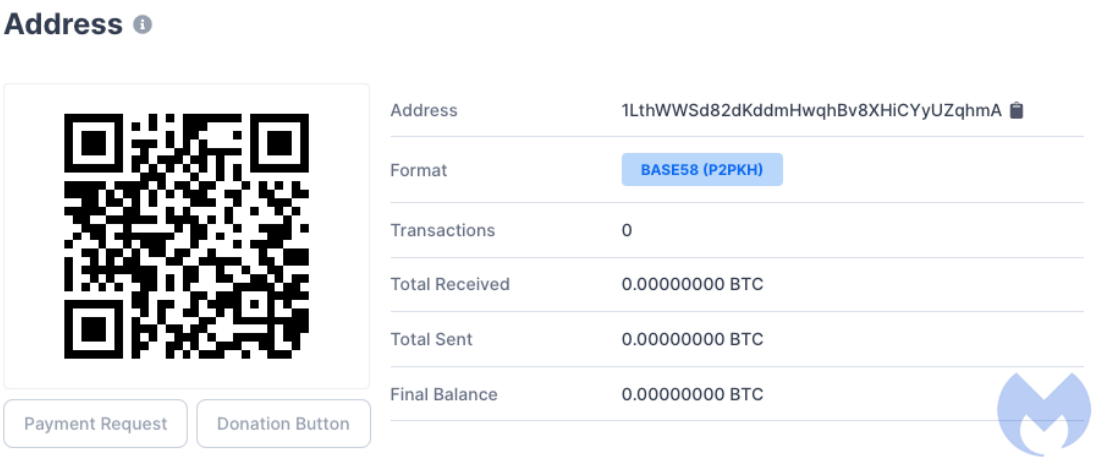
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

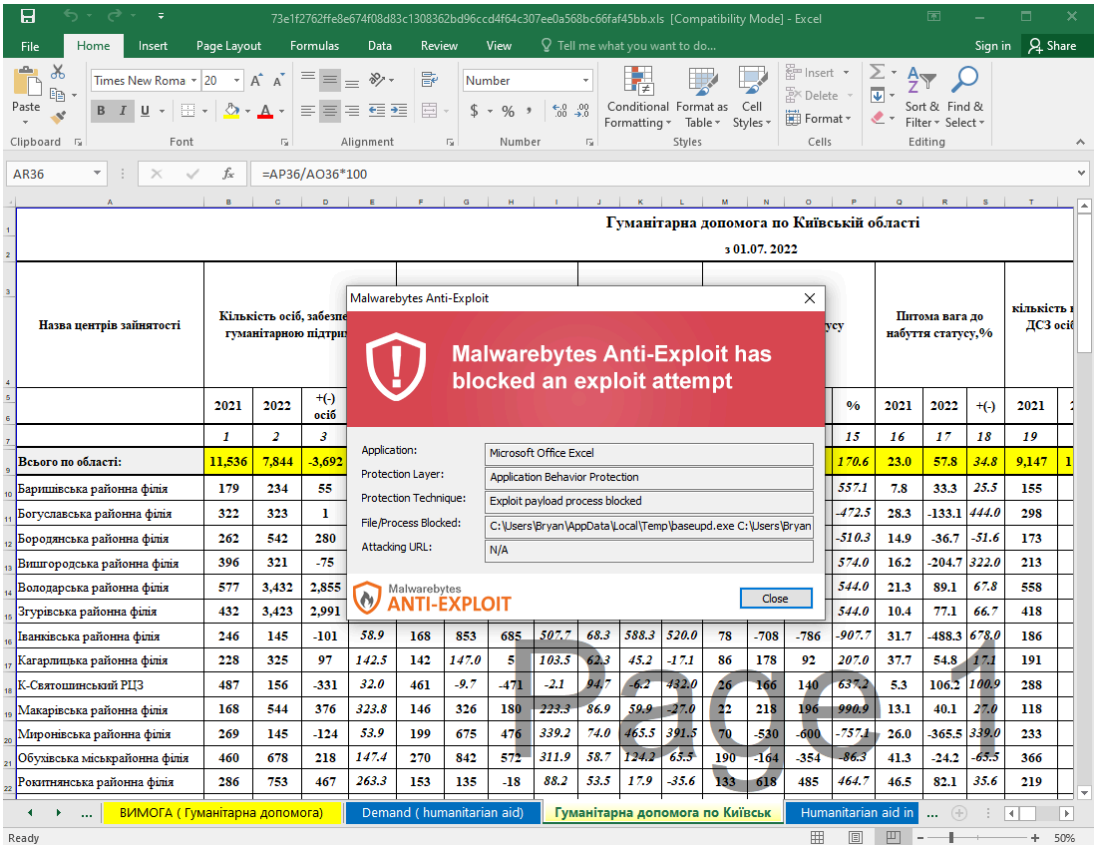
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

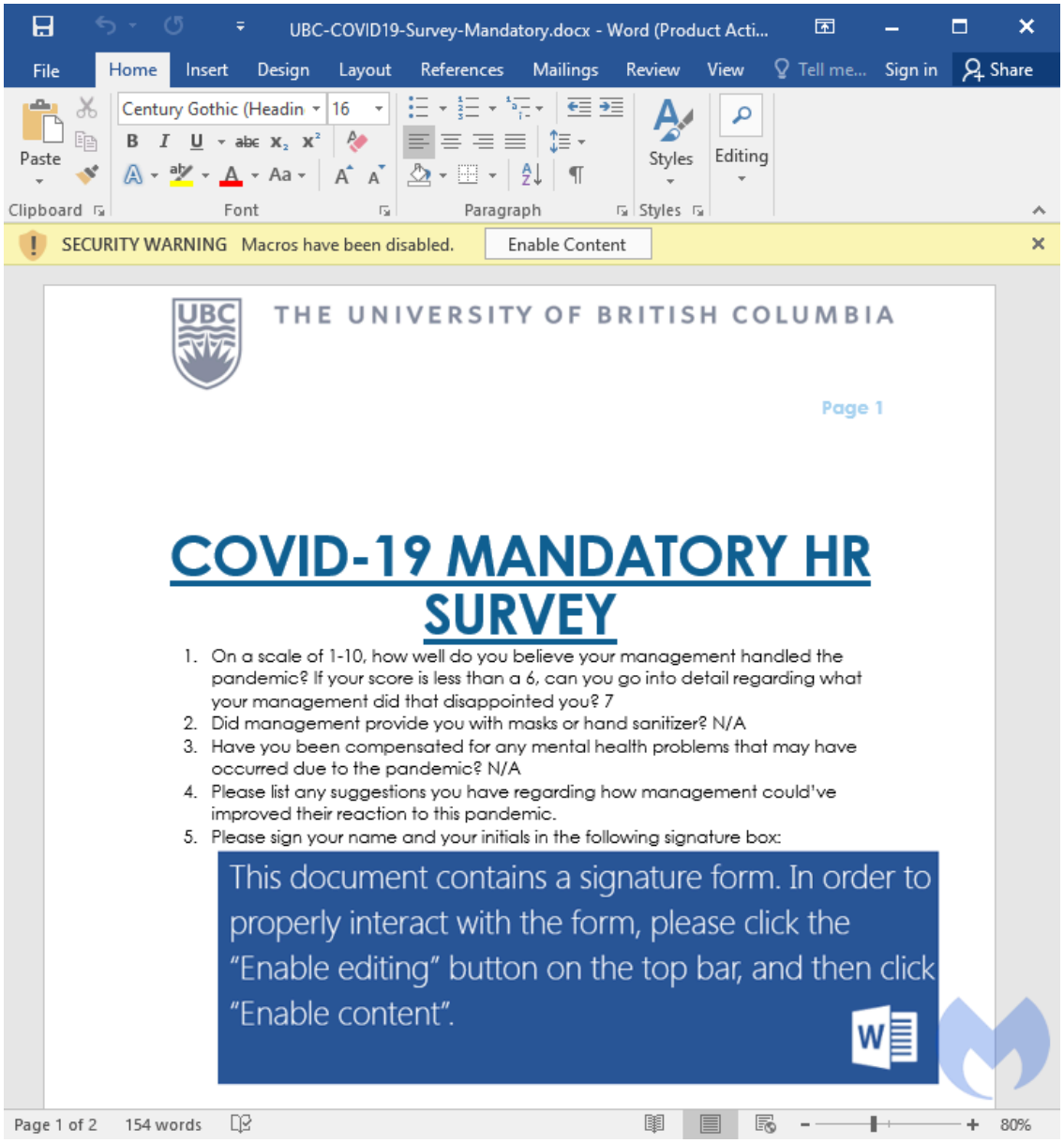
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

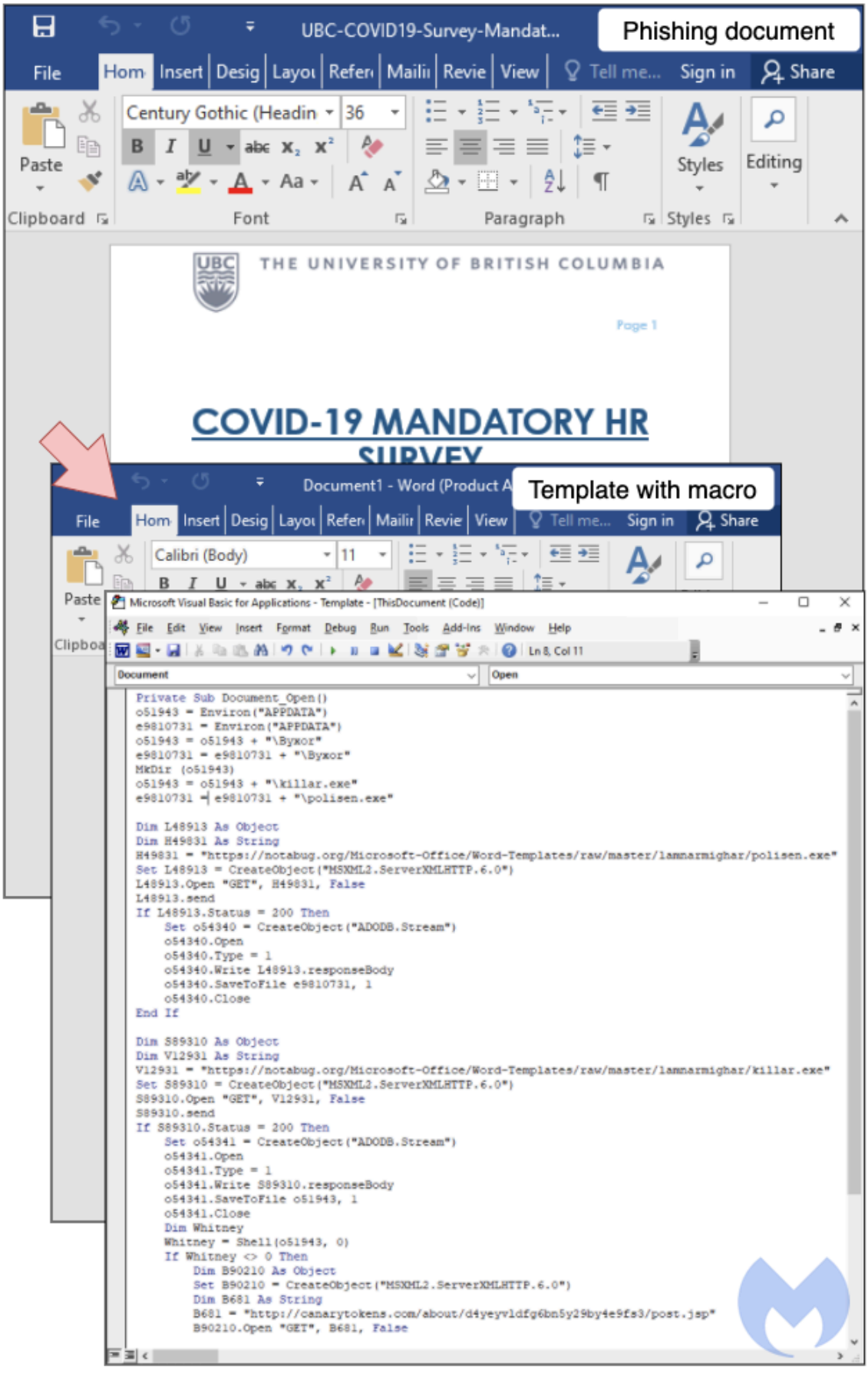
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

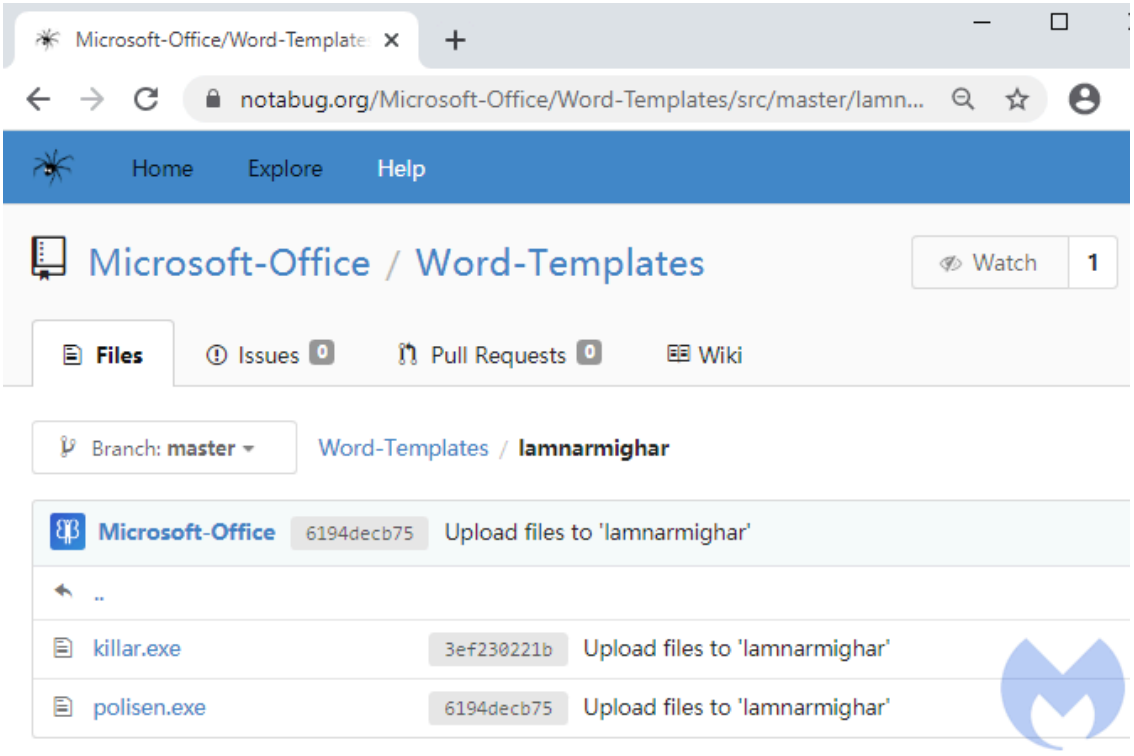
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



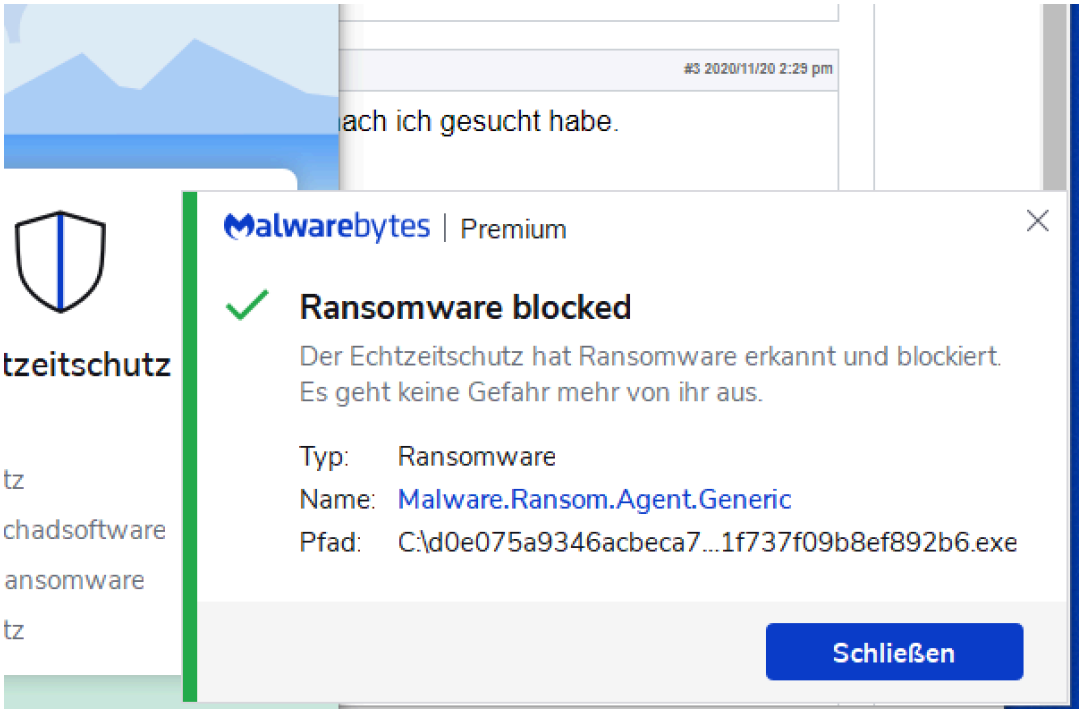
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

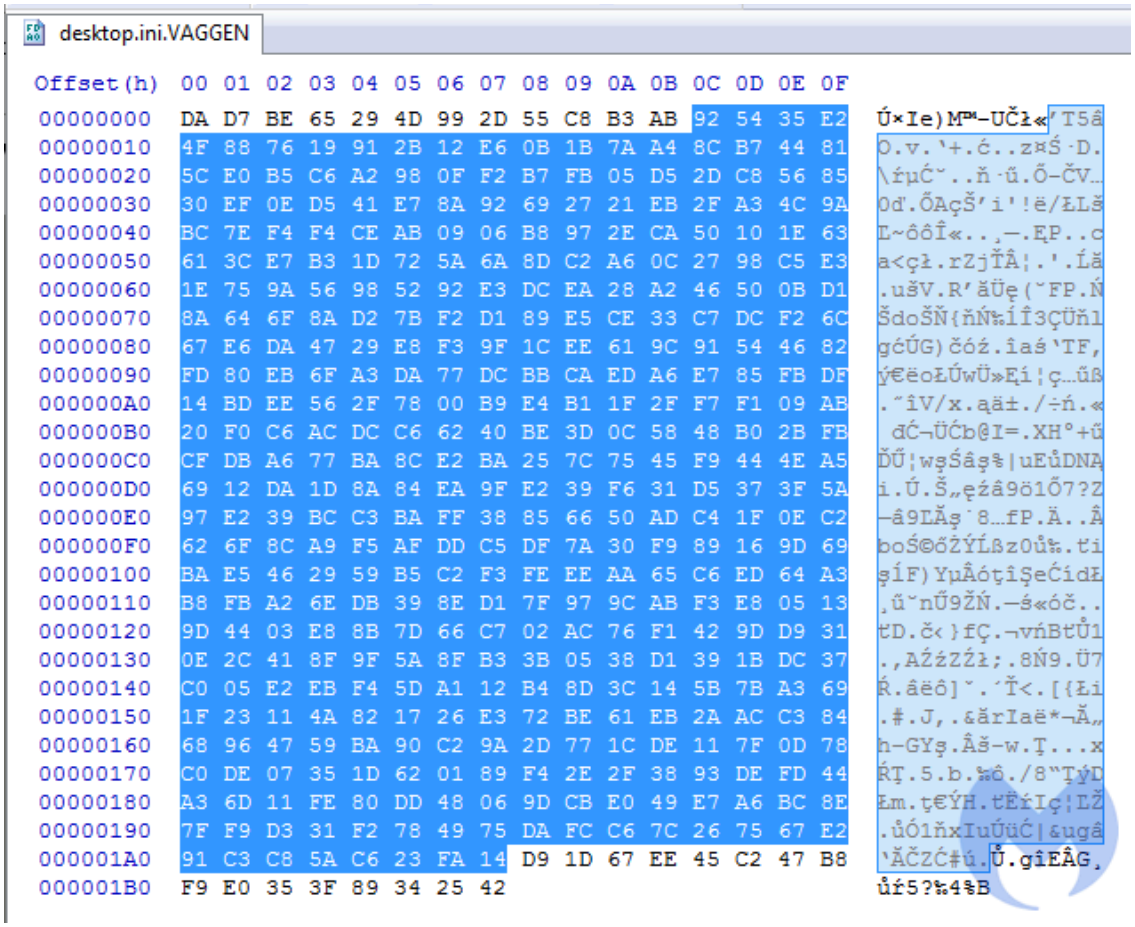
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



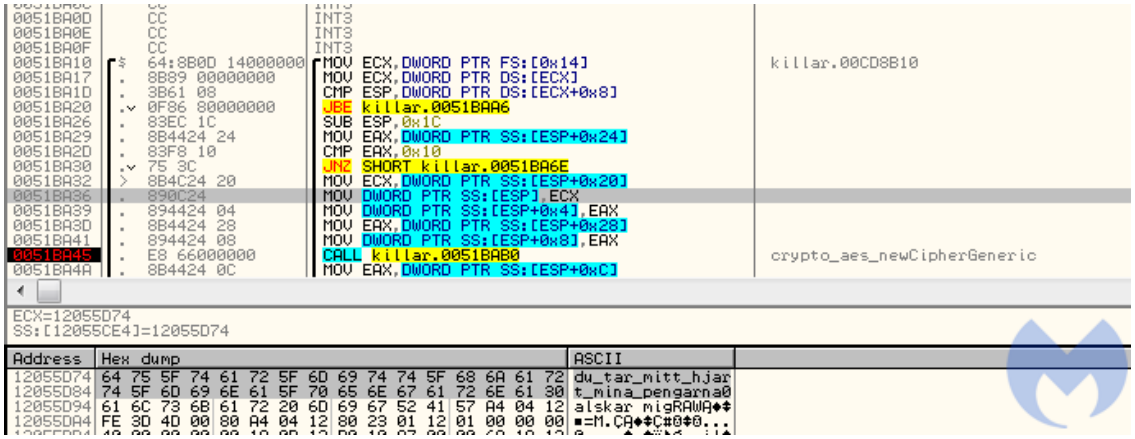
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

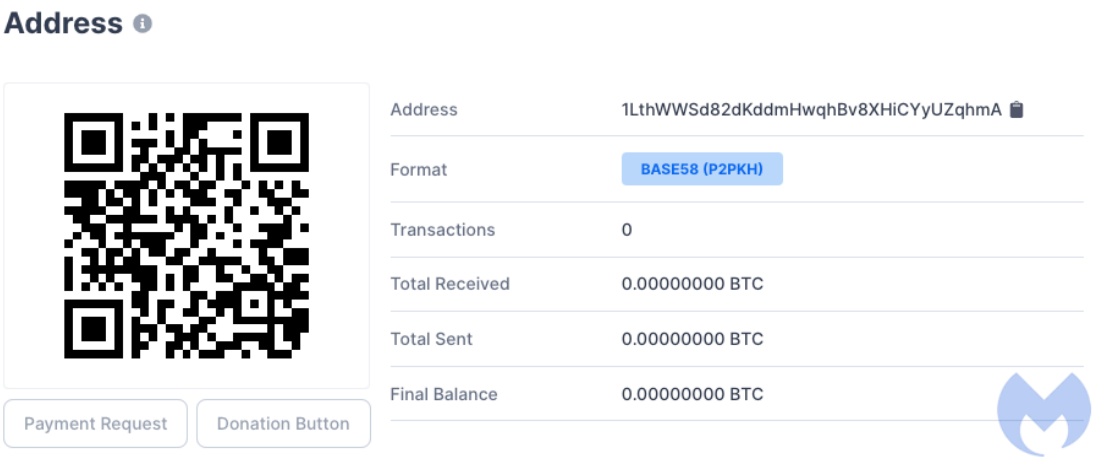
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

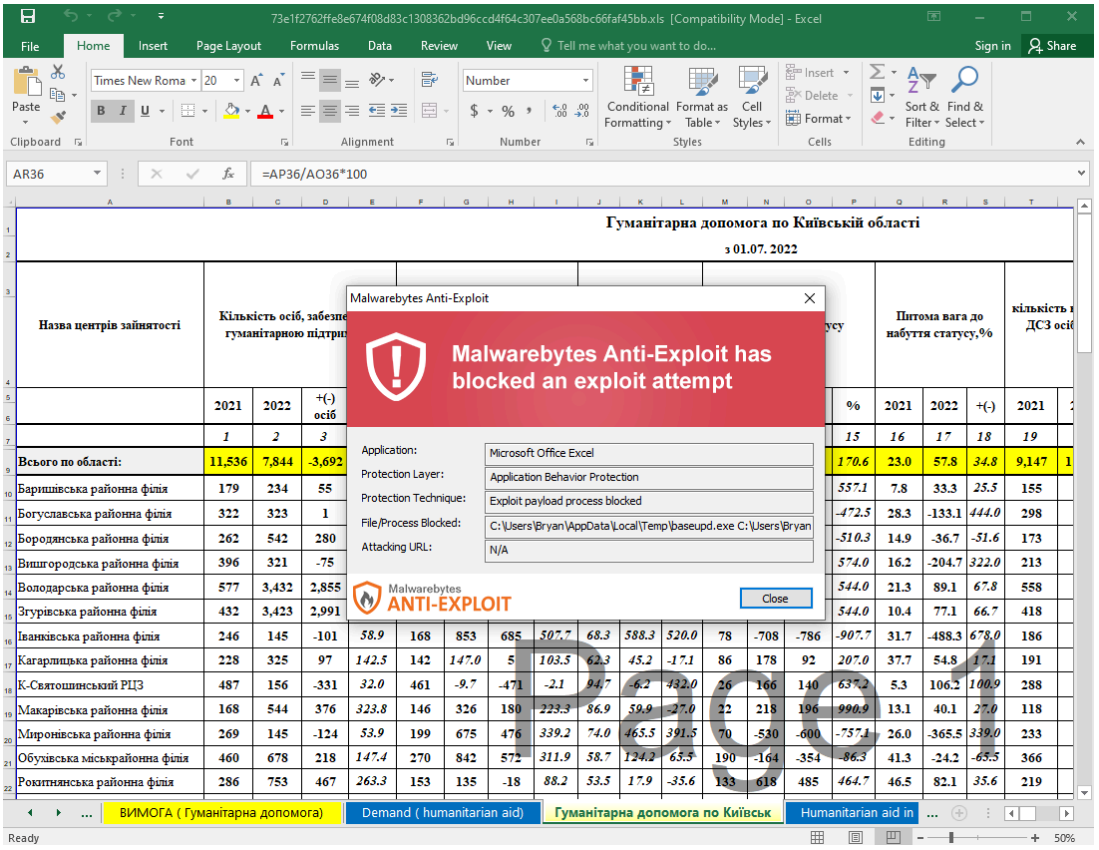
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

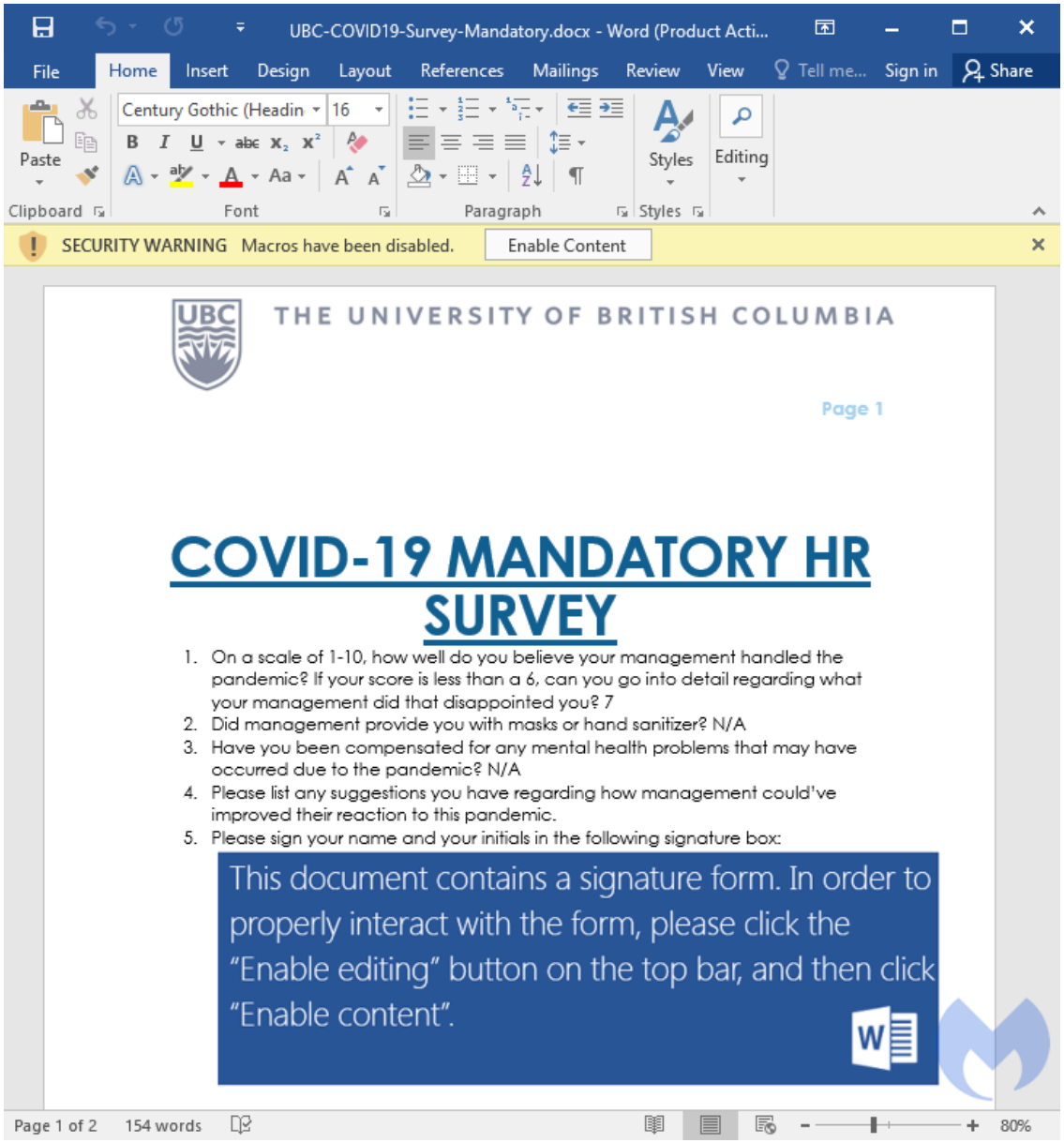
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

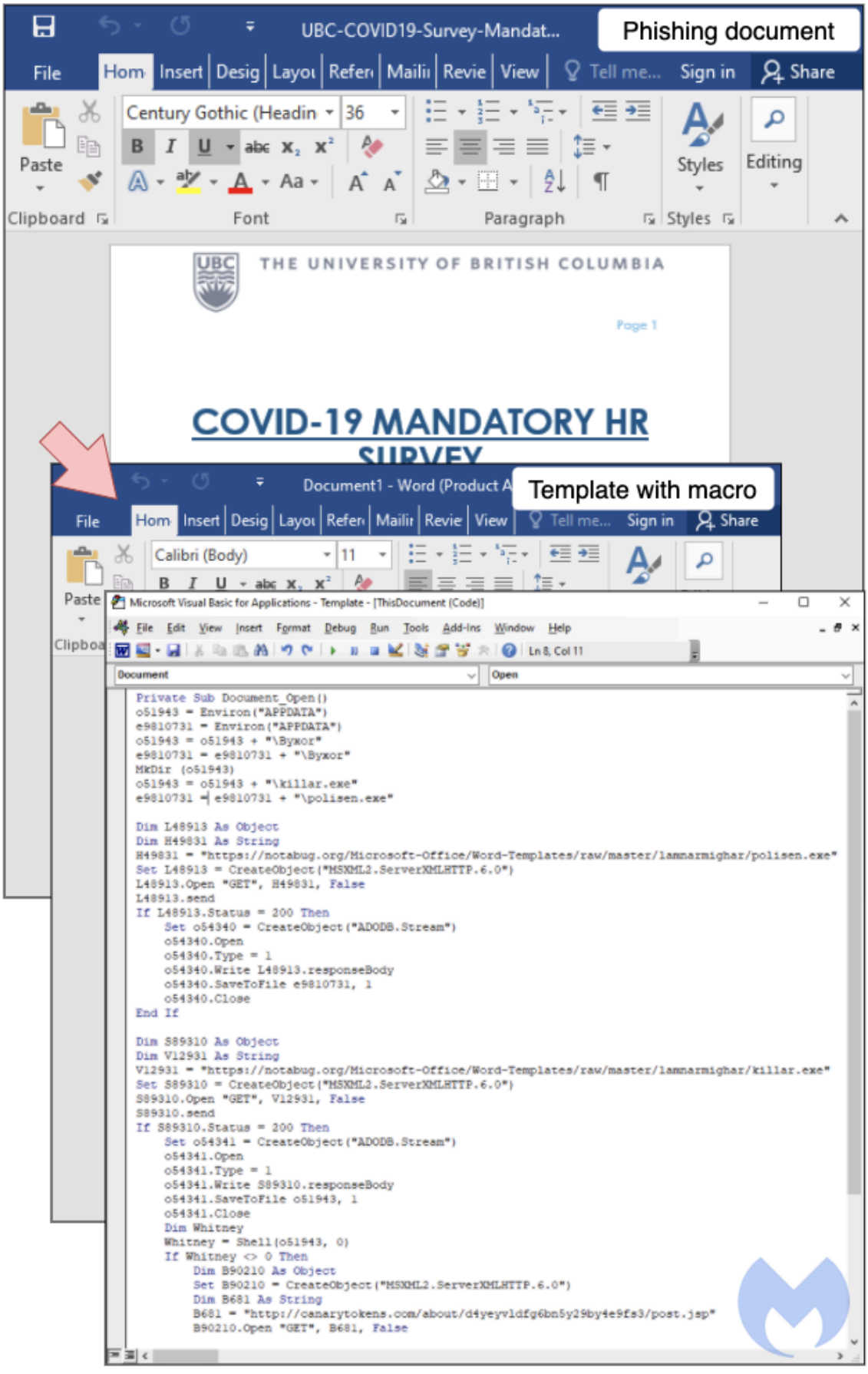
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

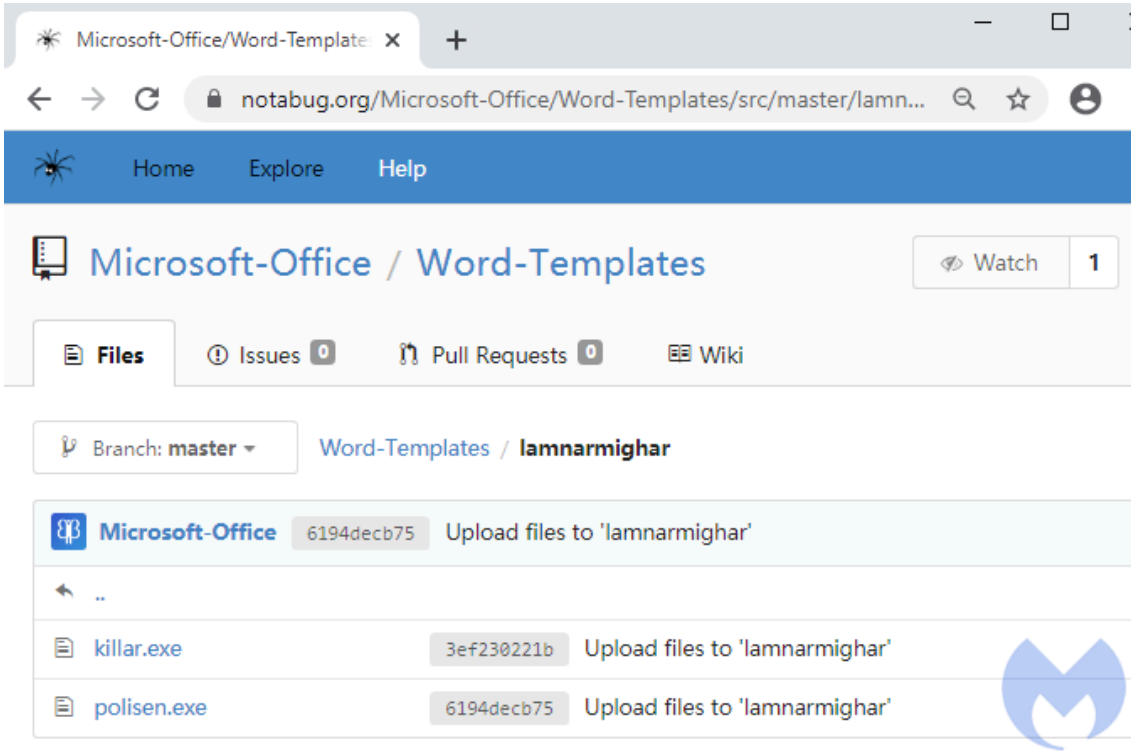
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



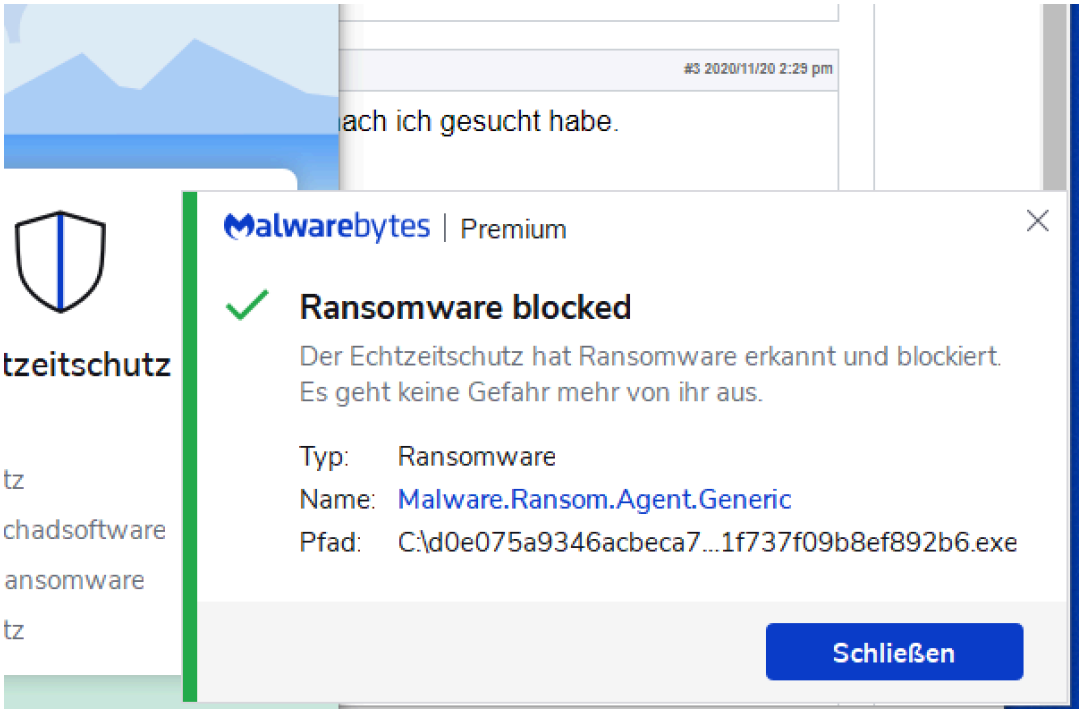
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

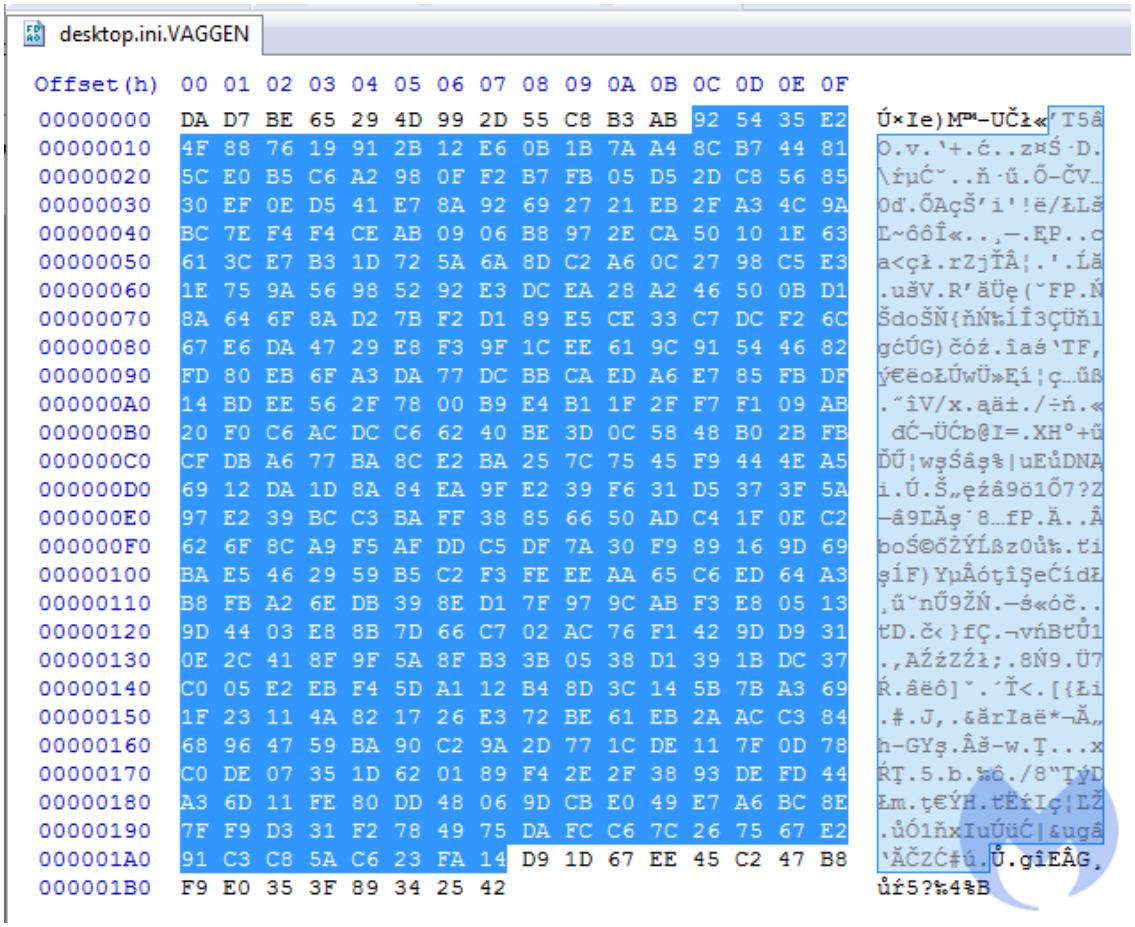
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



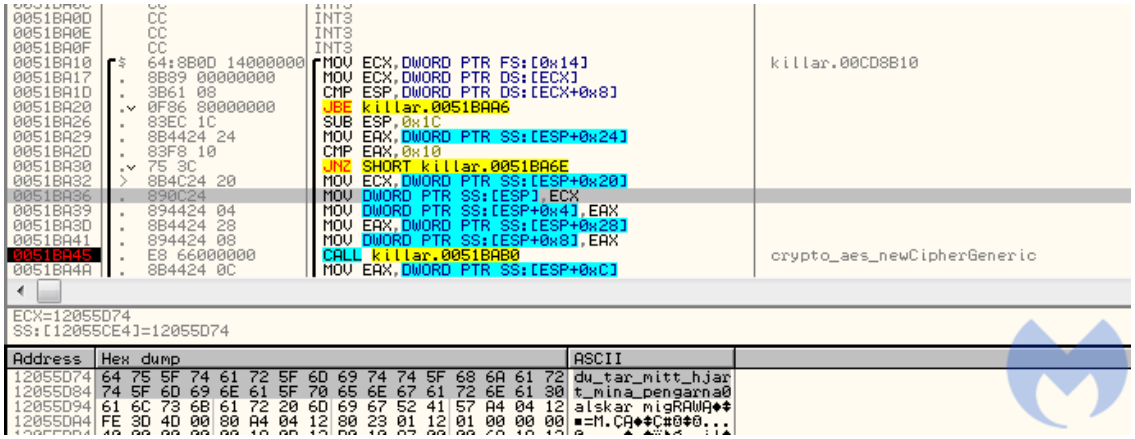
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

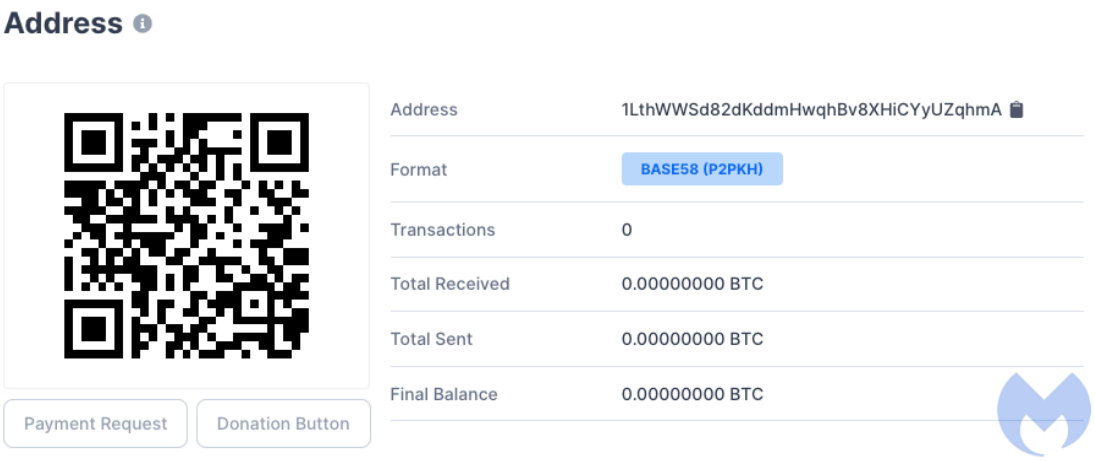
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

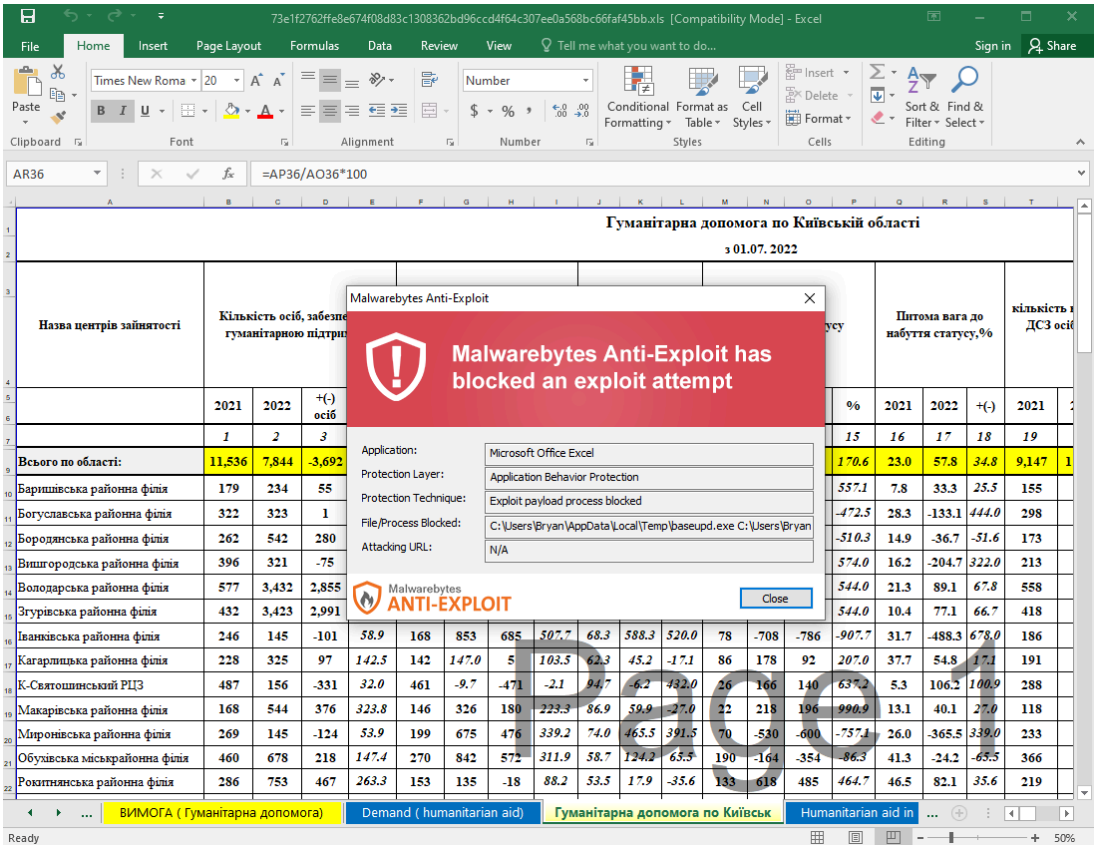
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

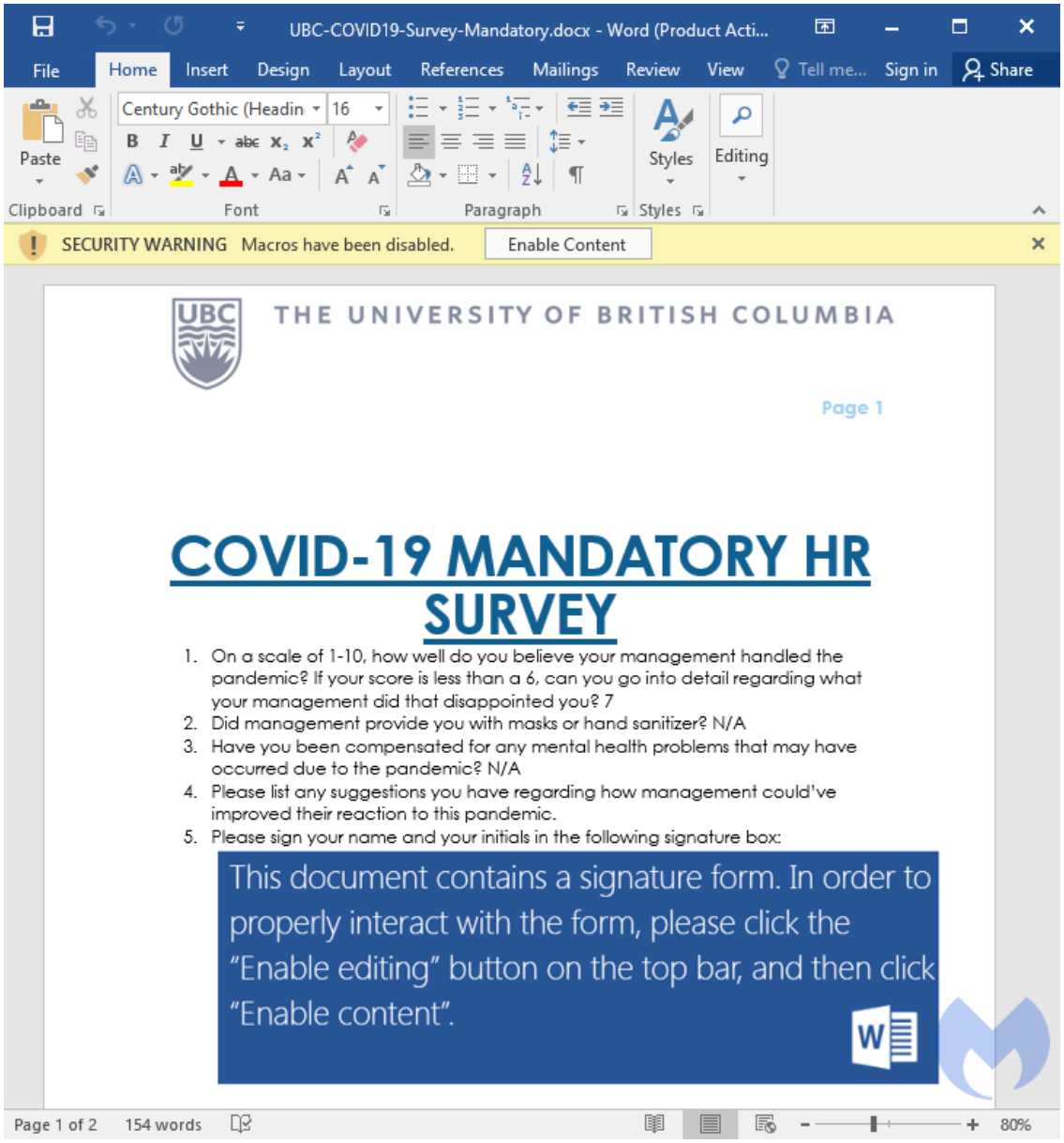
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

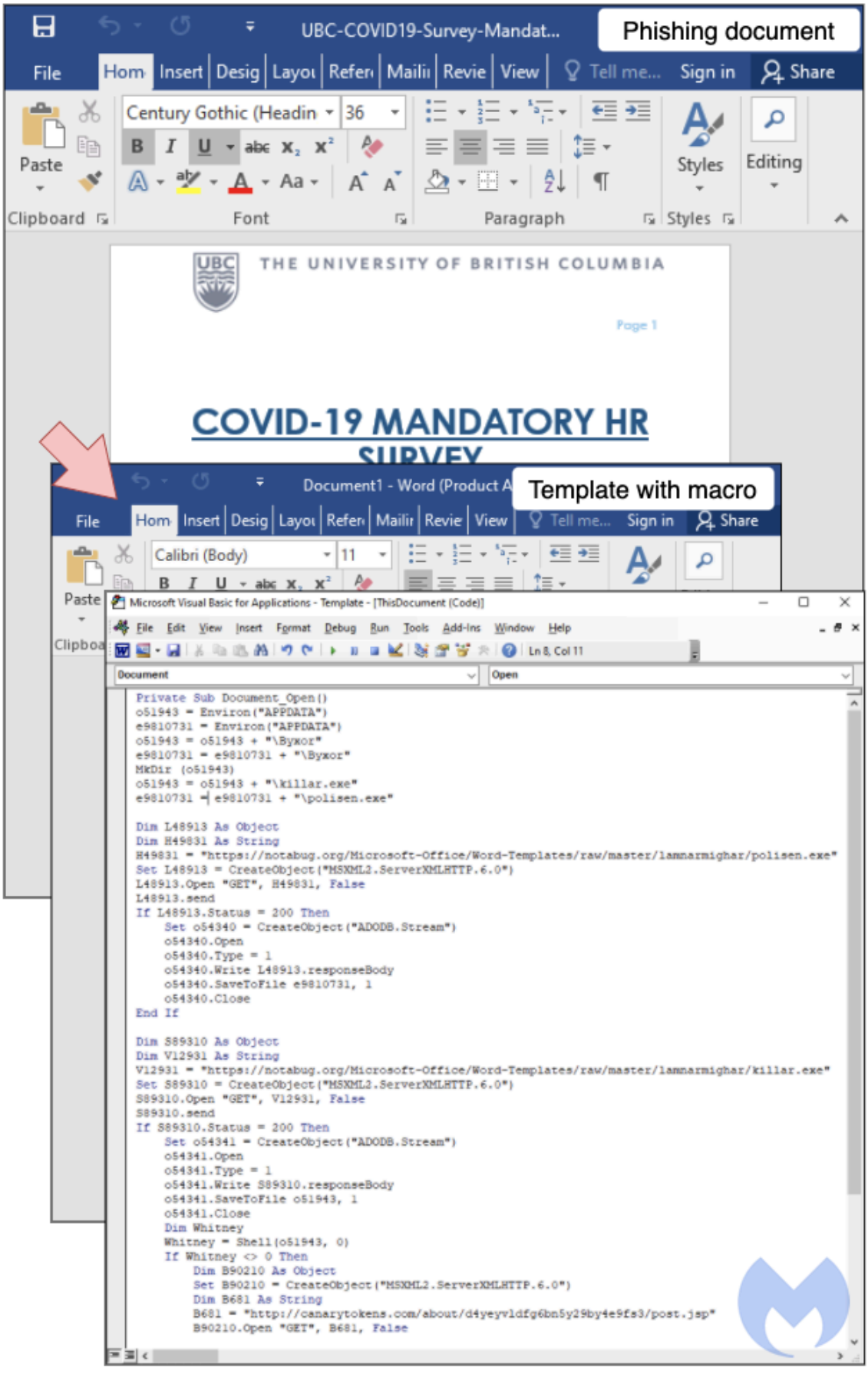
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

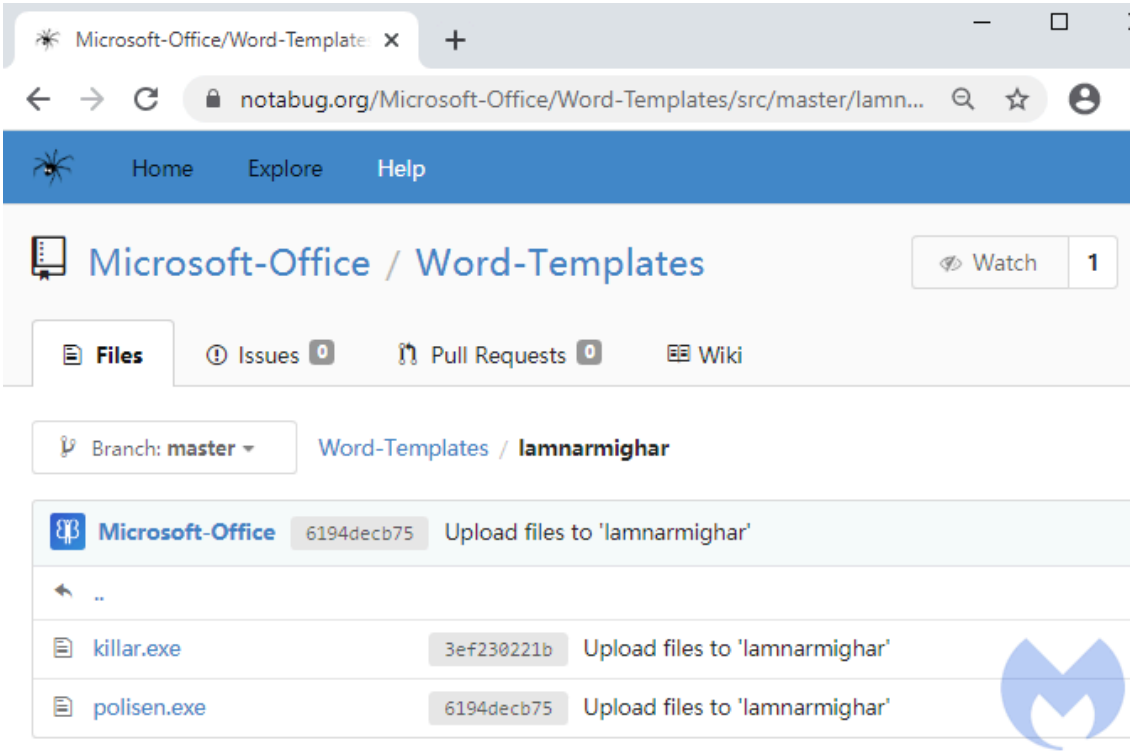
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



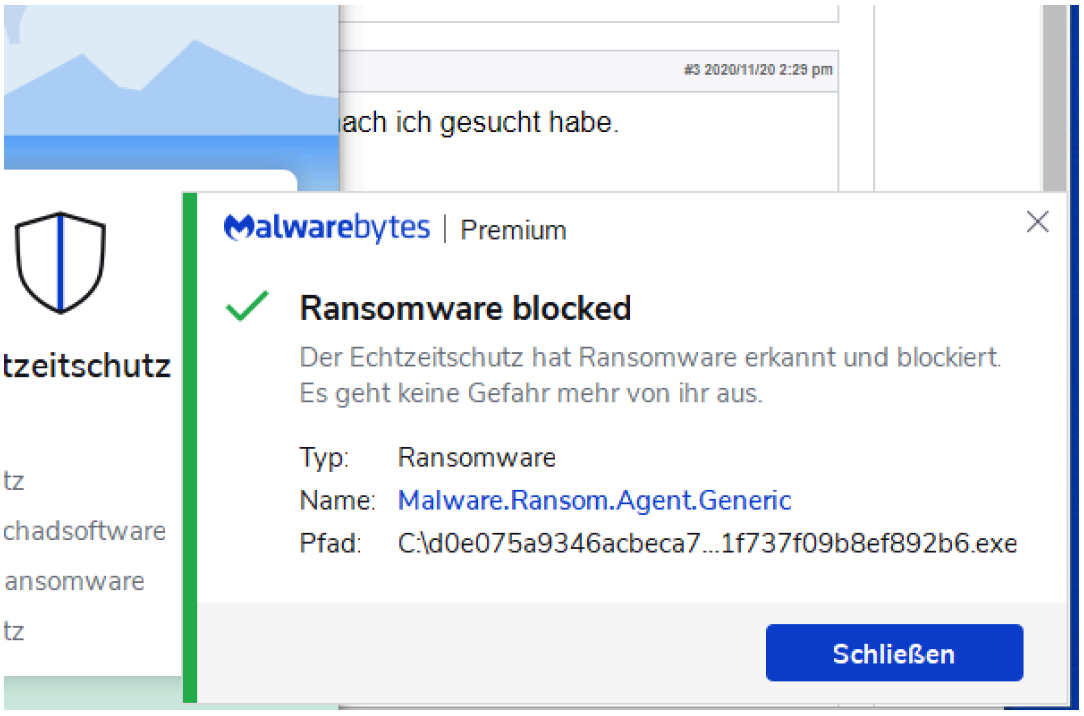
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

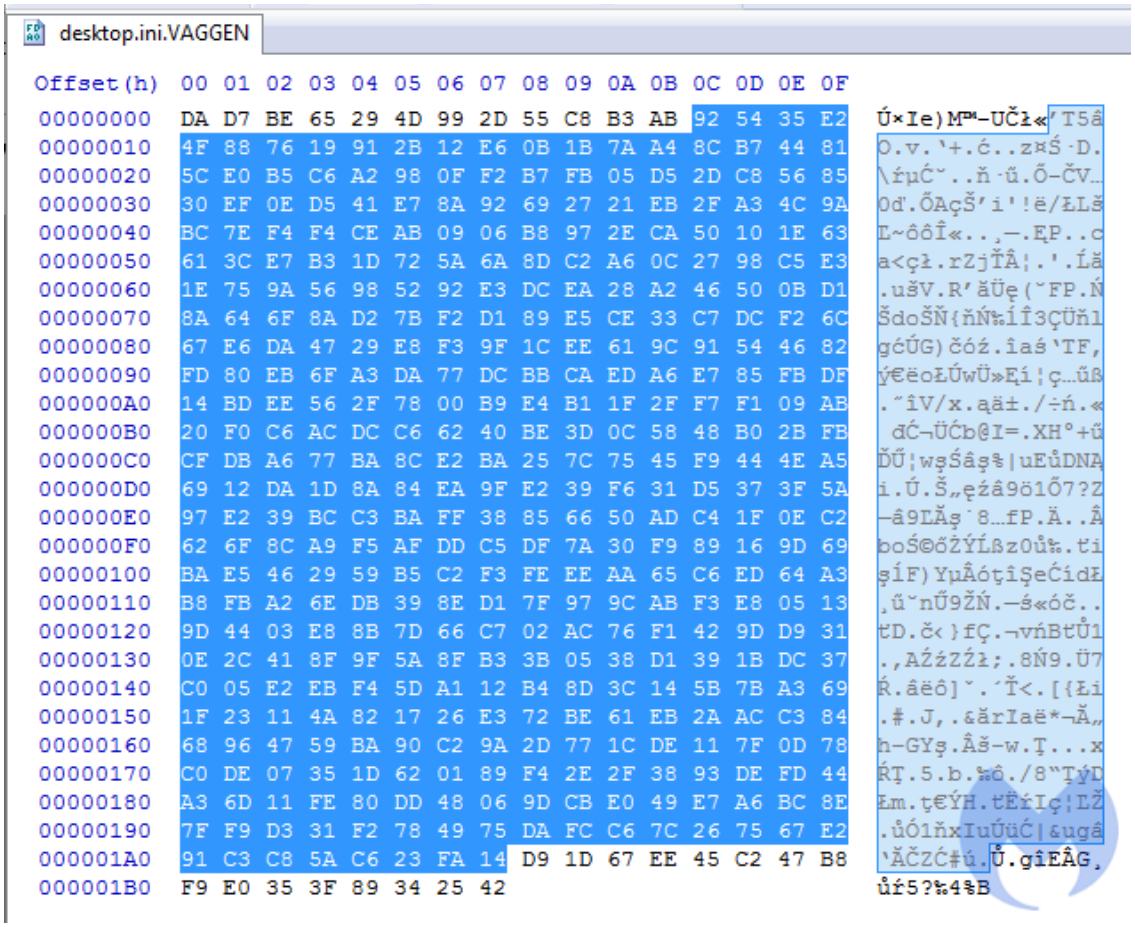
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

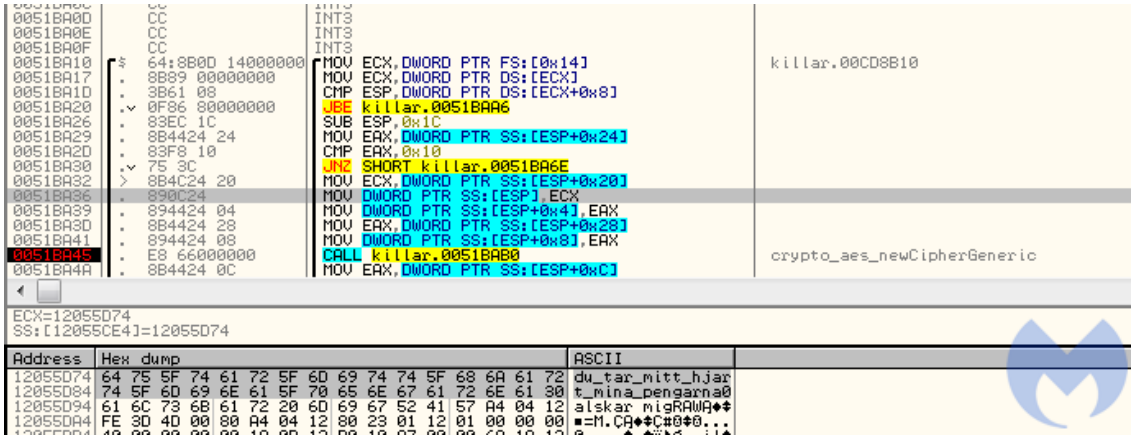


The content of the output file (with .VAGGEN extension) contains:

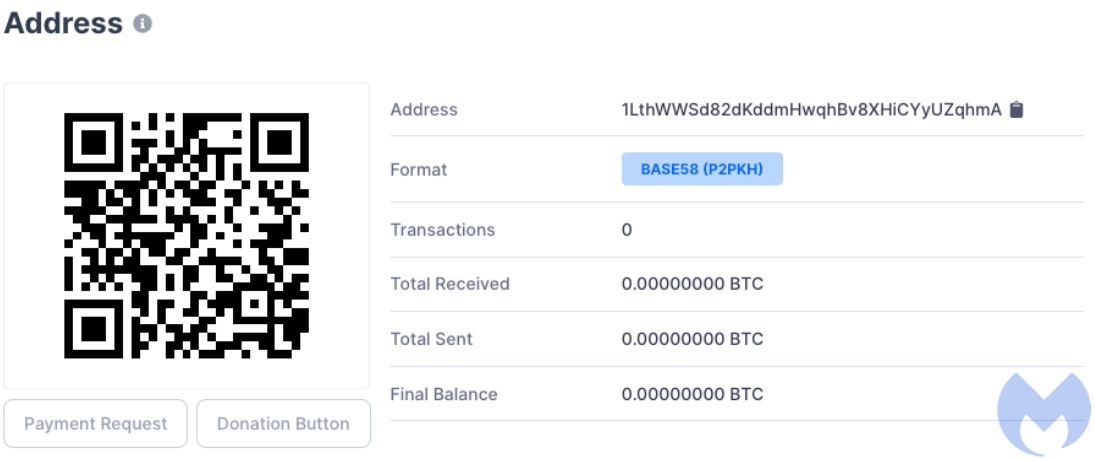
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

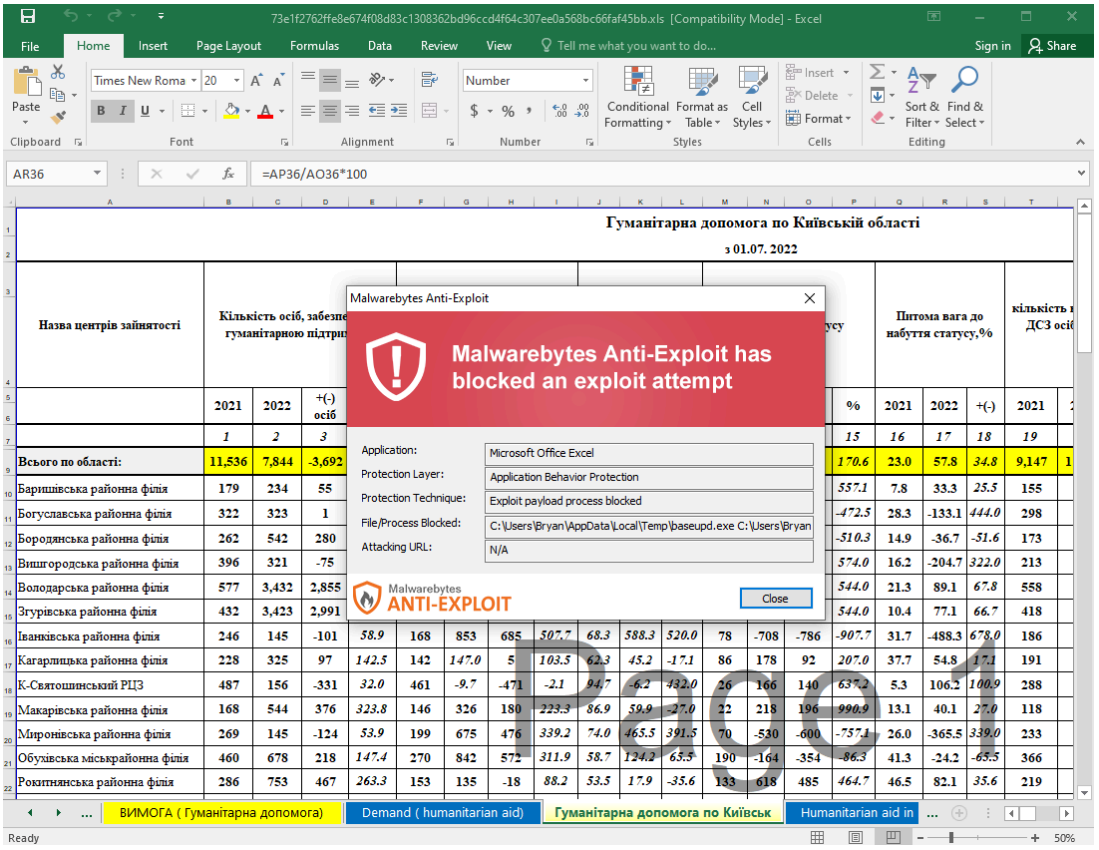
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

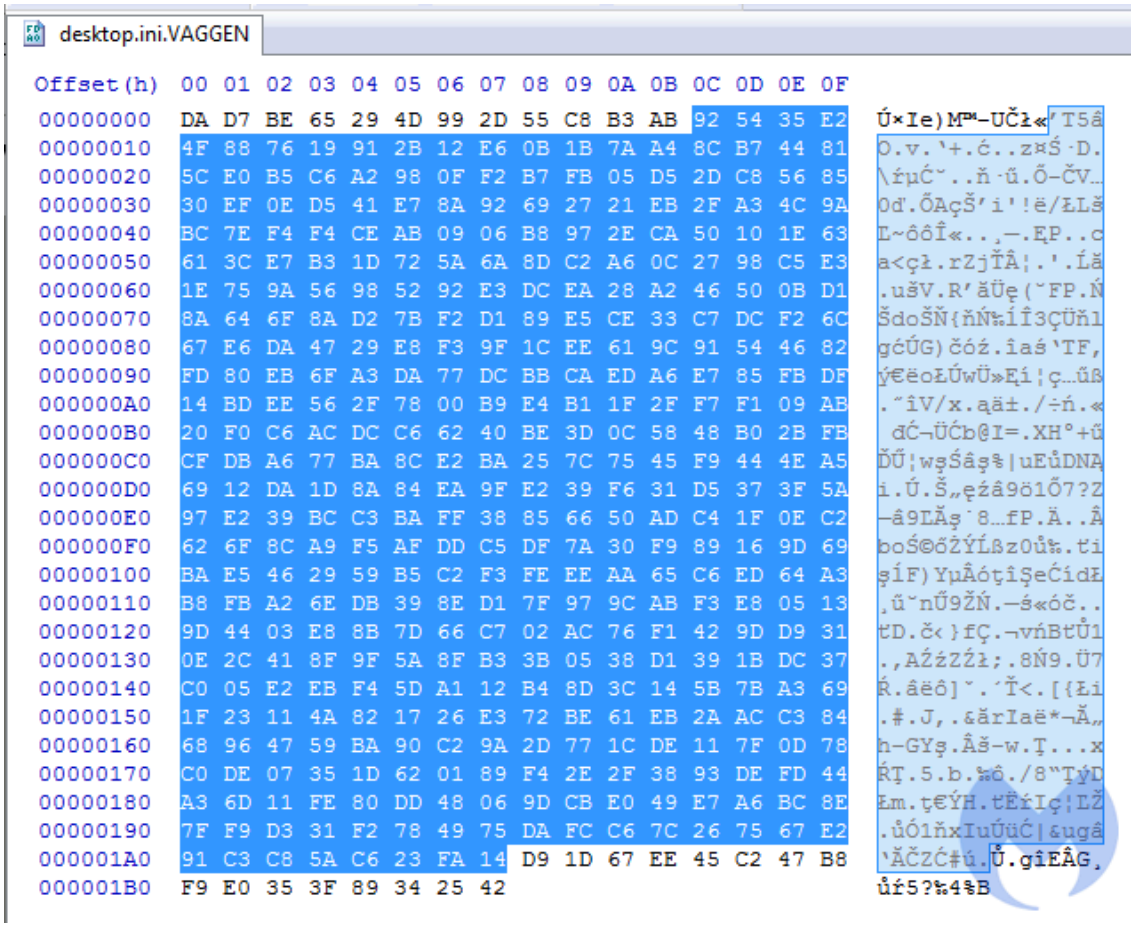
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

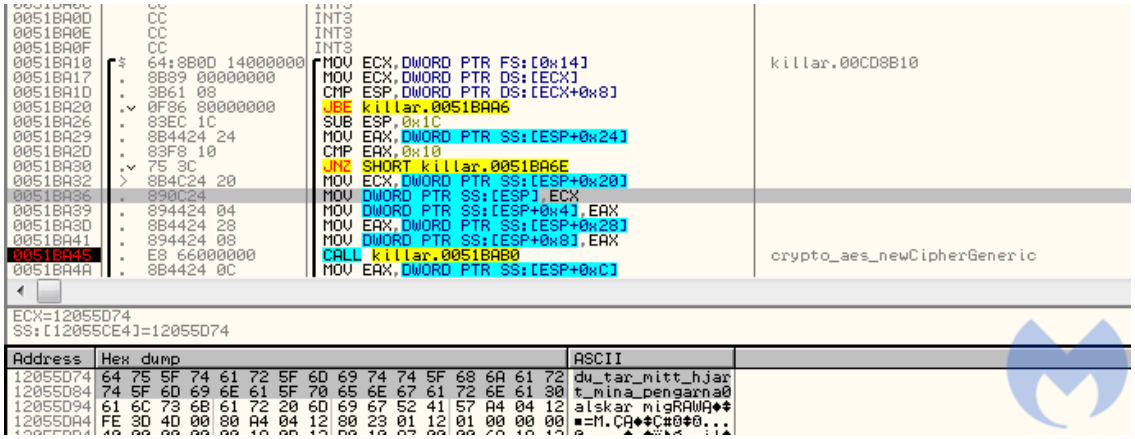
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

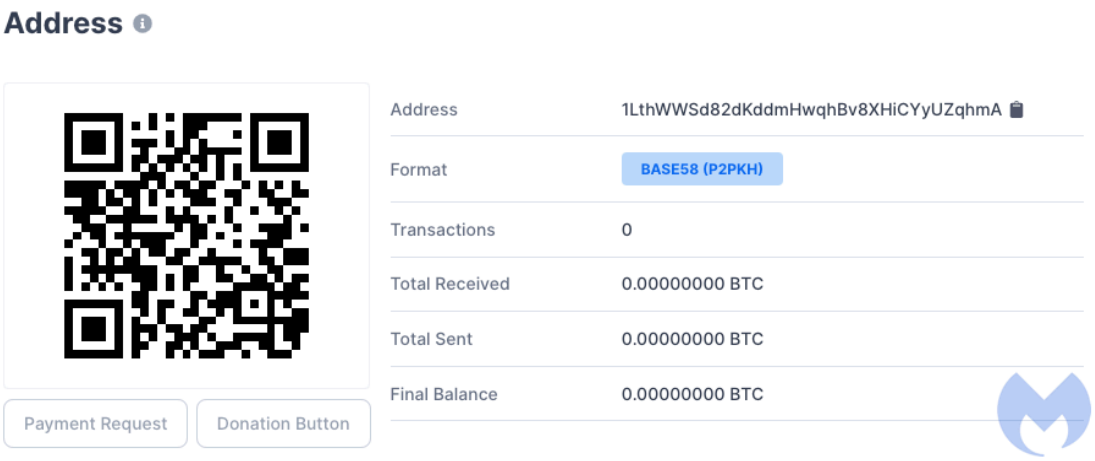
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

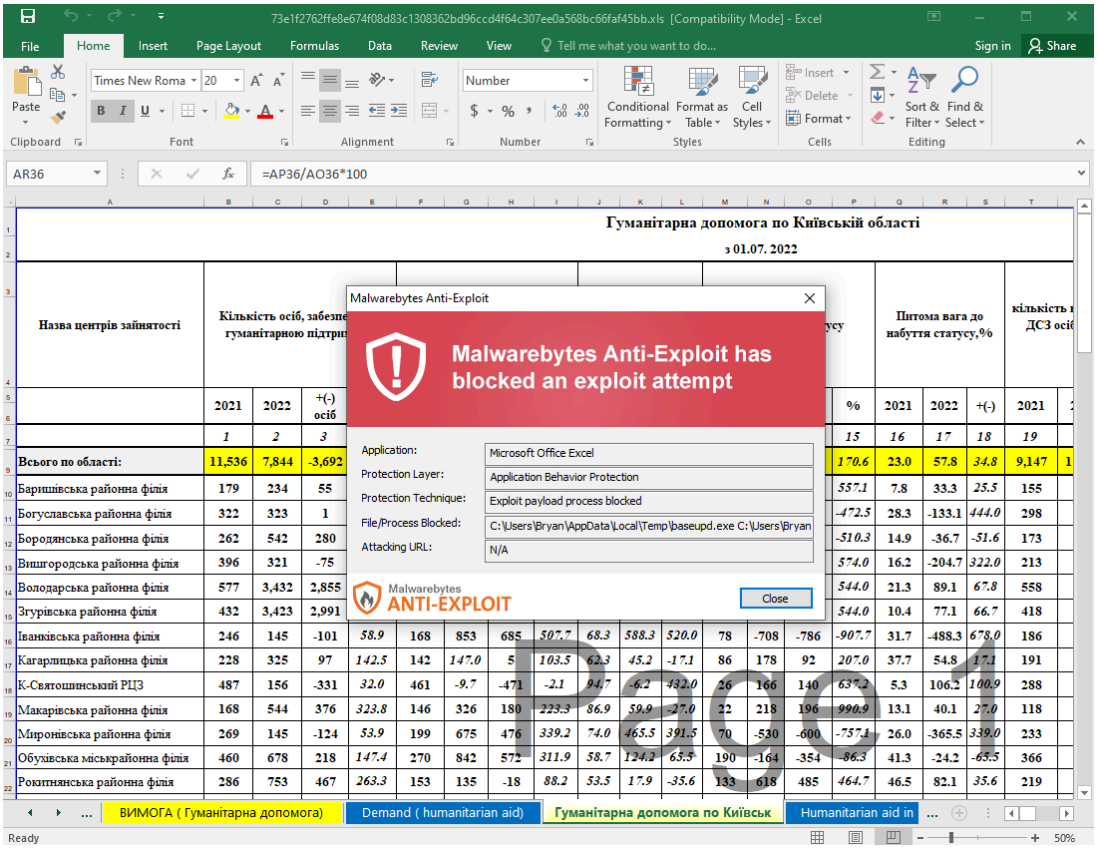
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

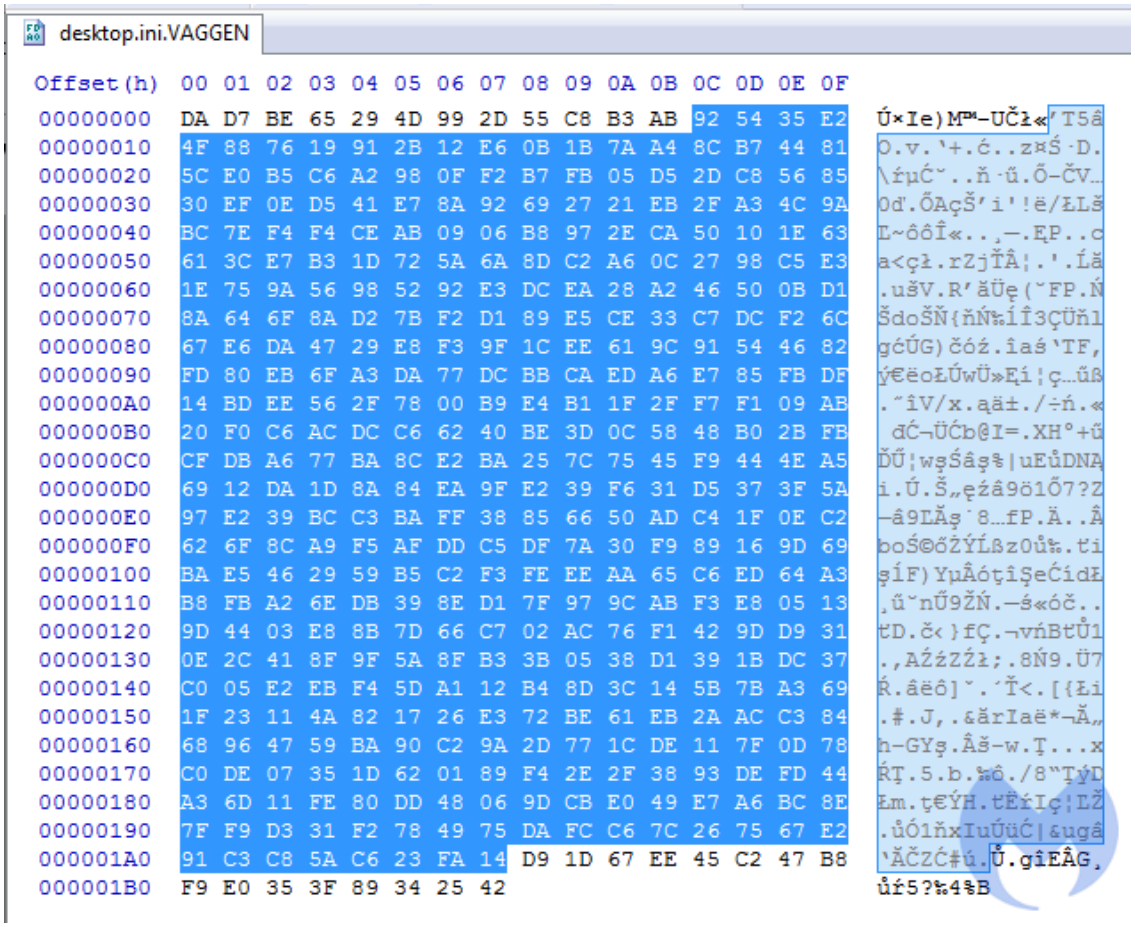
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

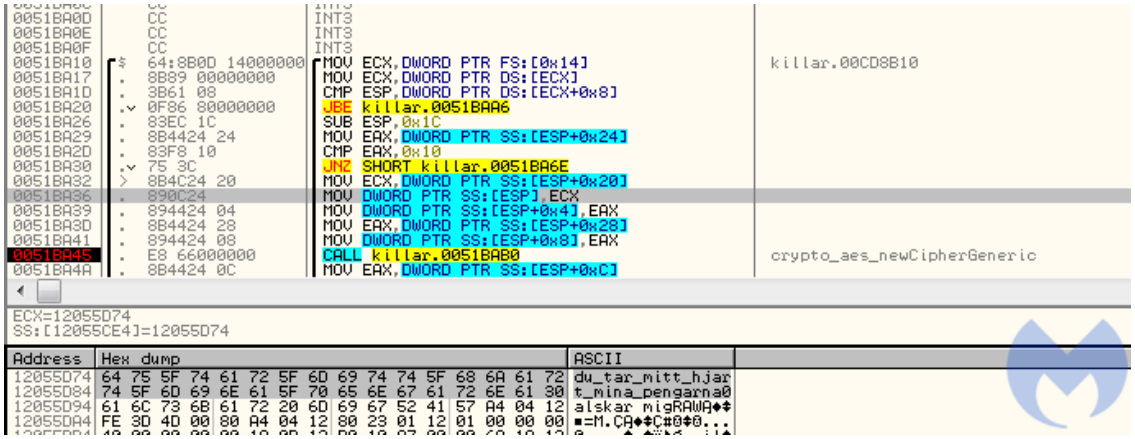


The content of the output file (with .VAGGEN extension) contains:

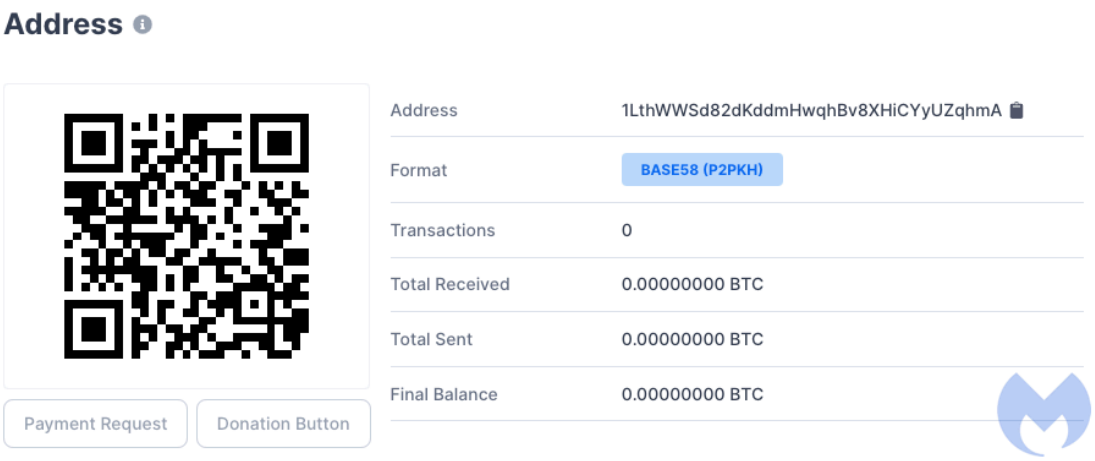
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

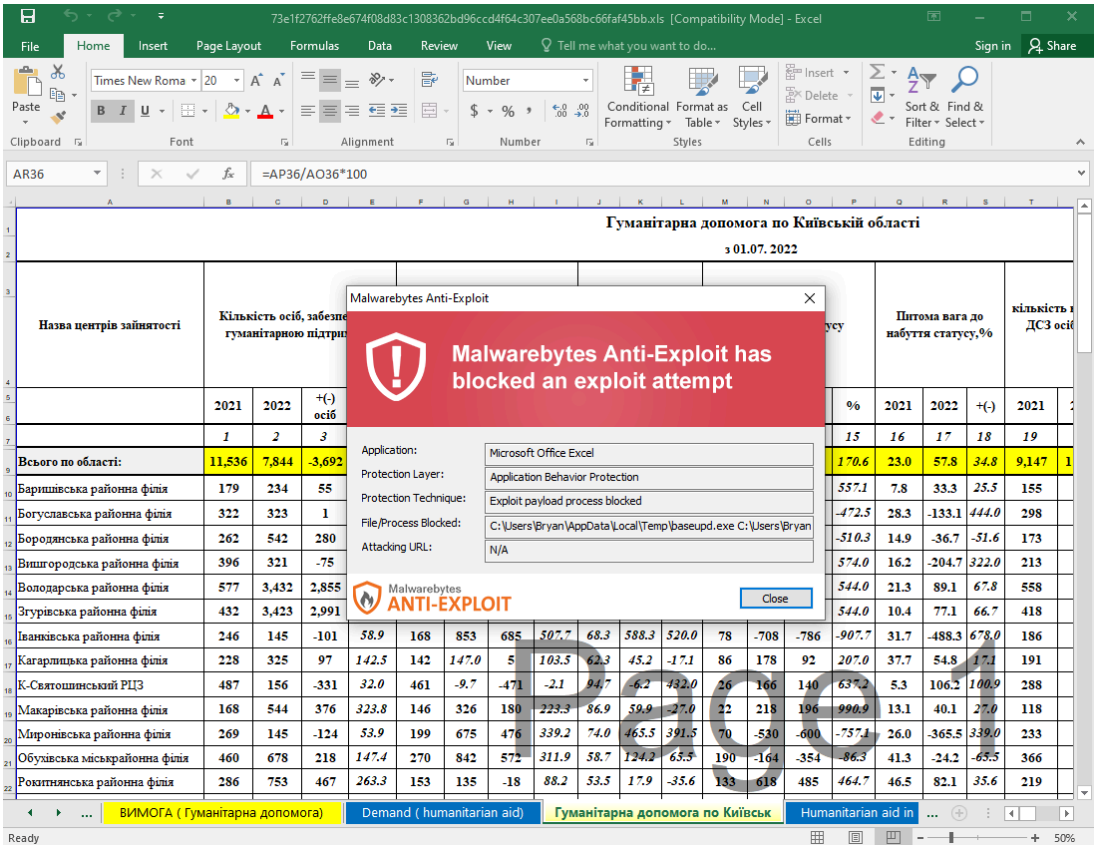
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

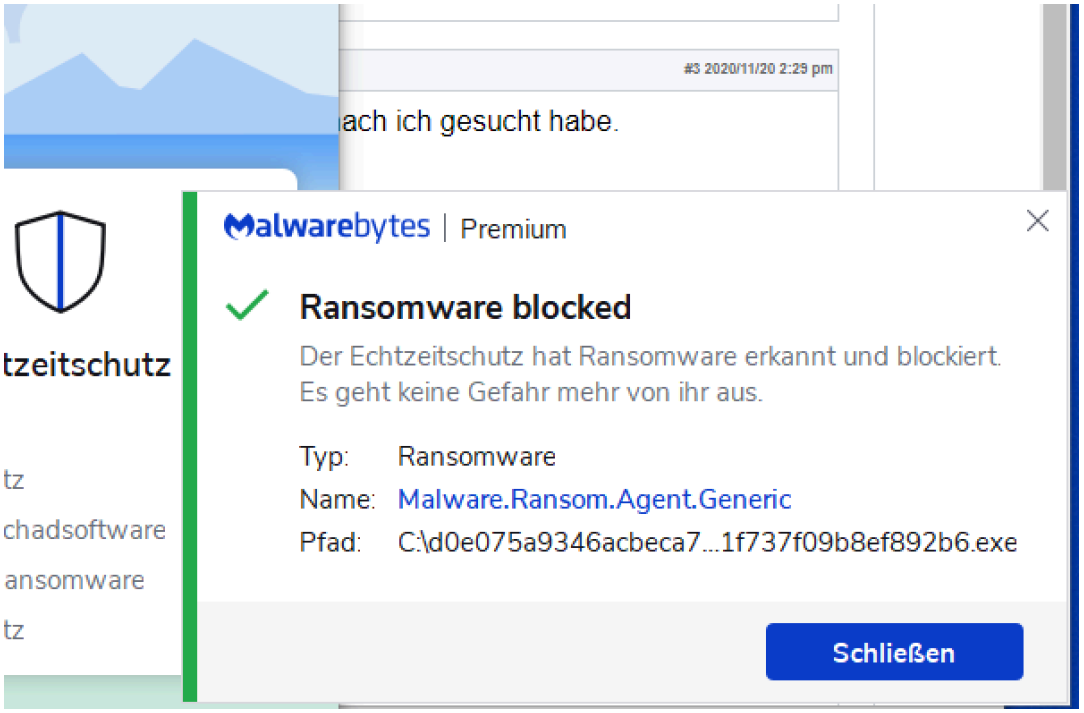
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

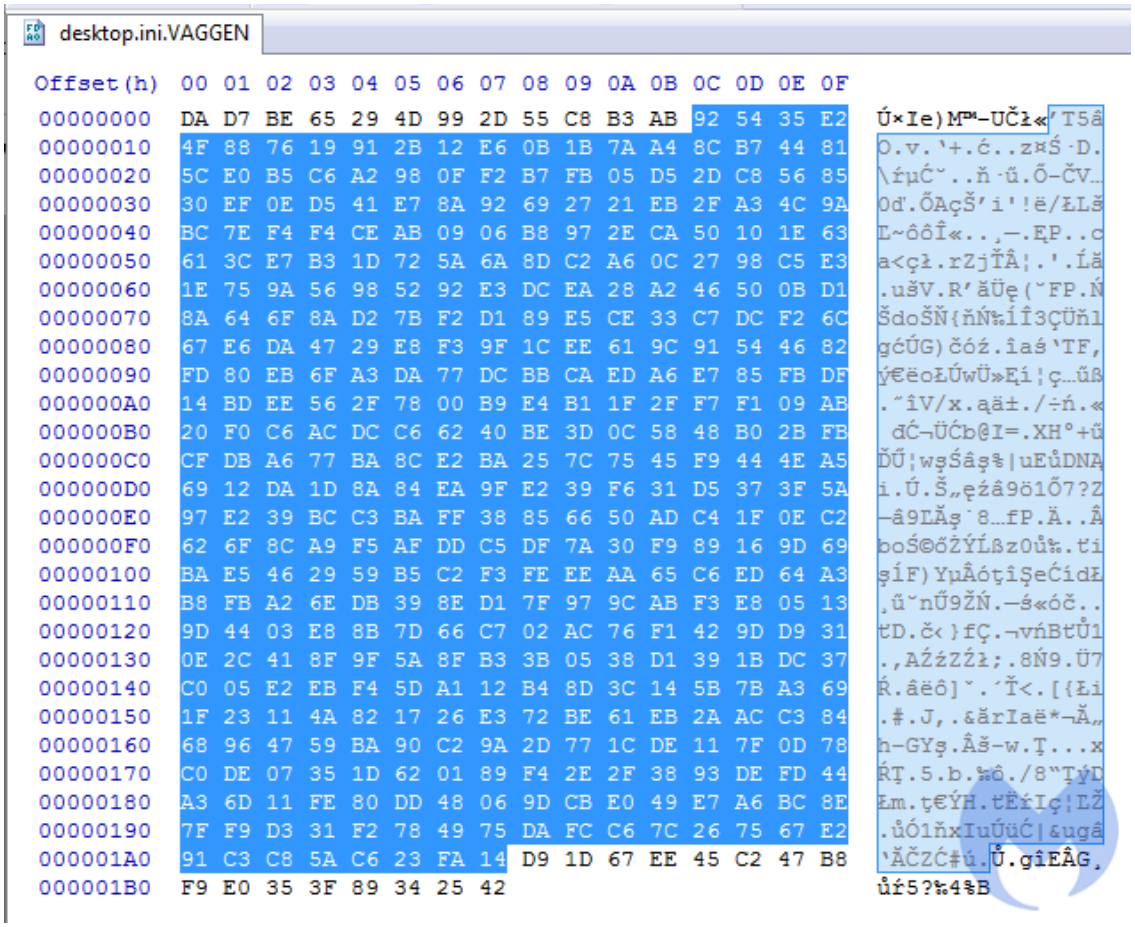
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



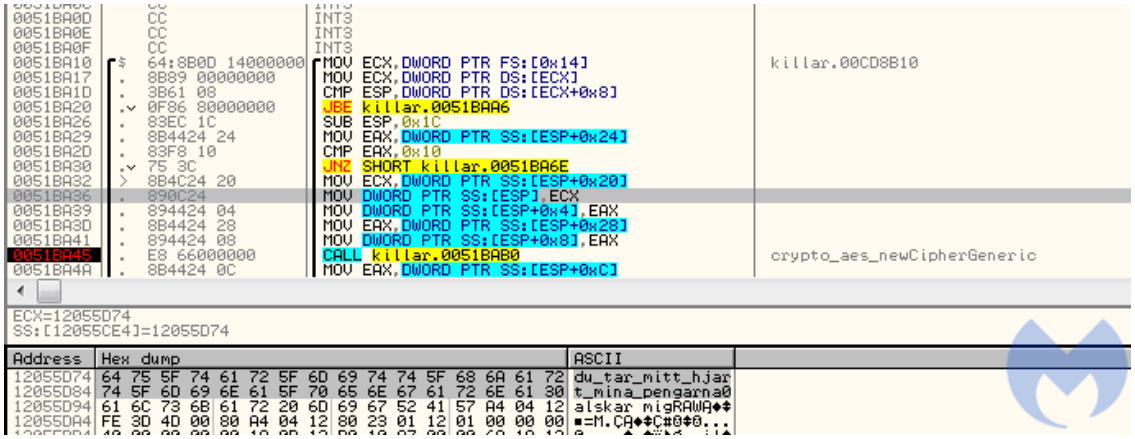
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

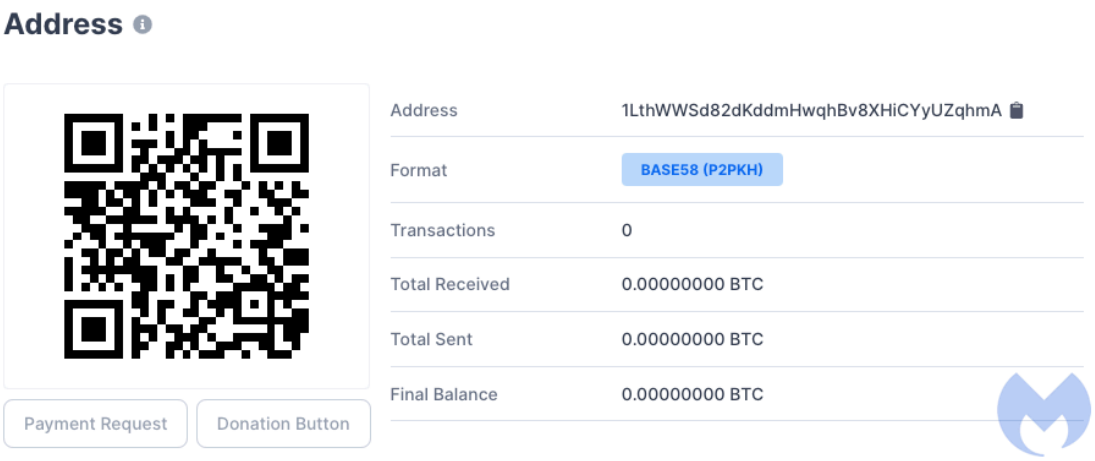
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

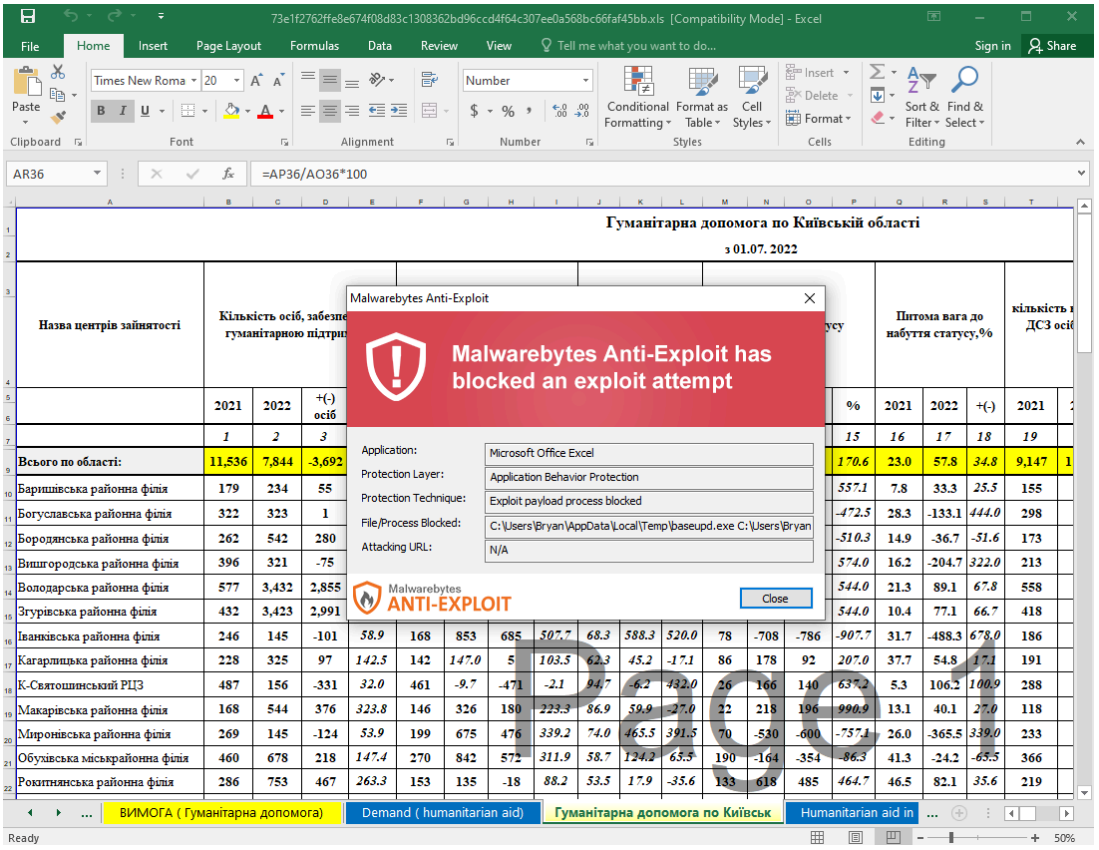
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

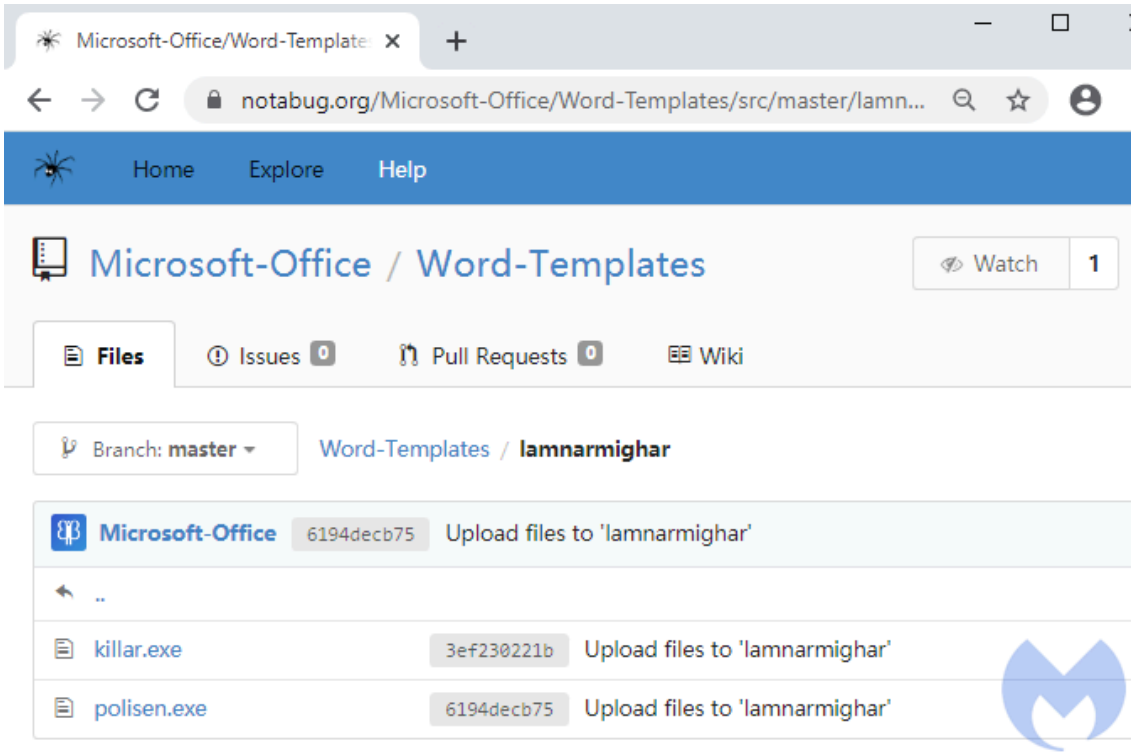
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



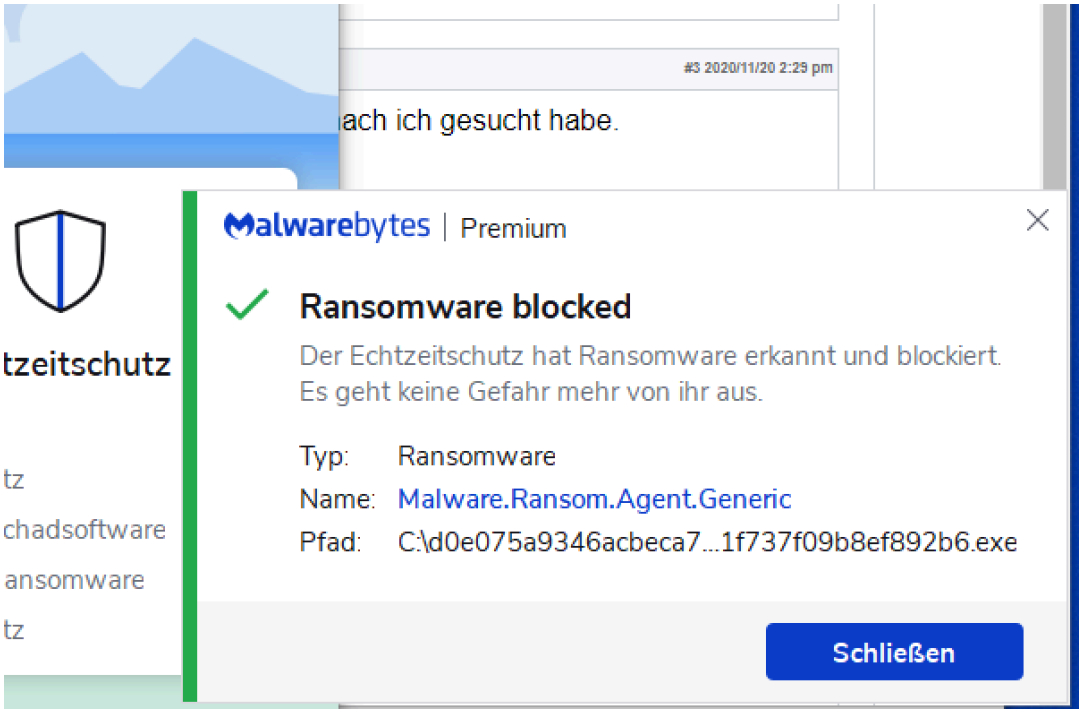
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

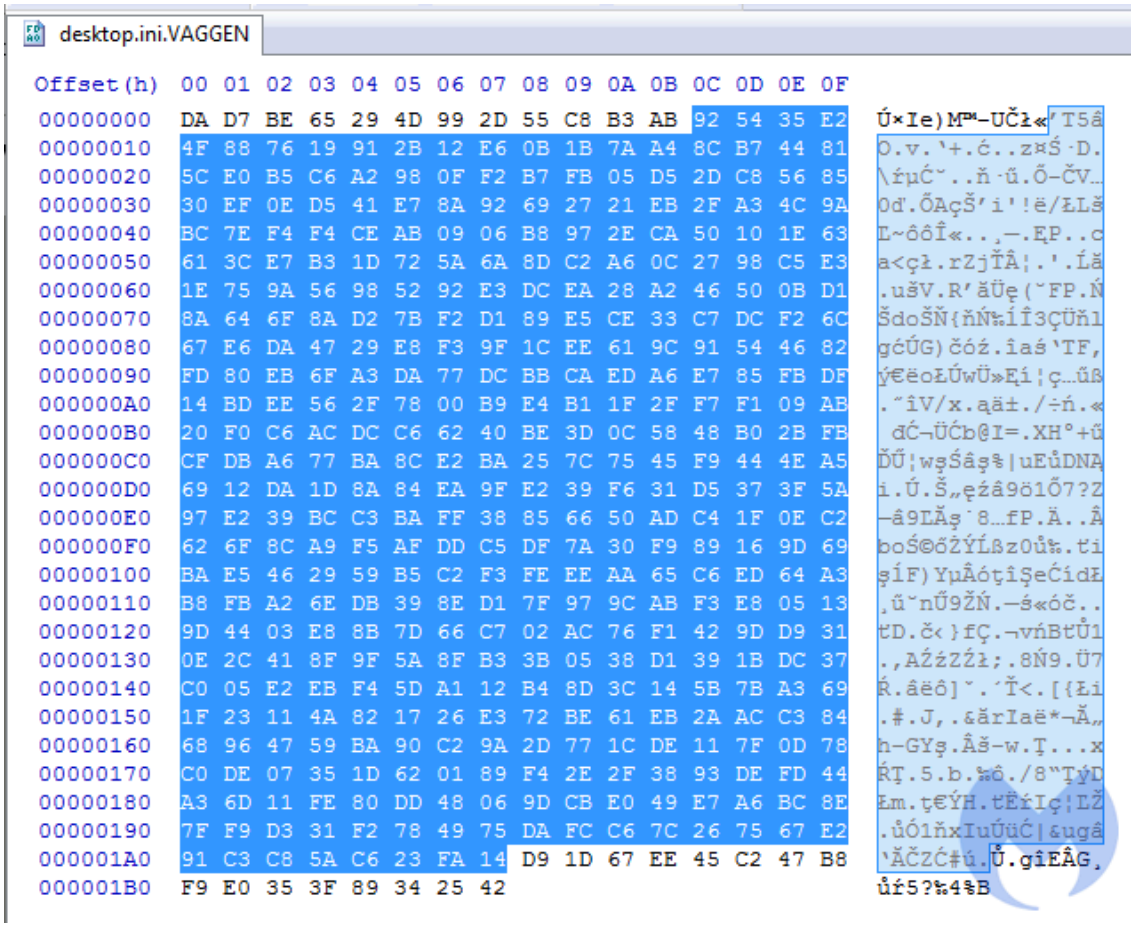
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

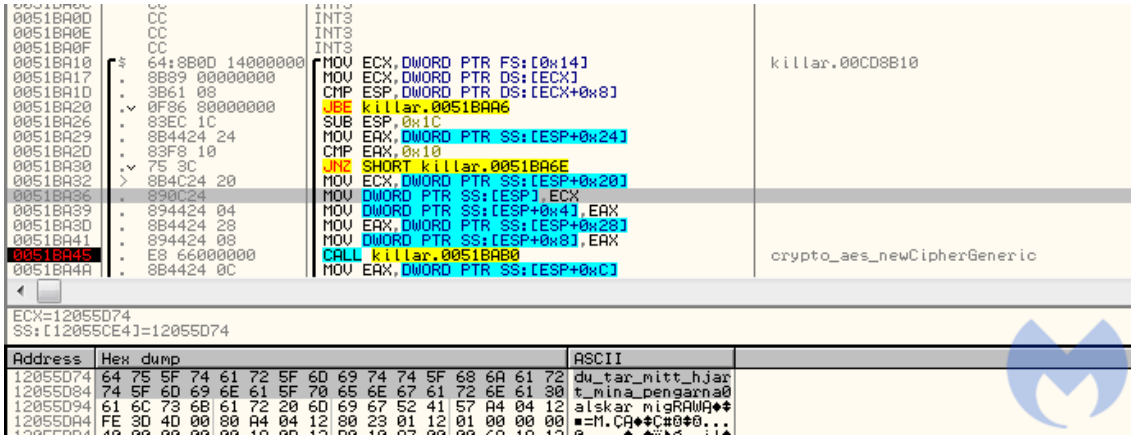


The content of the output file (with .VAGGEN extension) contains:

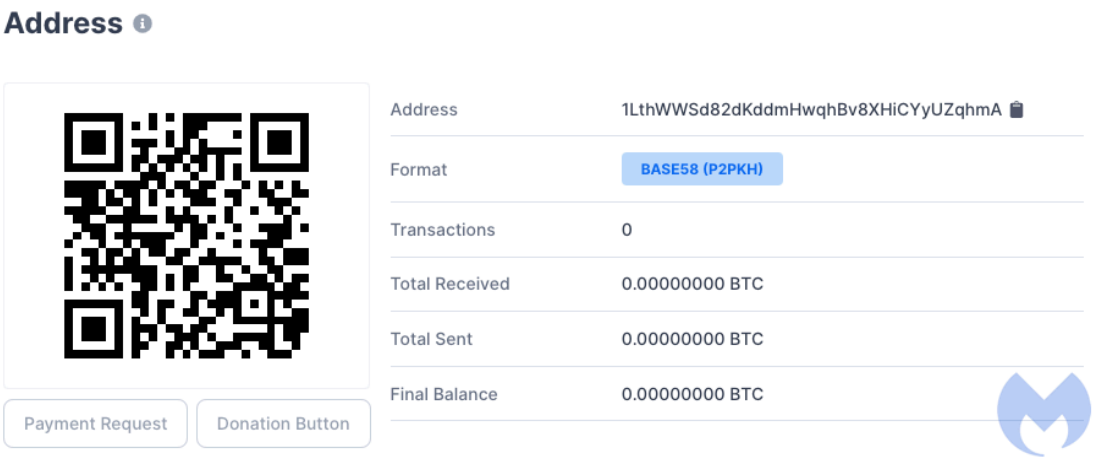
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

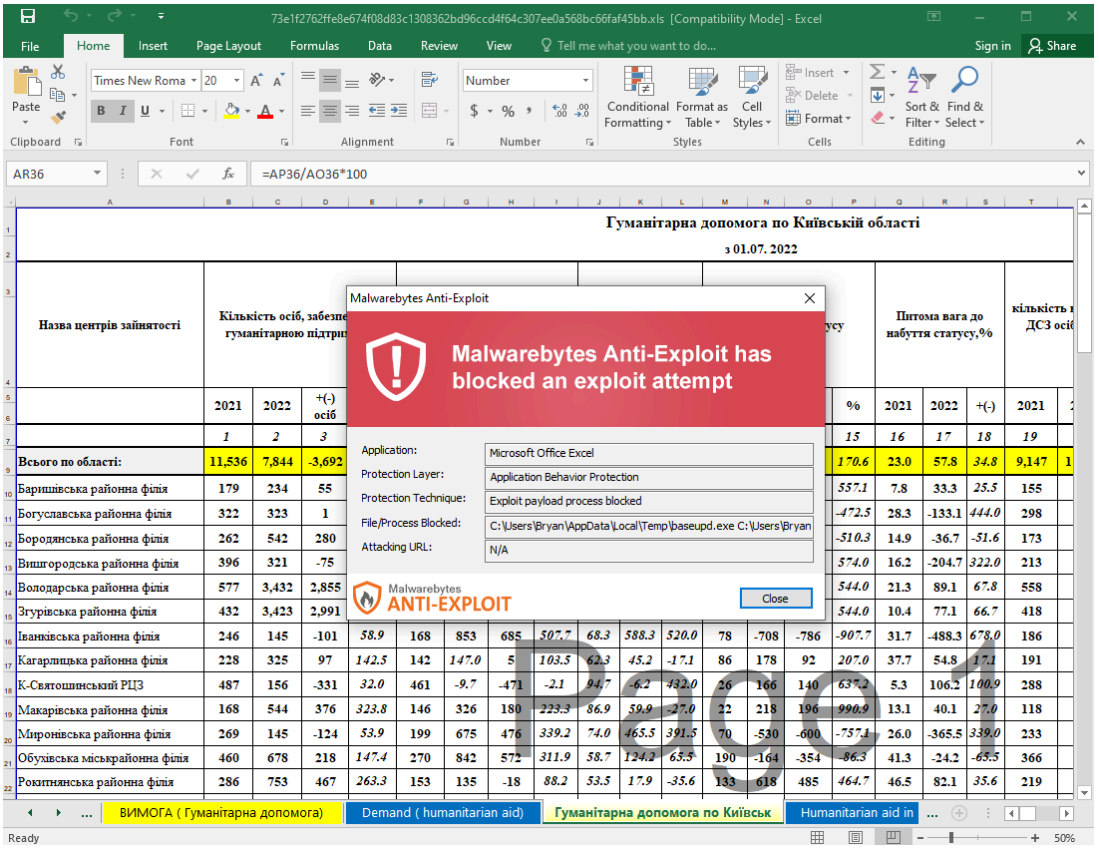
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

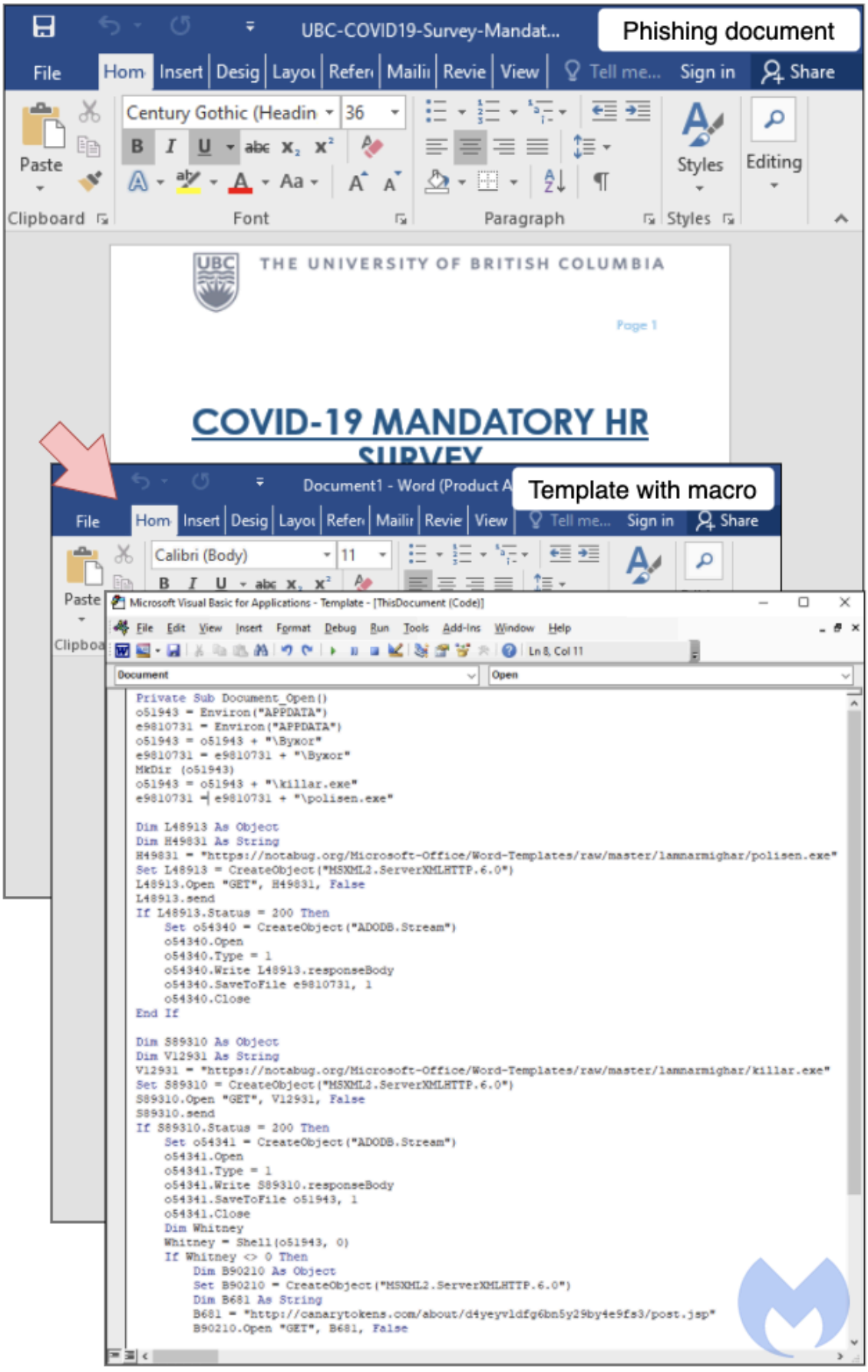
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

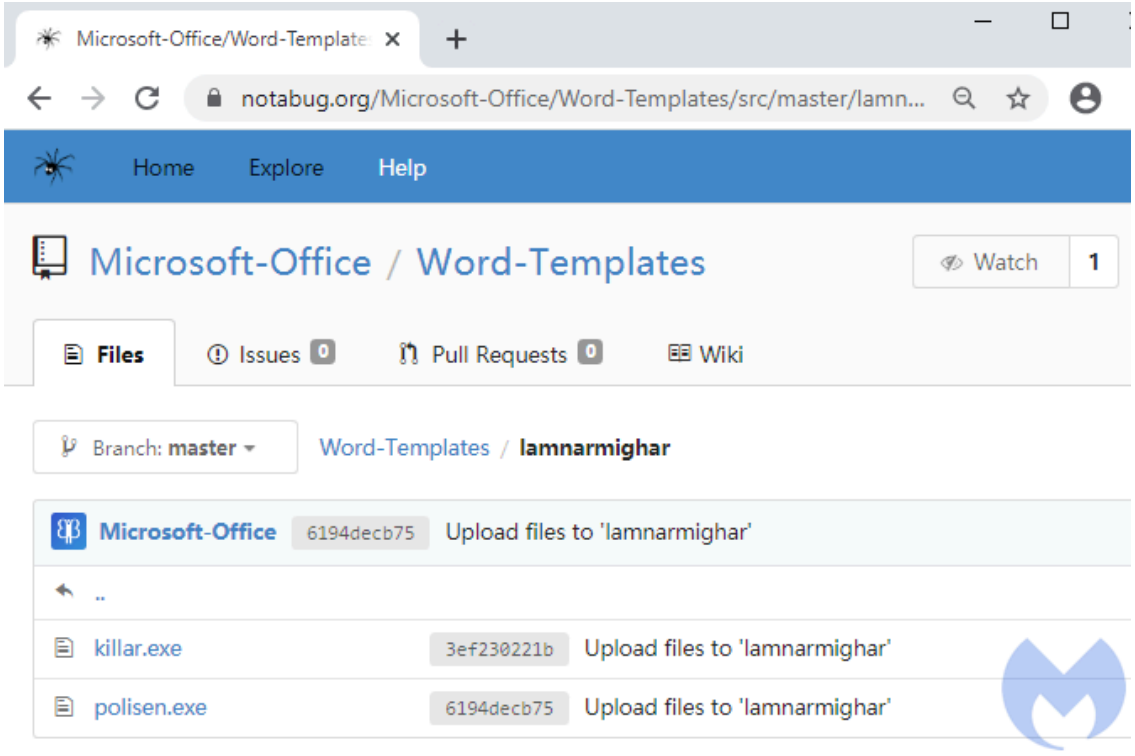
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



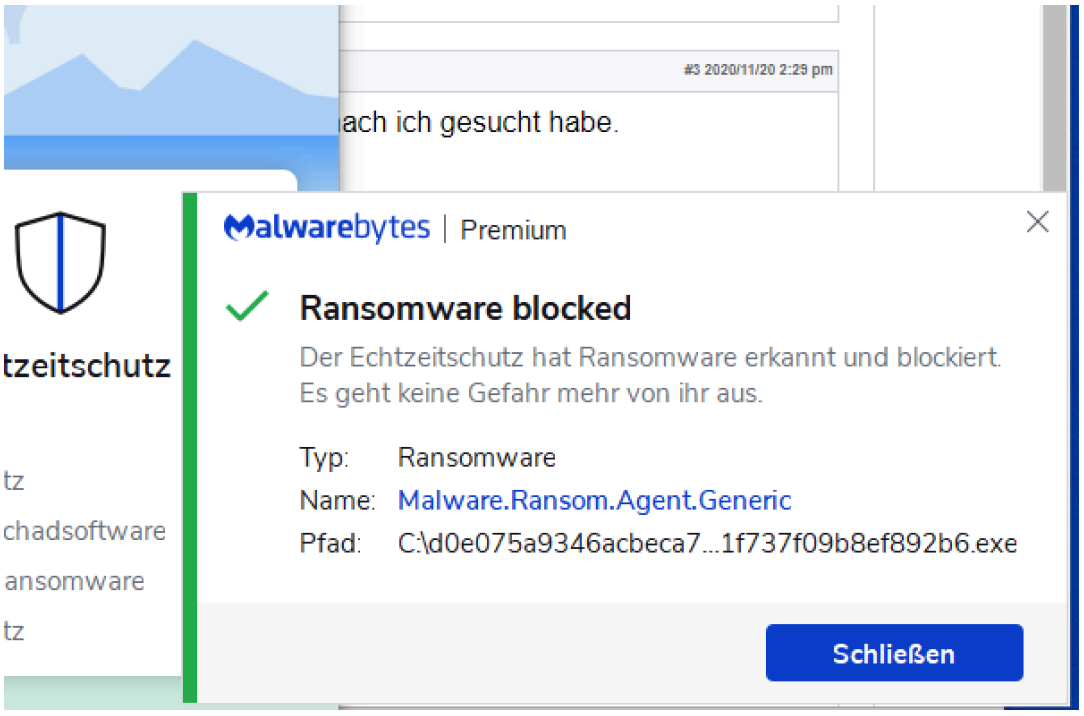
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

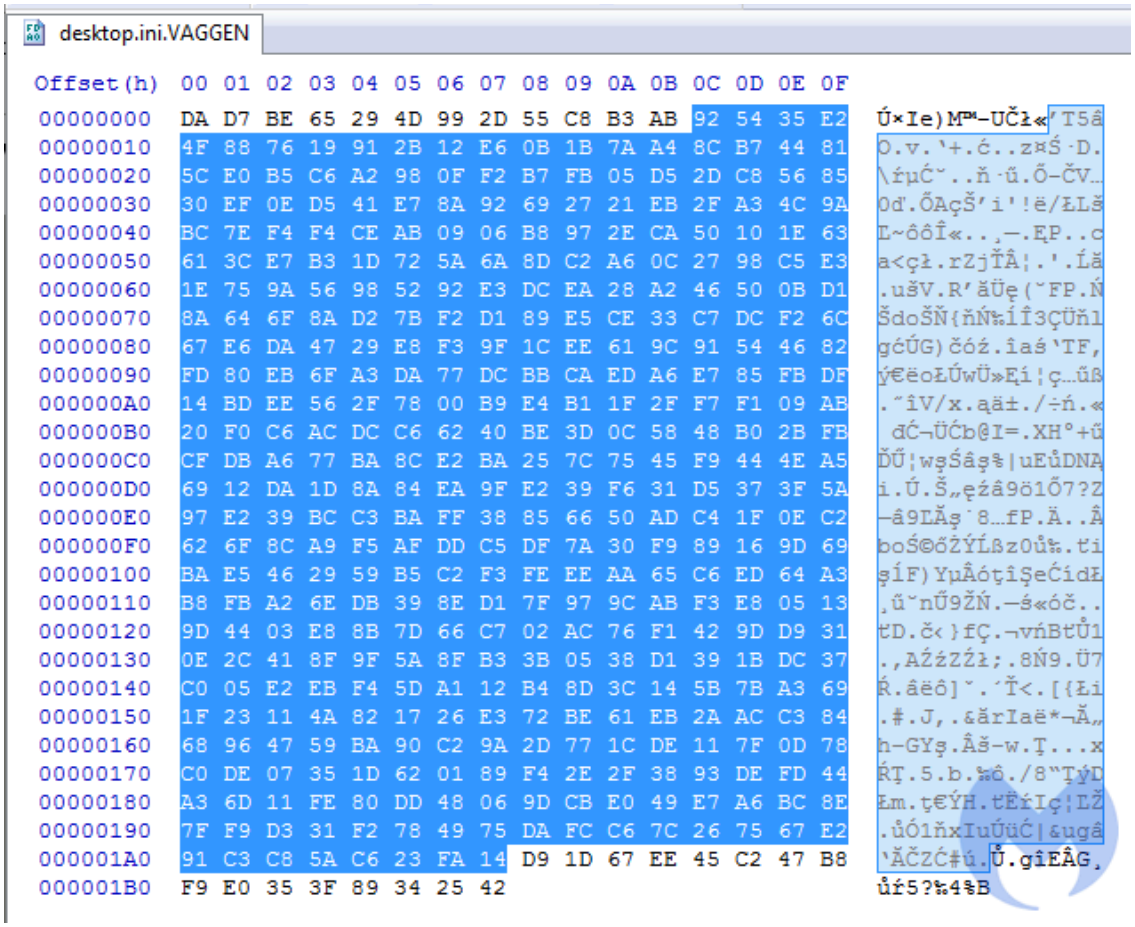
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



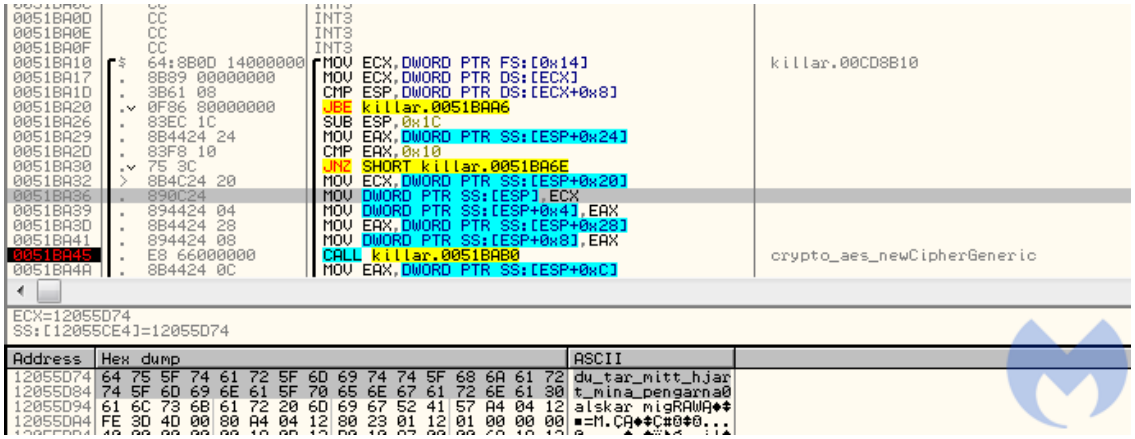
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

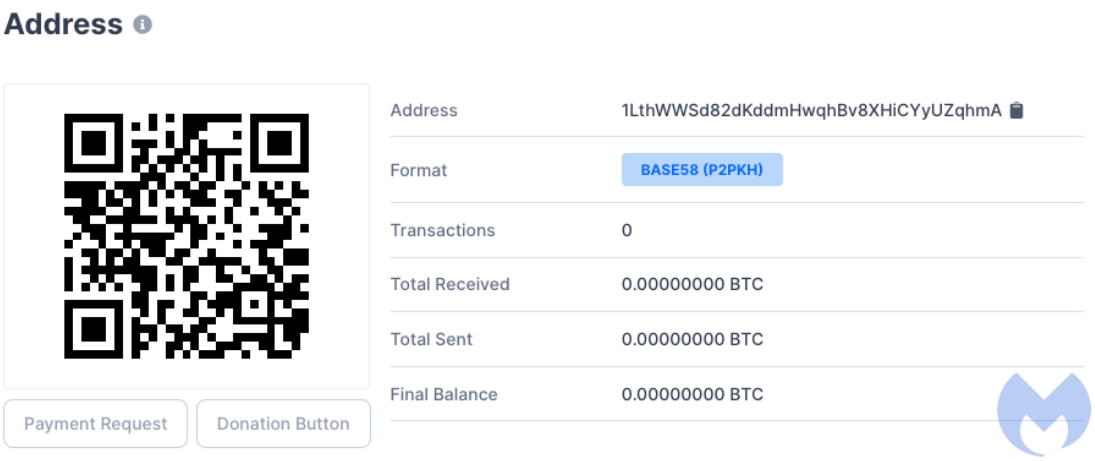
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

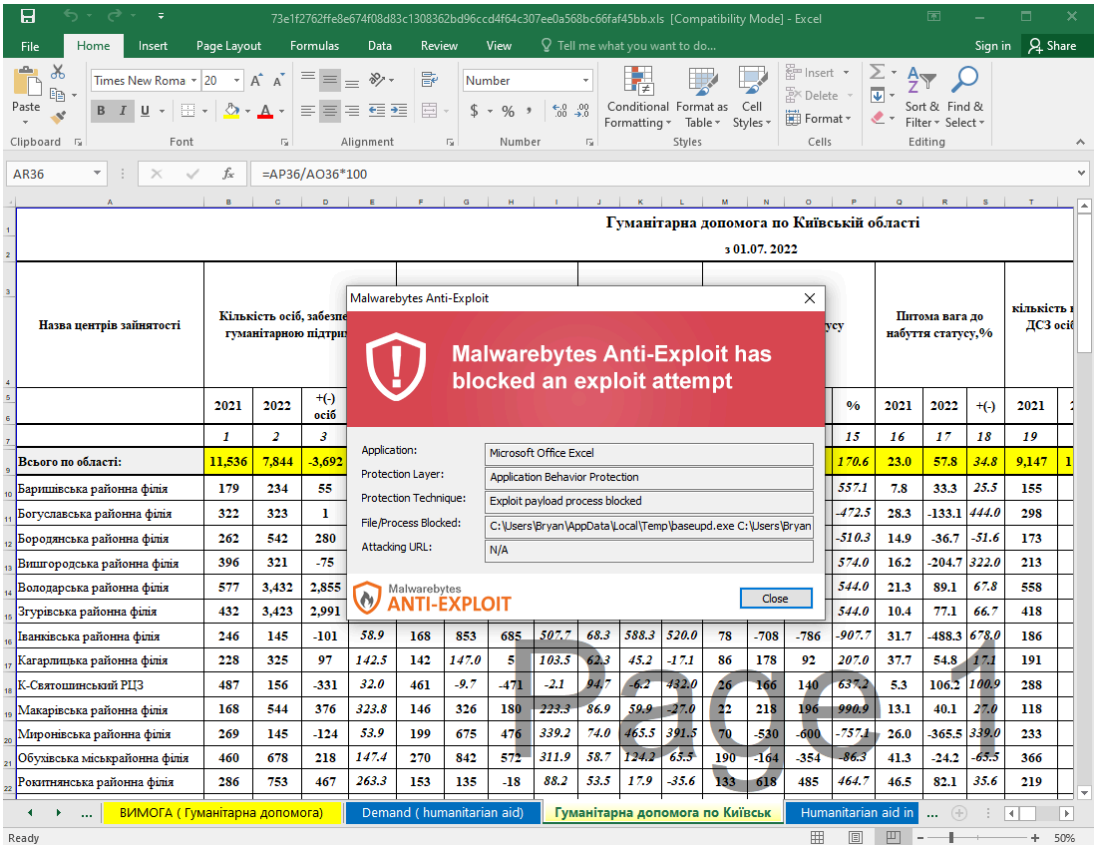
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

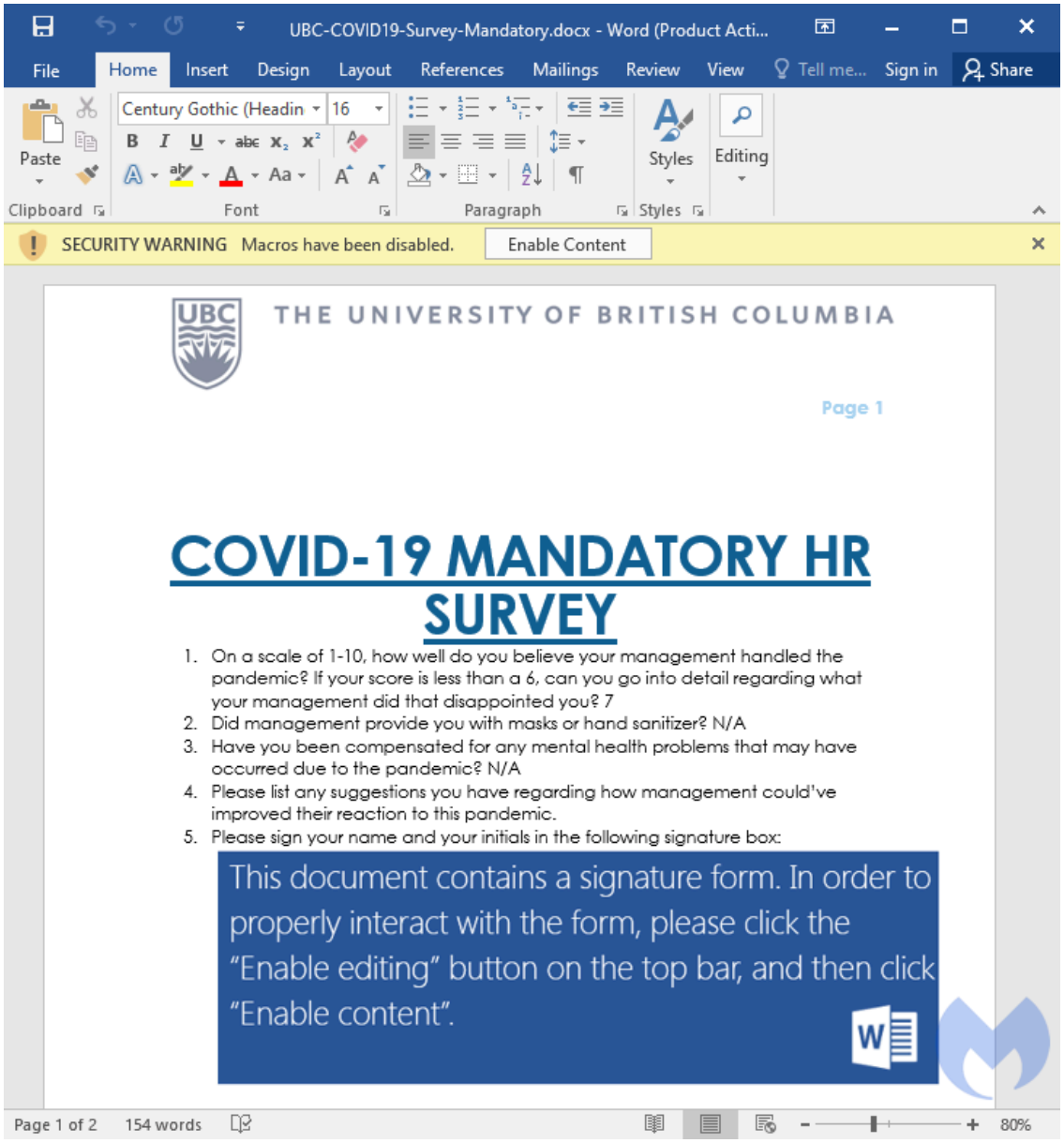
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

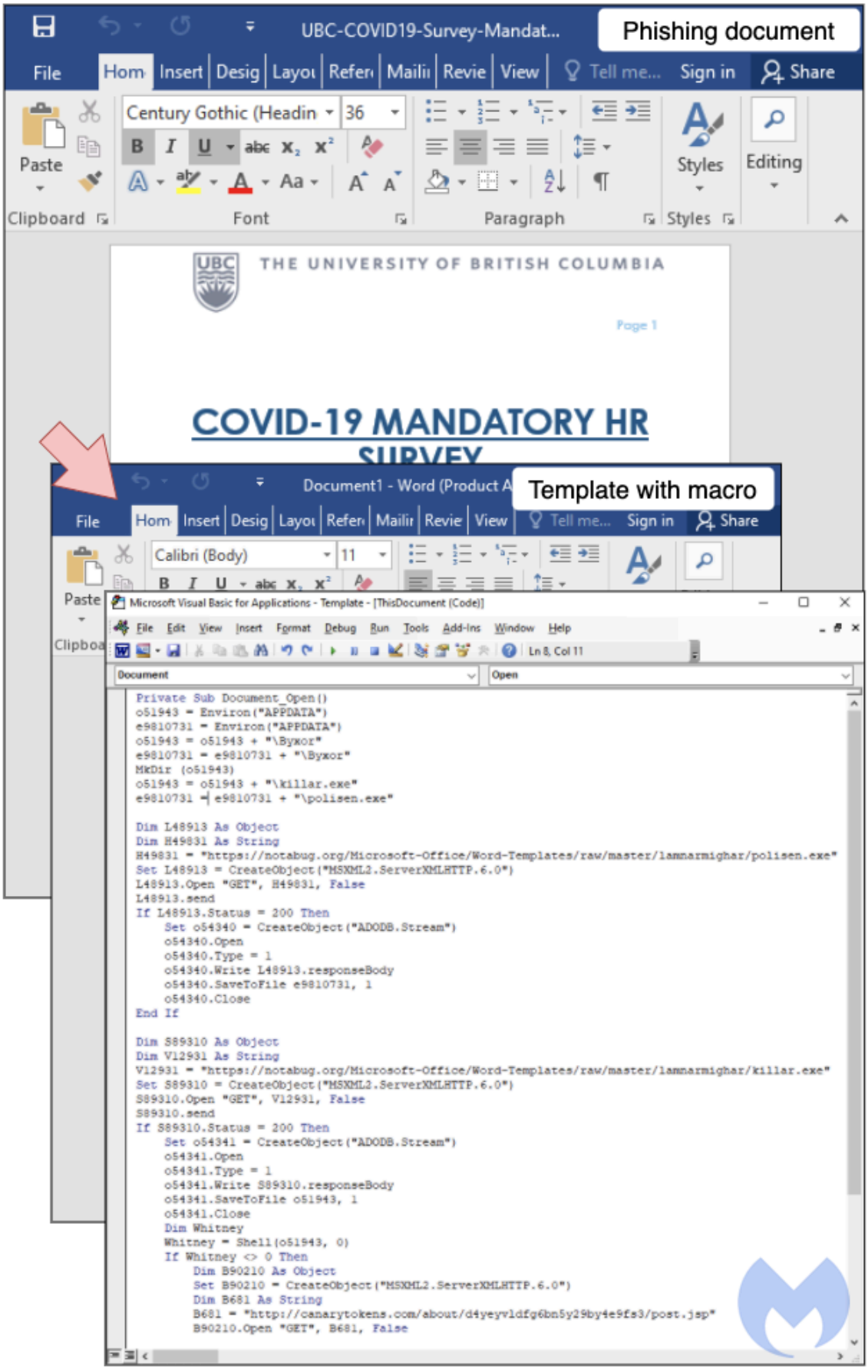
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

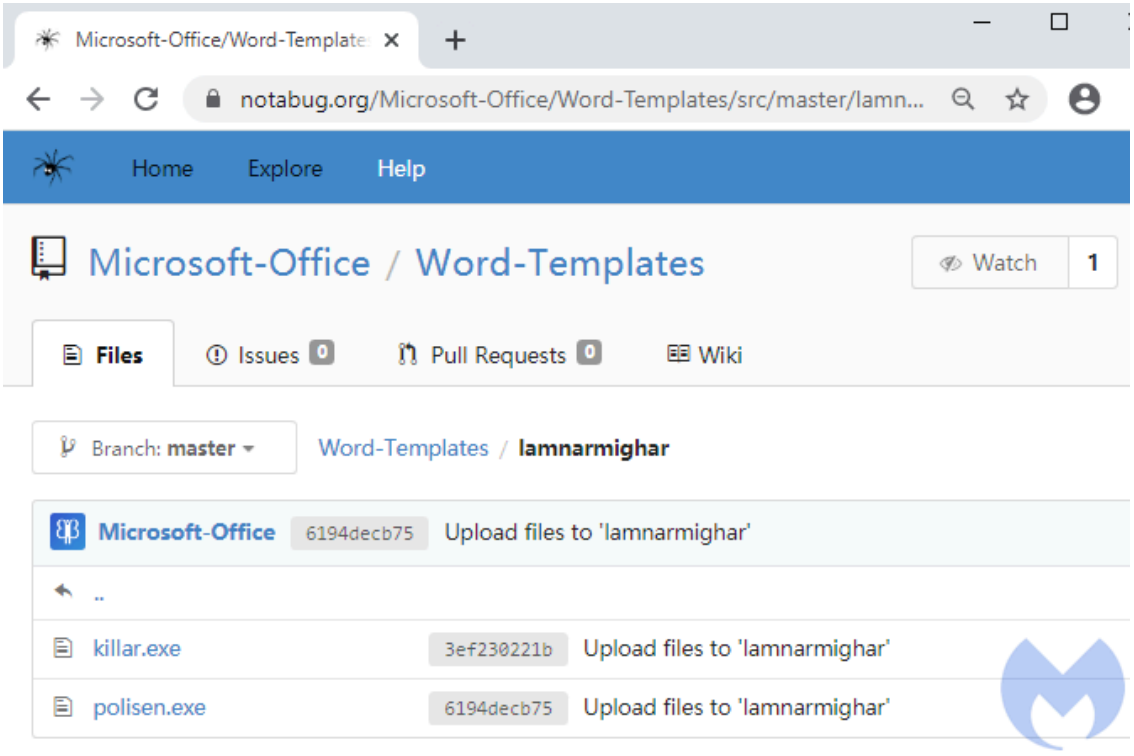
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



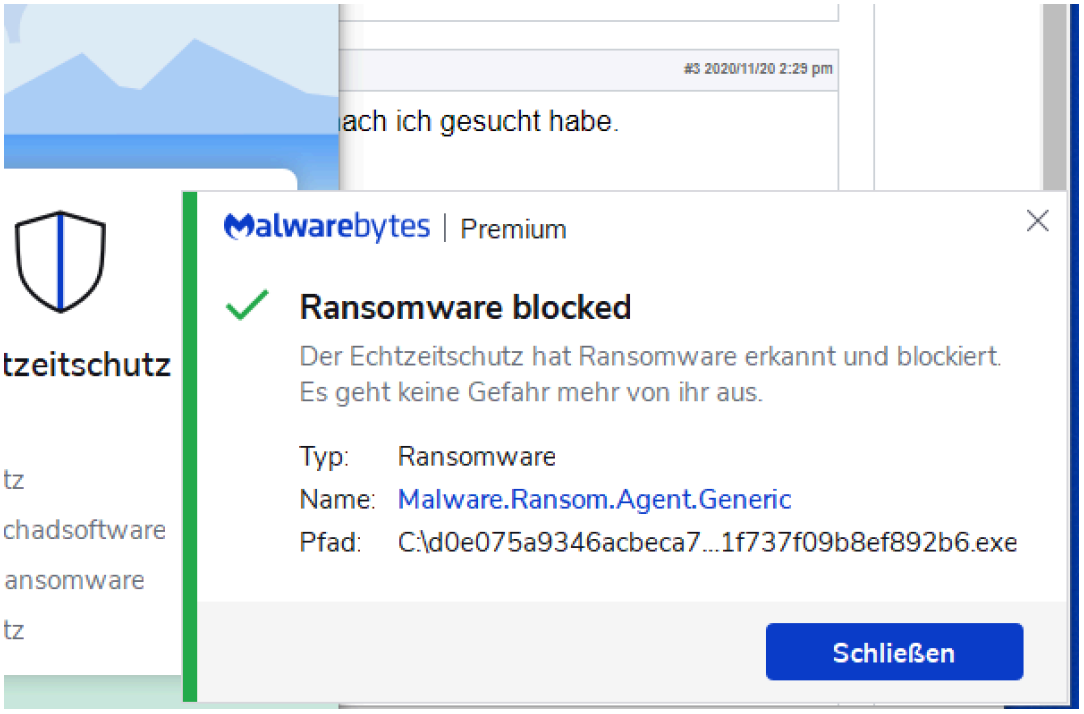
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

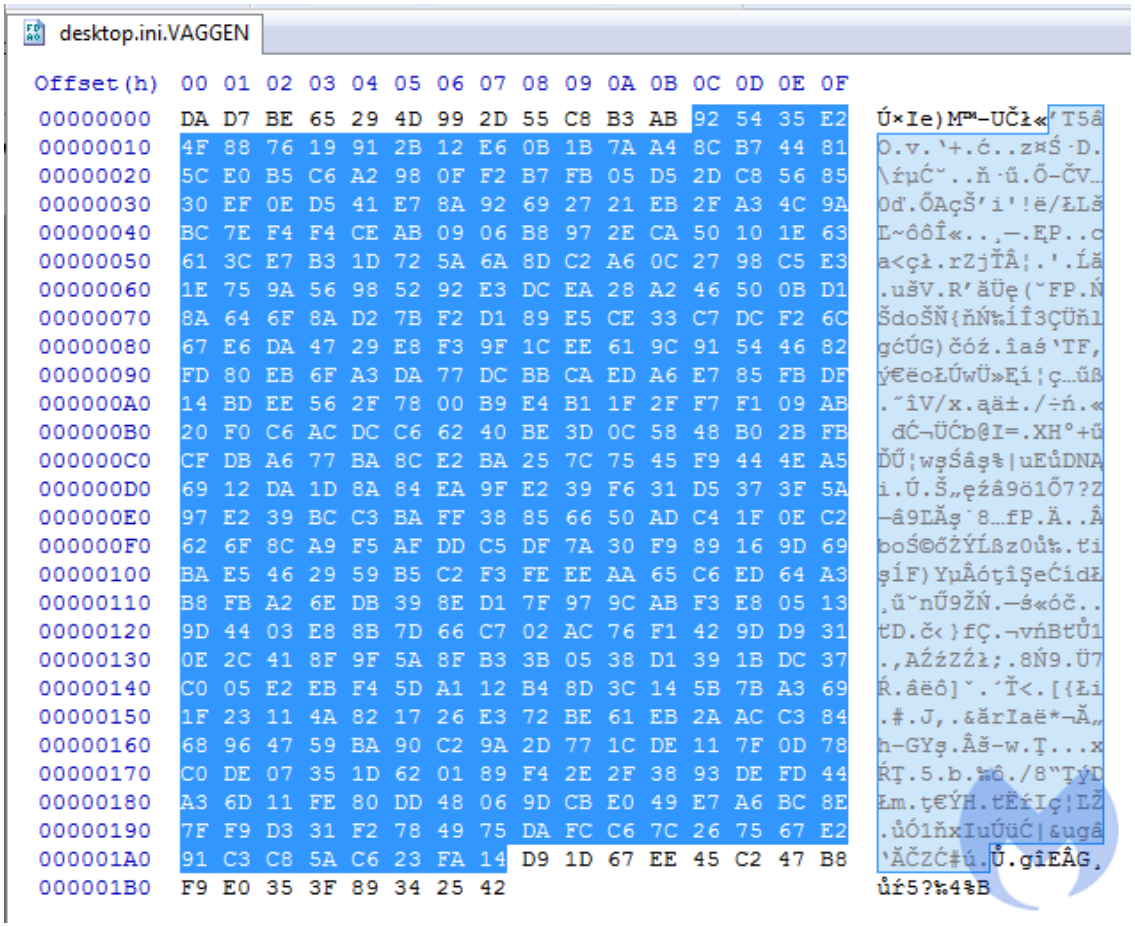
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



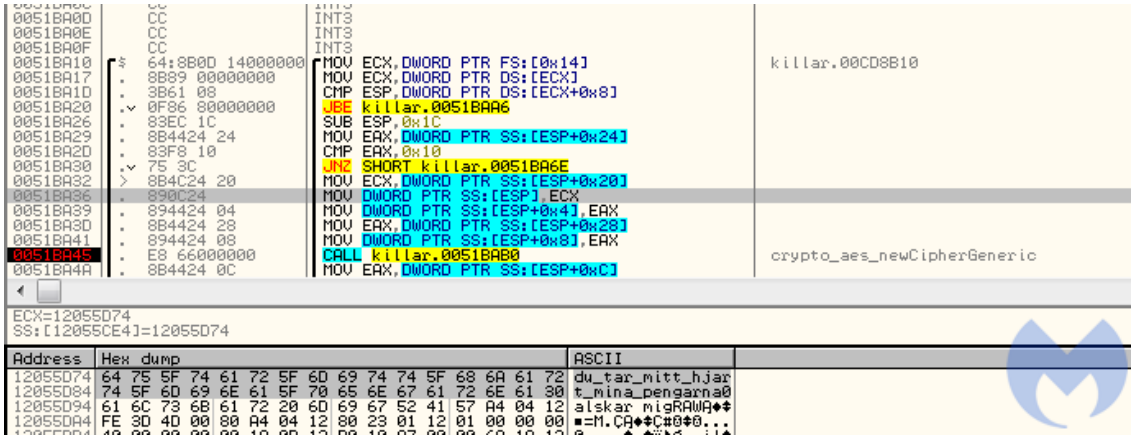
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

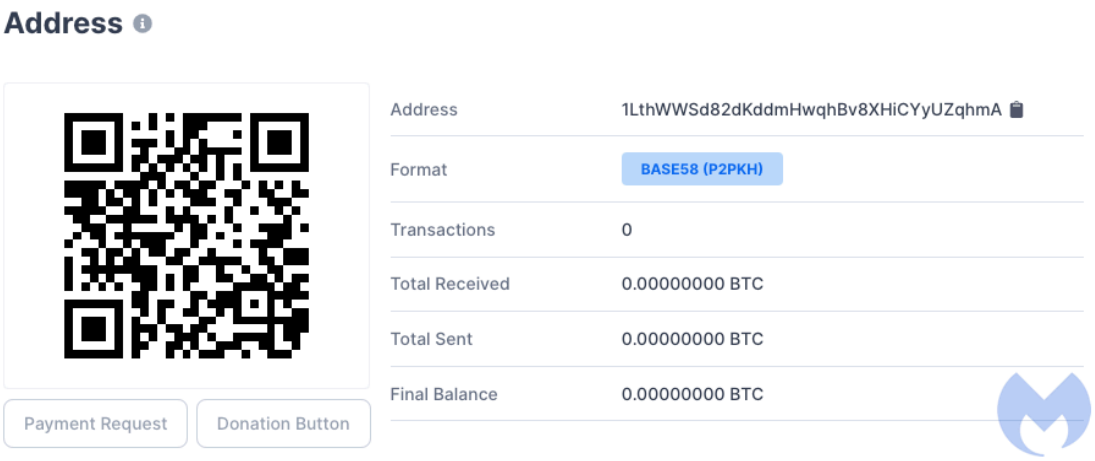
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

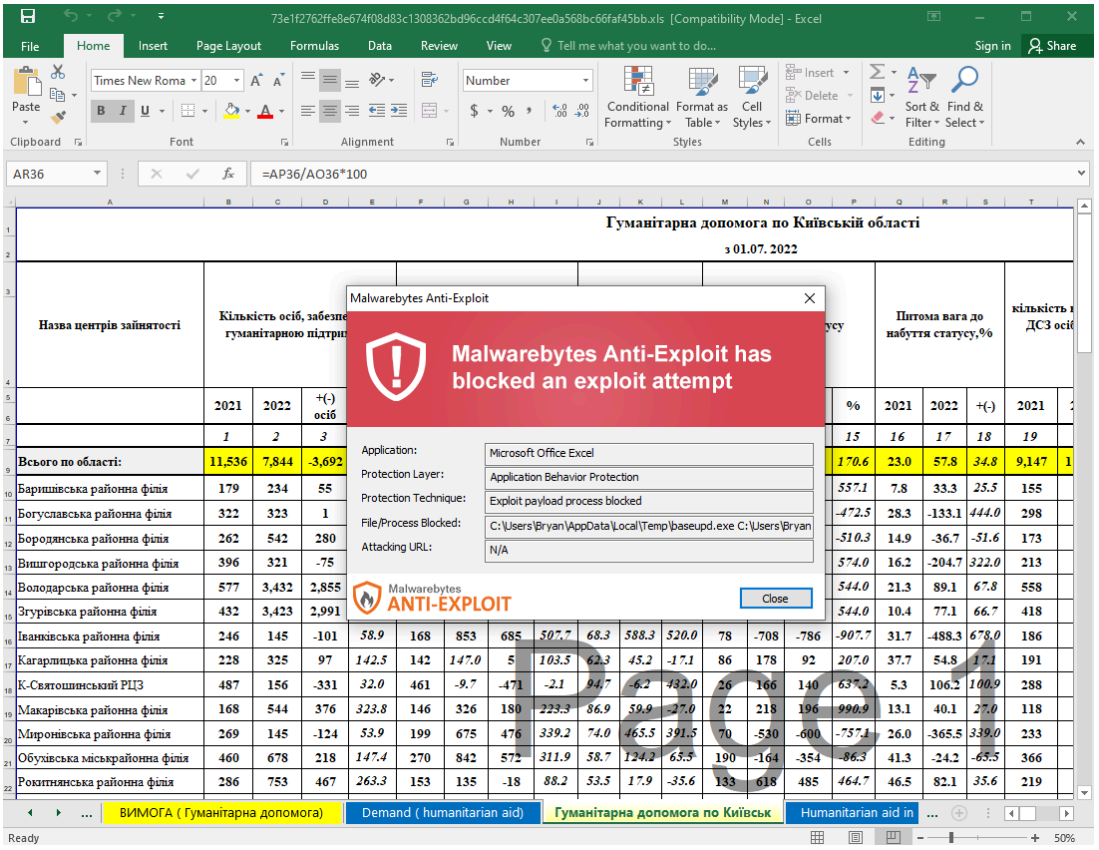
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

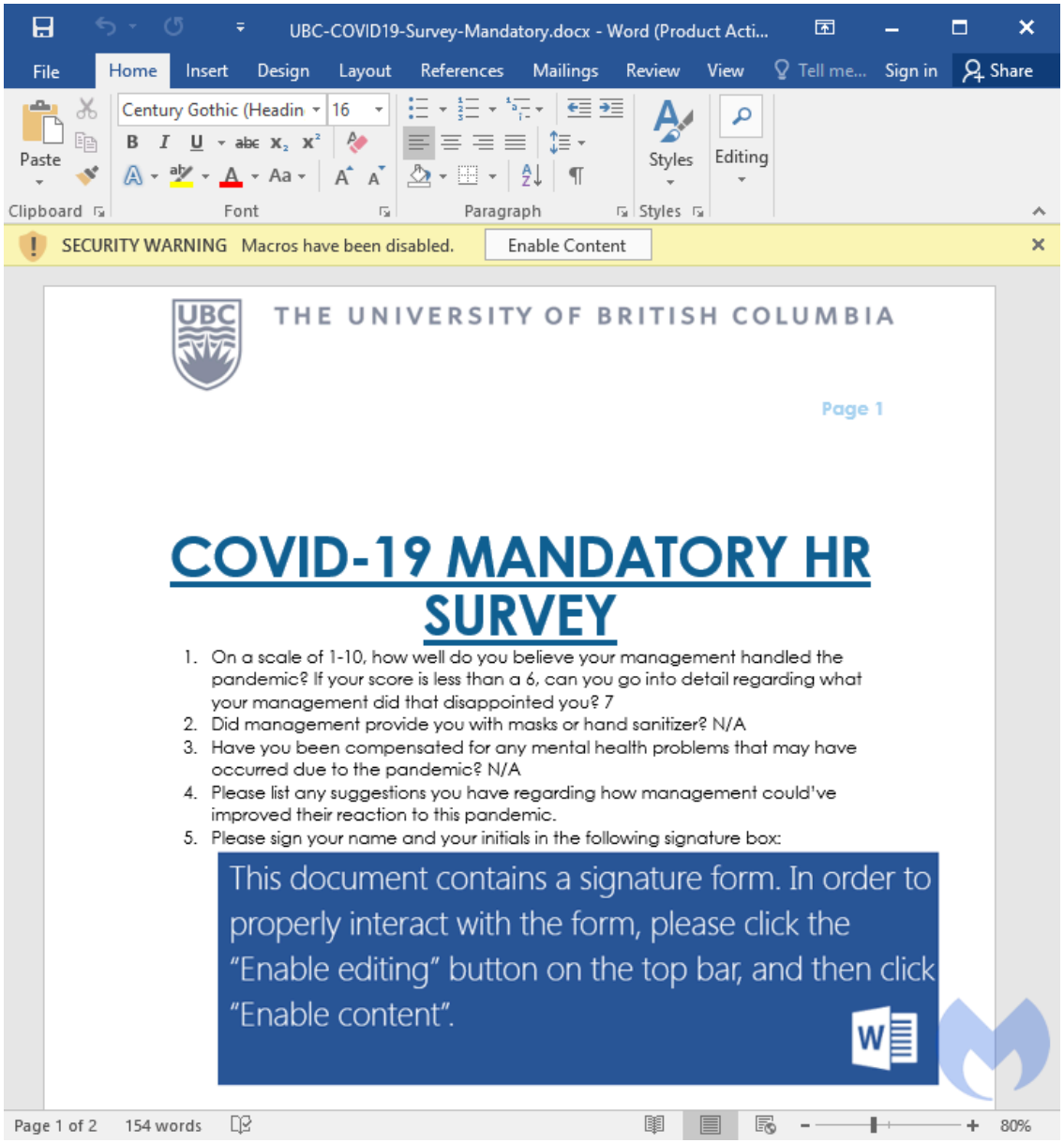
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

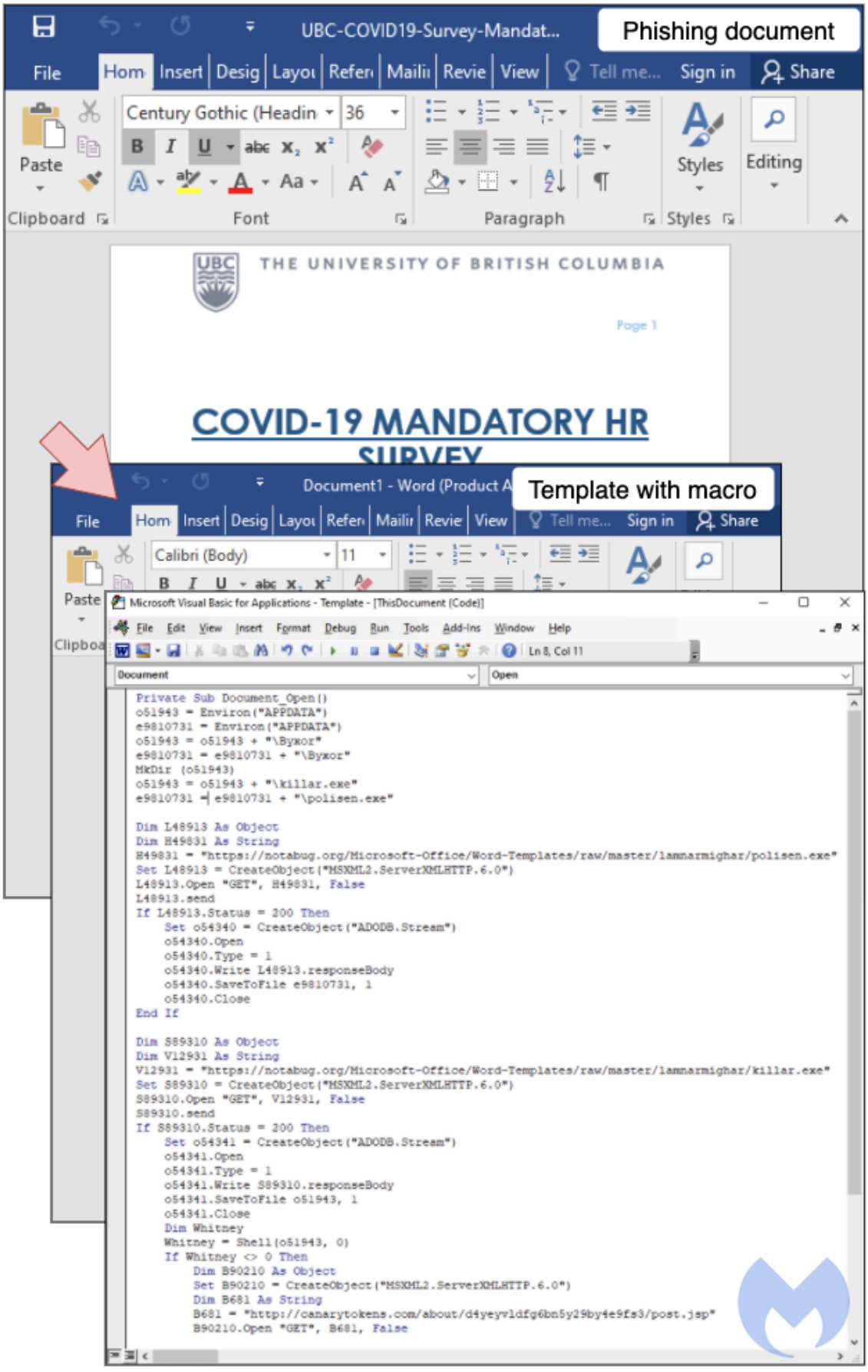
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

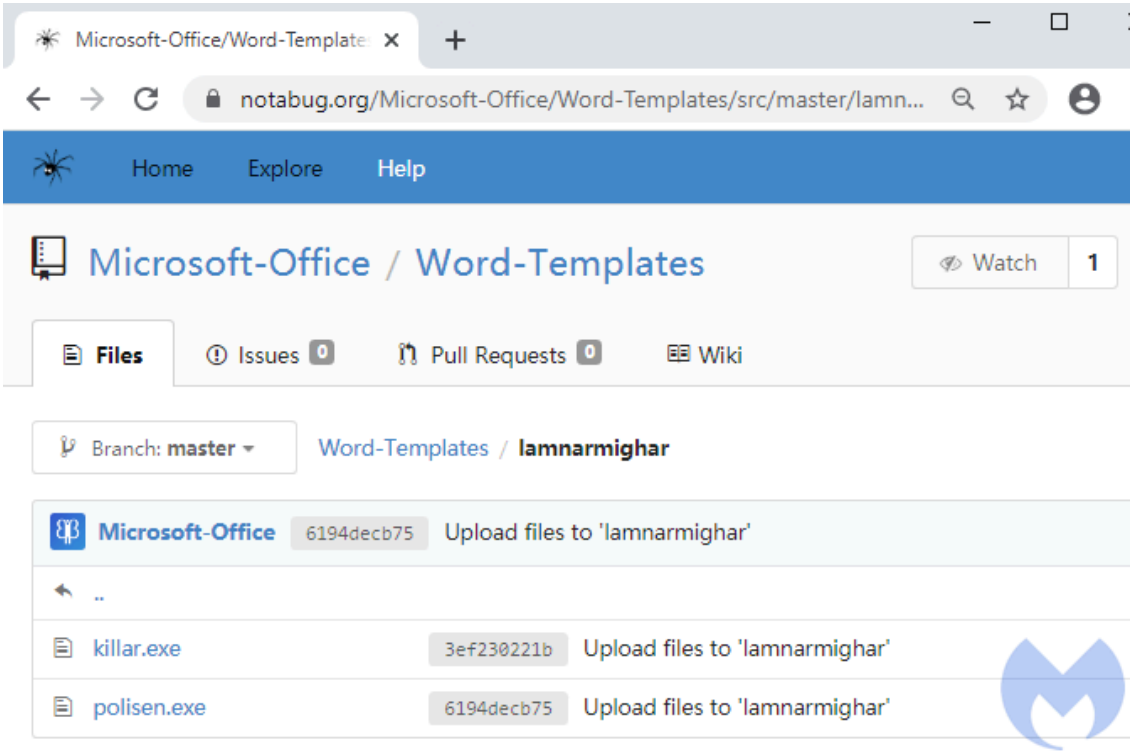
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



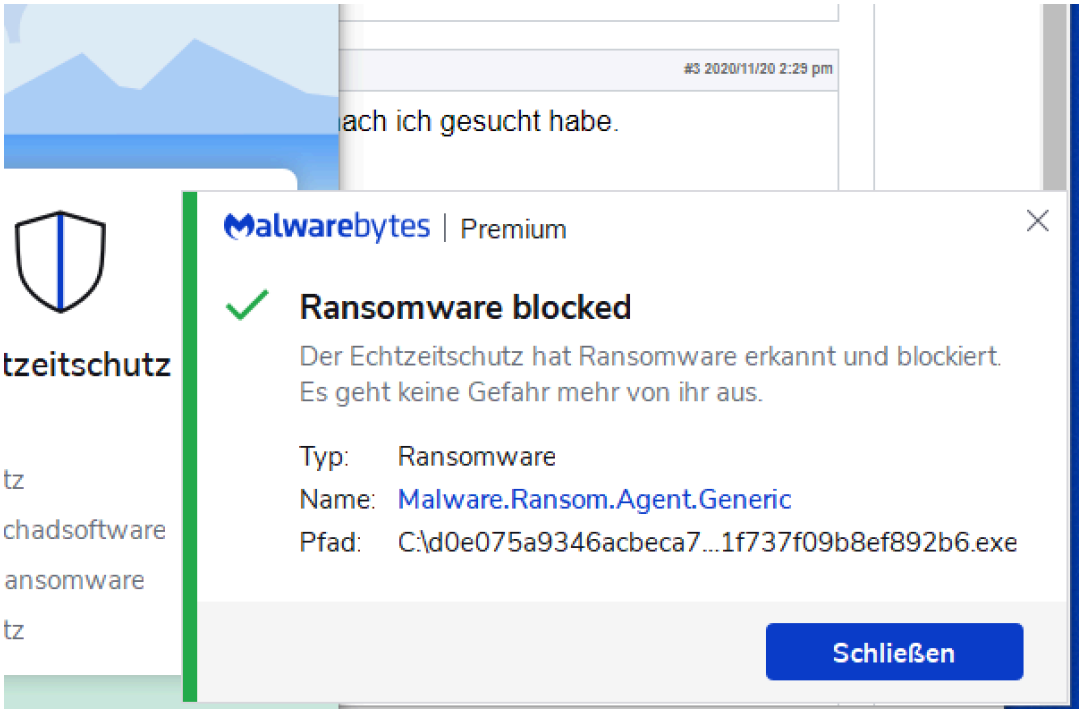
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

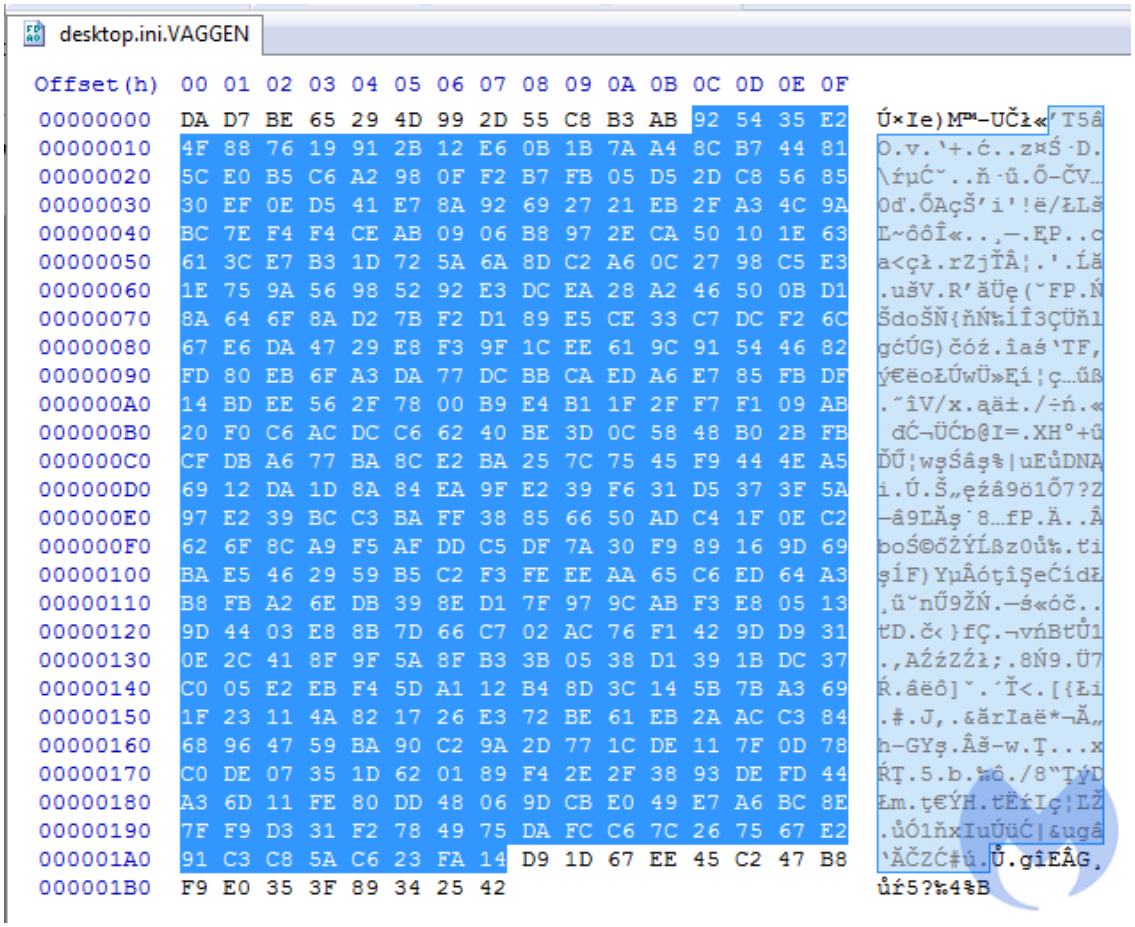
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

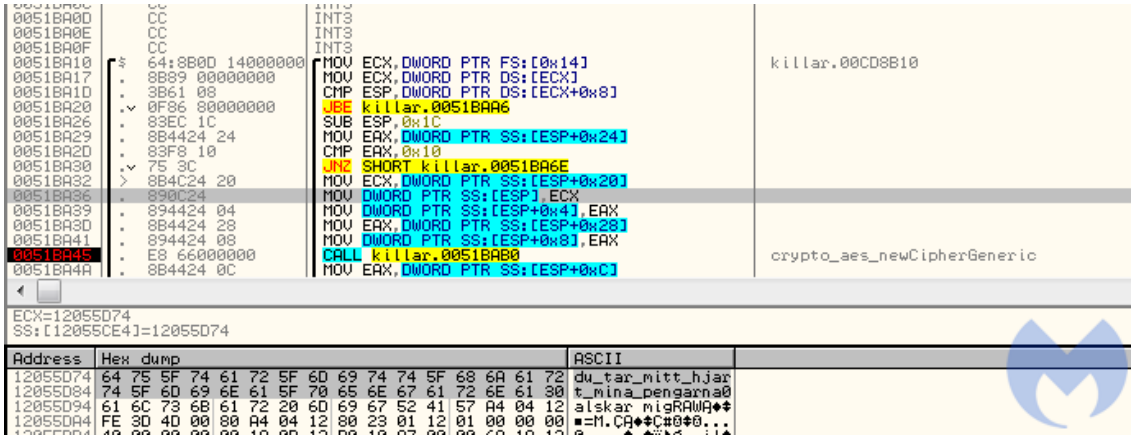


The content of the output file (with .VAGGEN extension) contains:

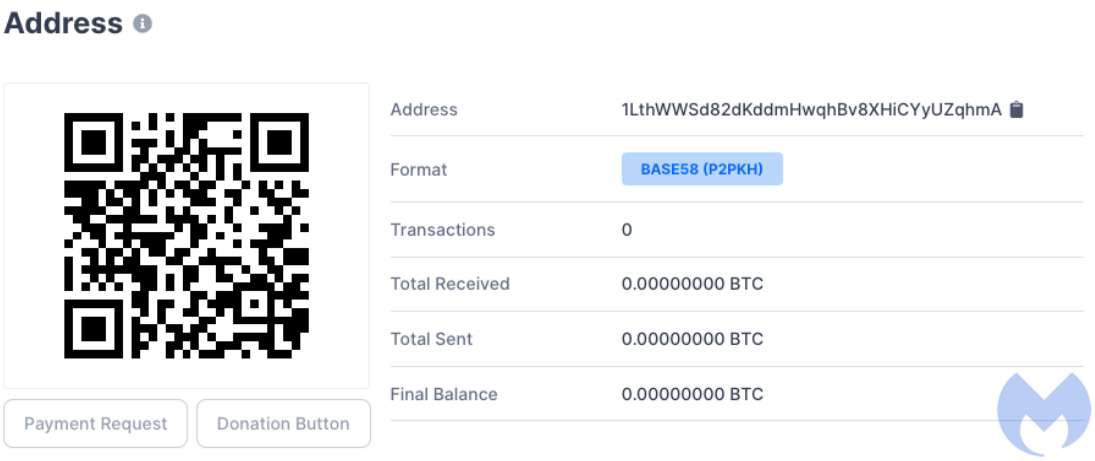
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

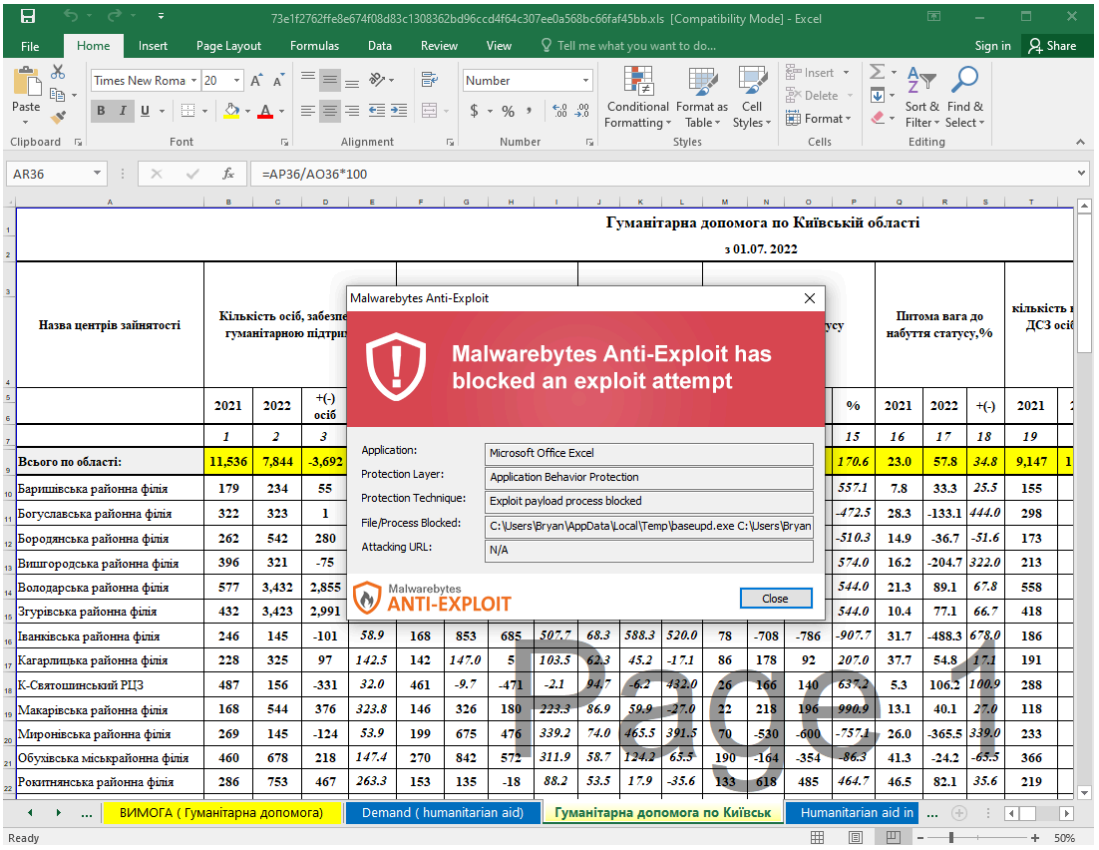
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

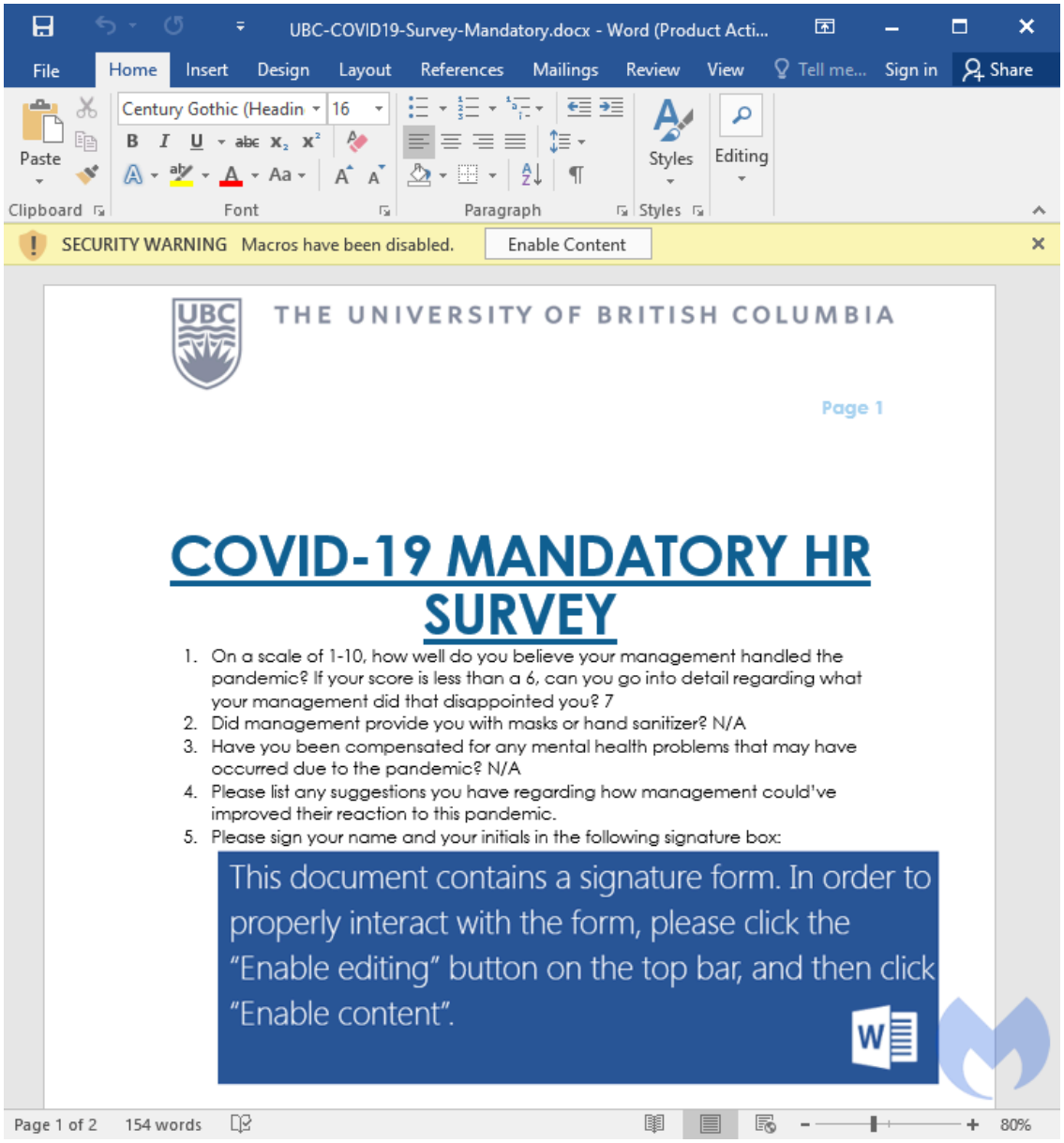
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

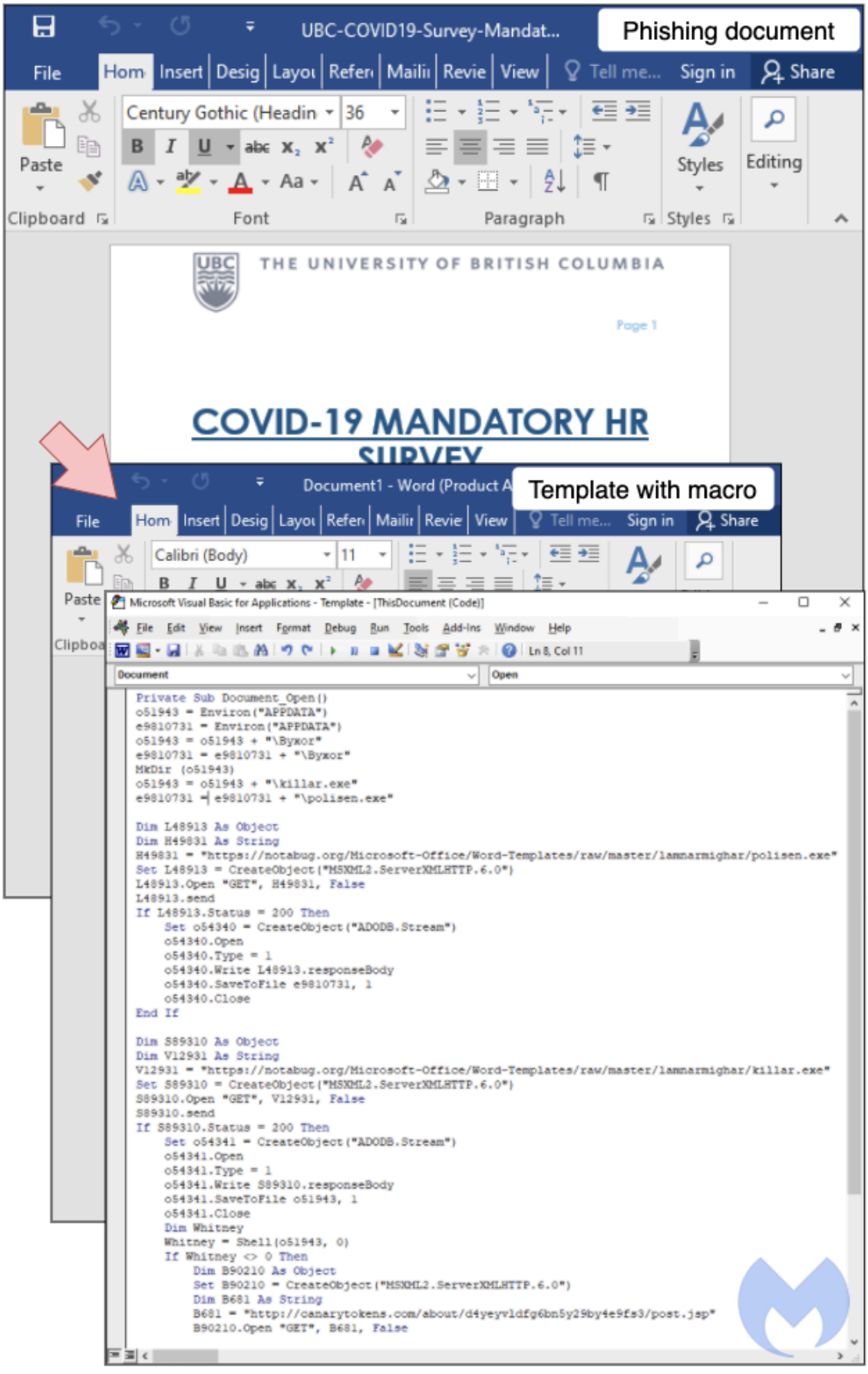
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

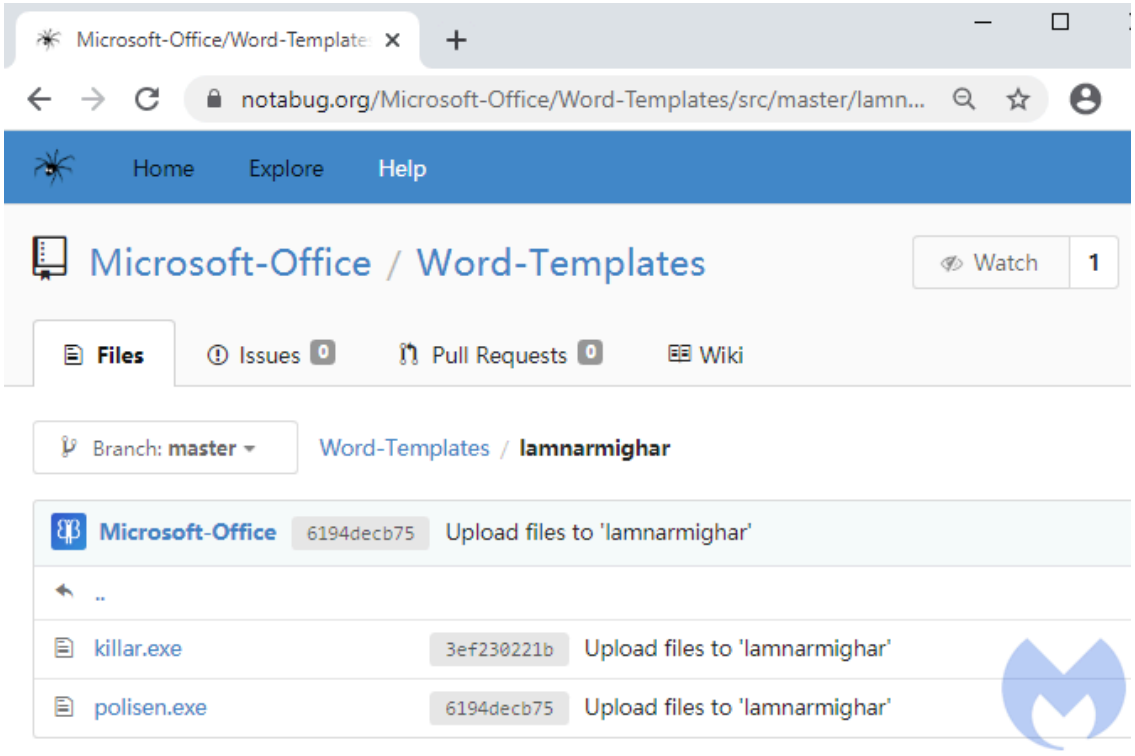
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



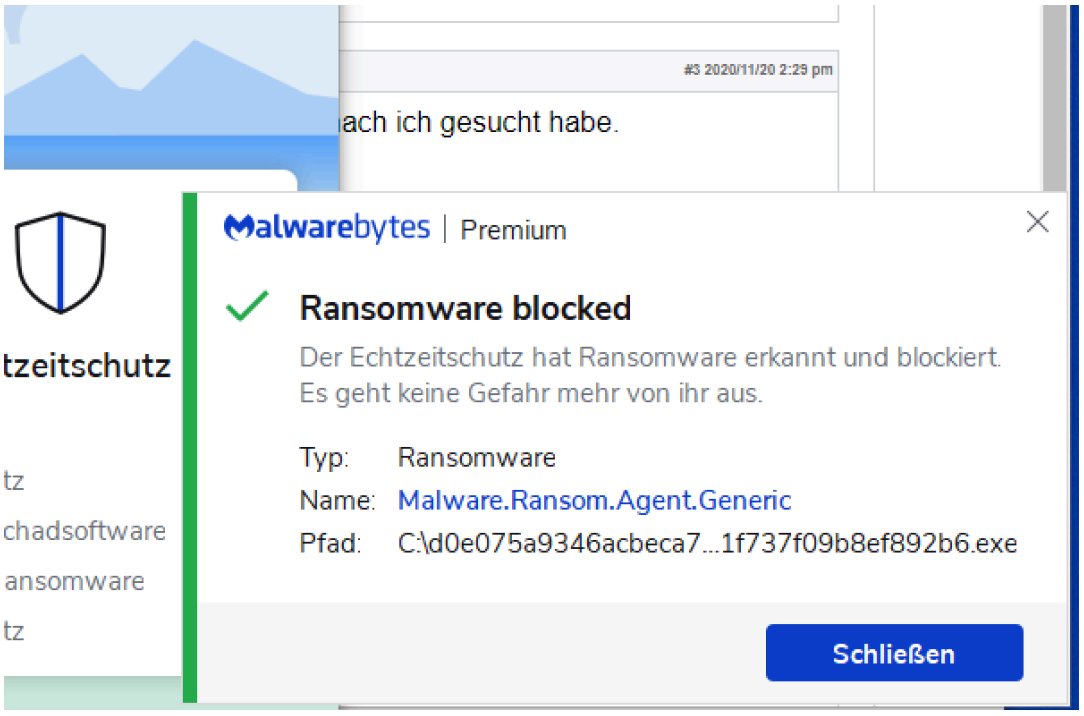
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

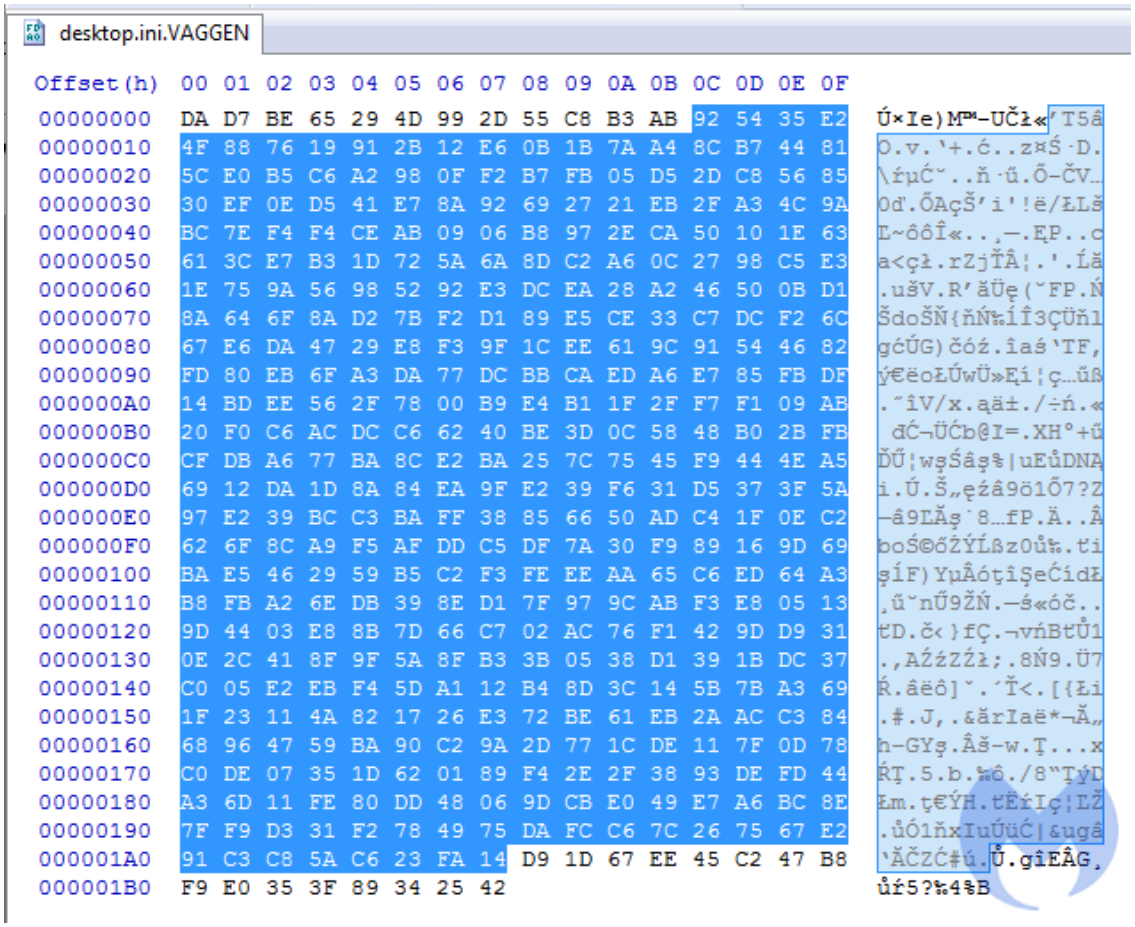
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

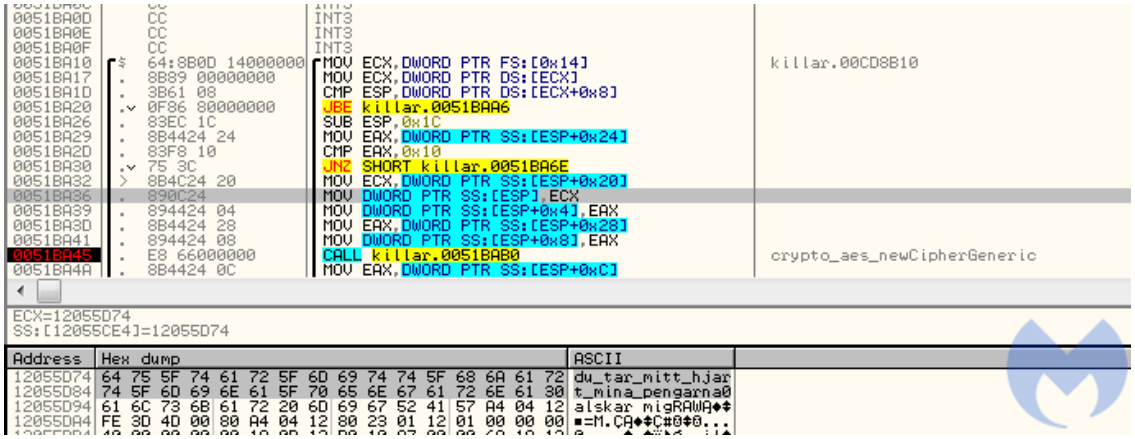


The content of the output file (with .VAGGEN extension) contains:

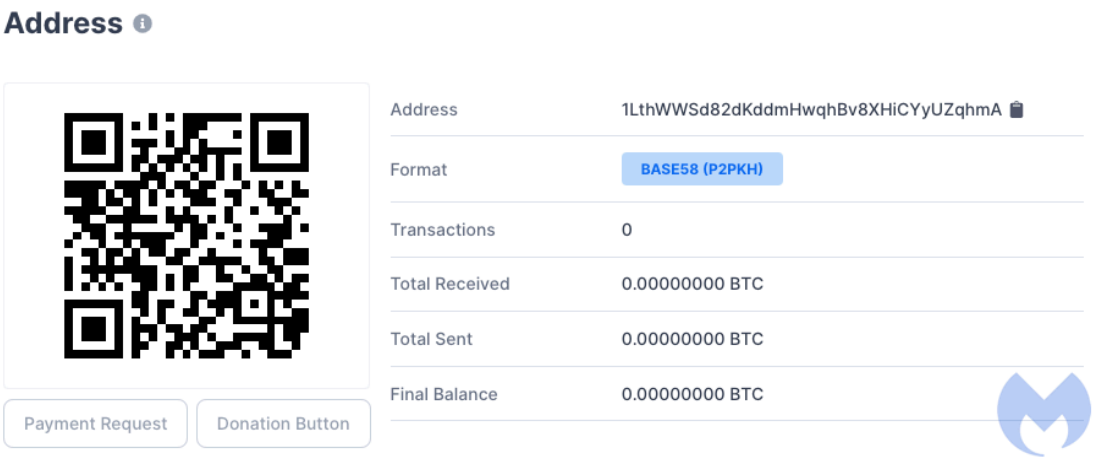
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

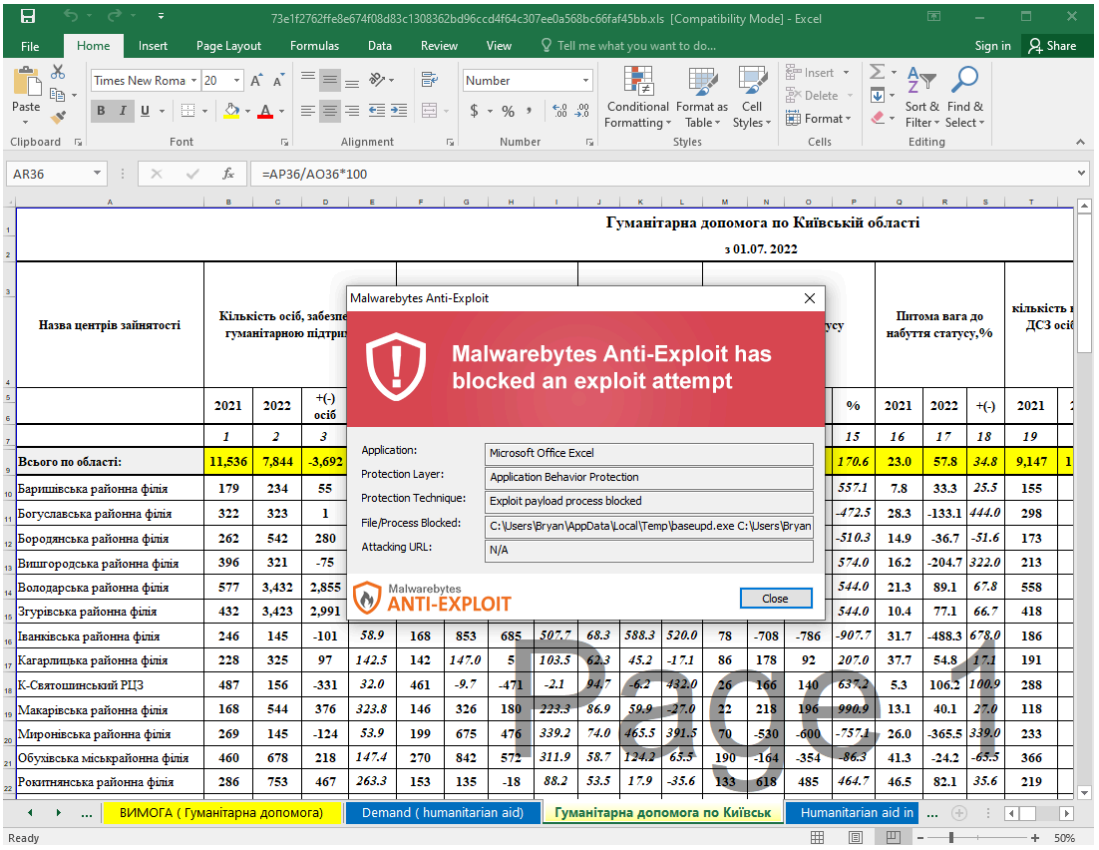
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

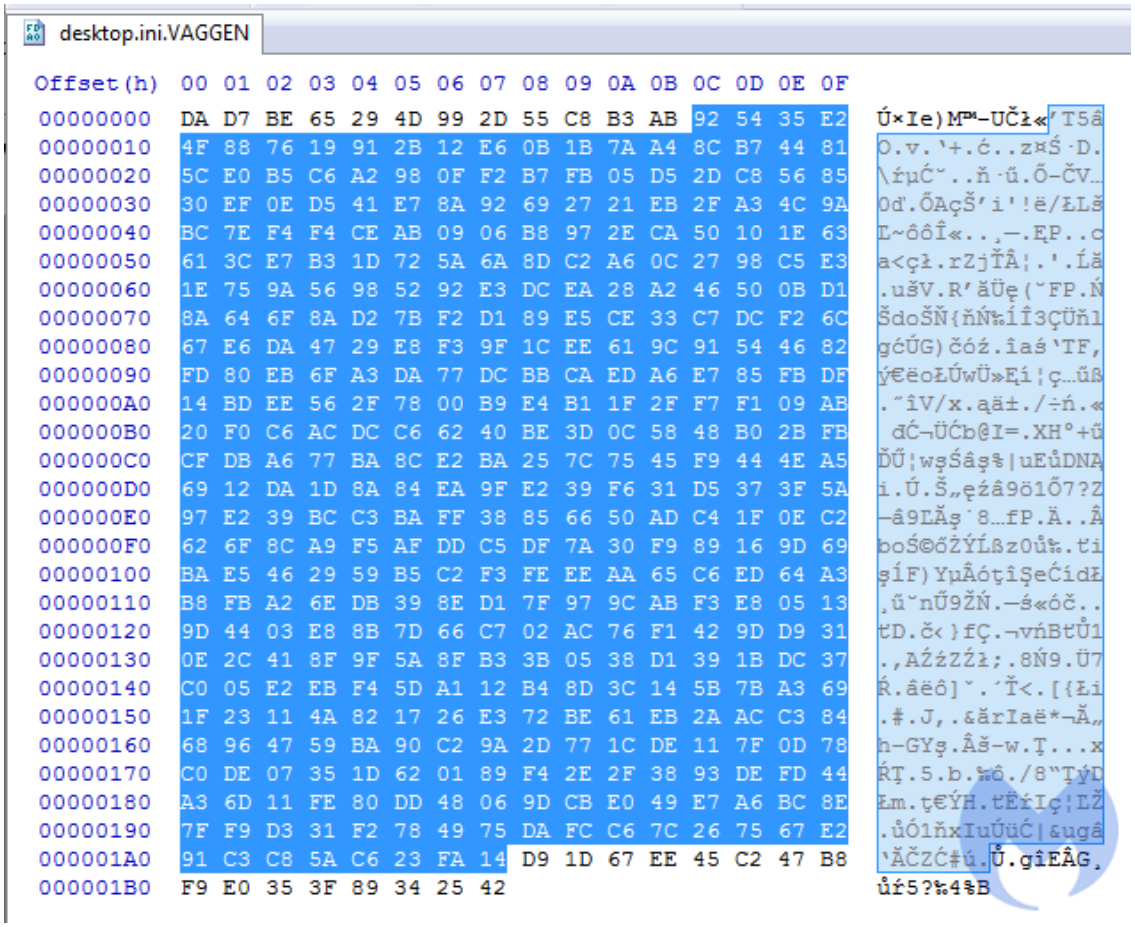
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

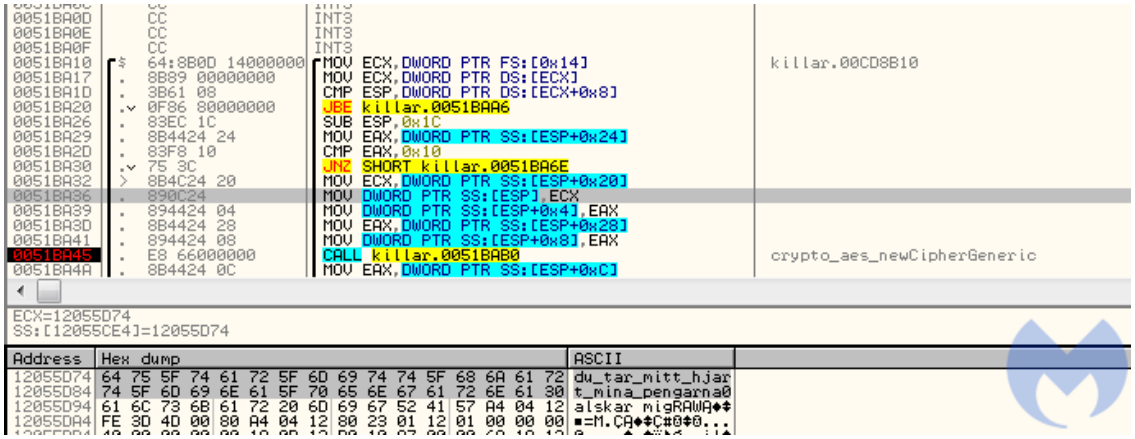


The content of the output file (with .VAGGEN extension) contains:

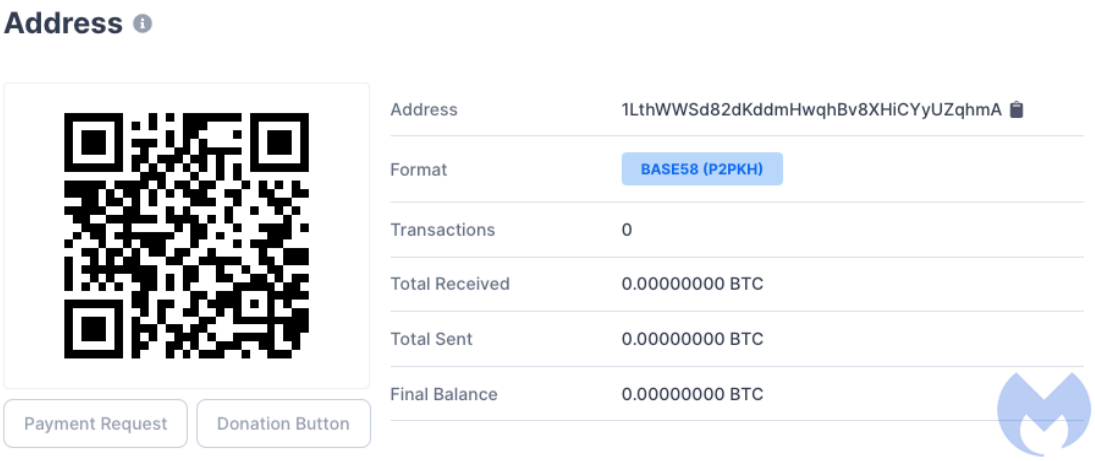
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

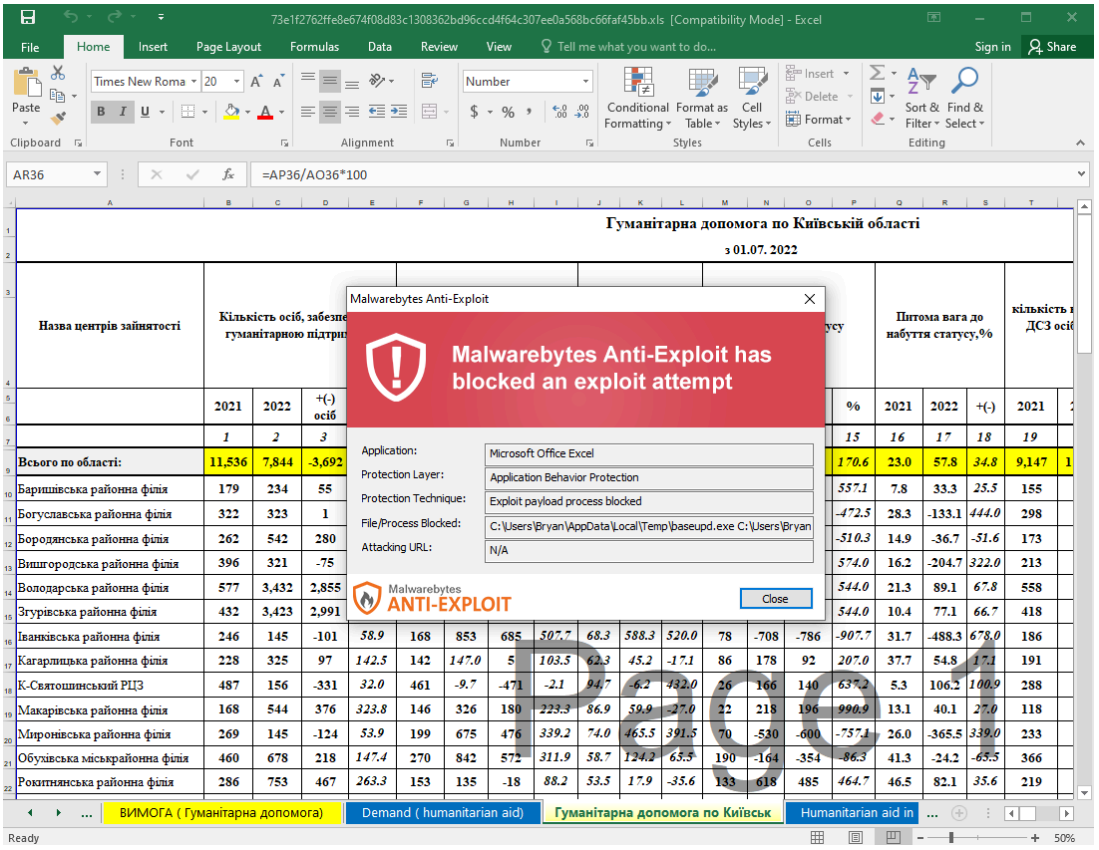
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

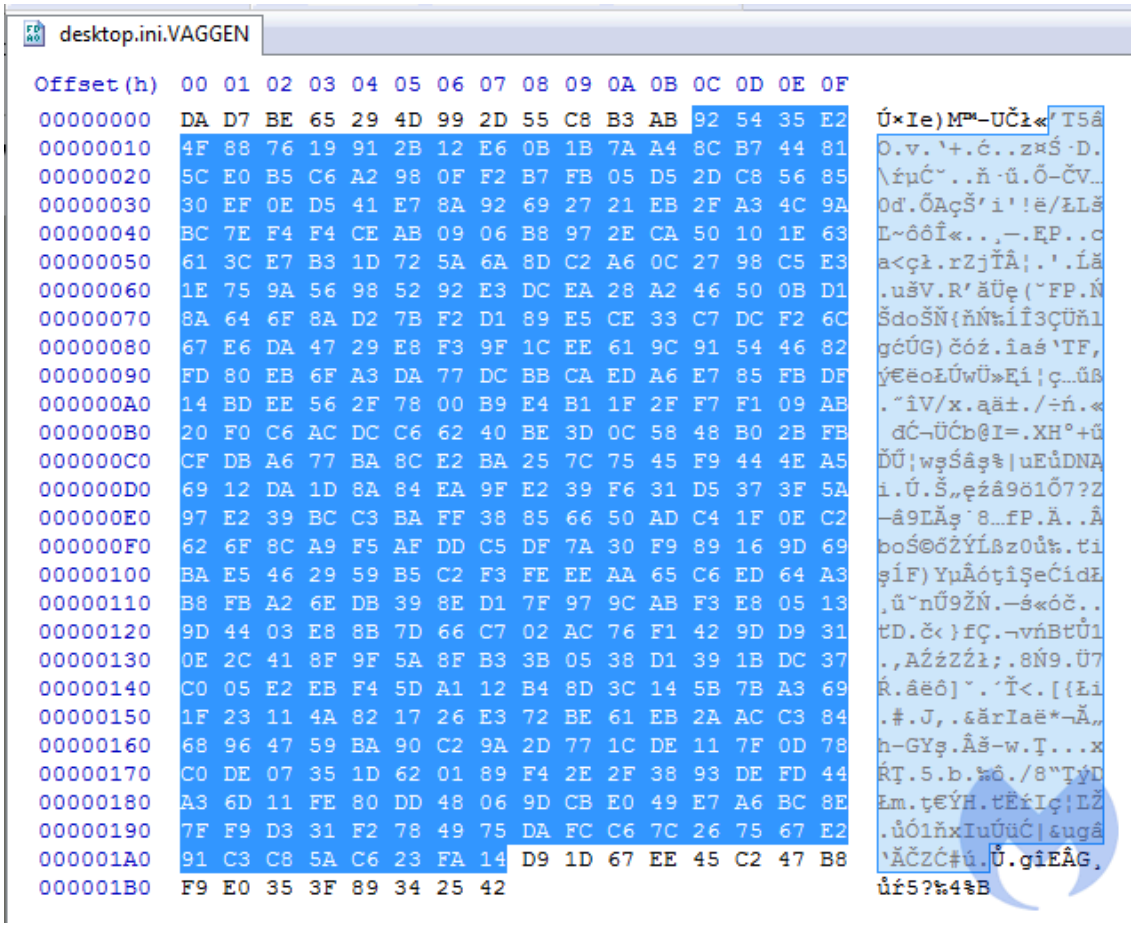
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



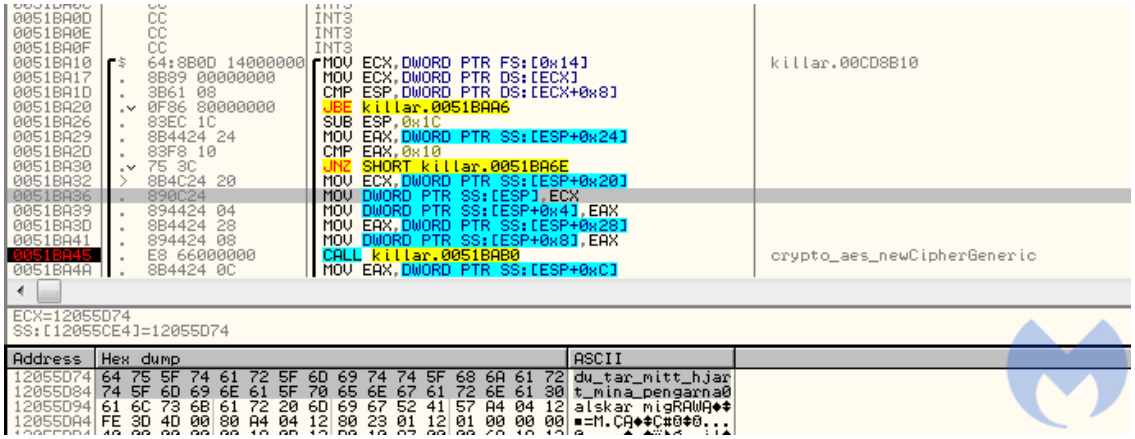
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

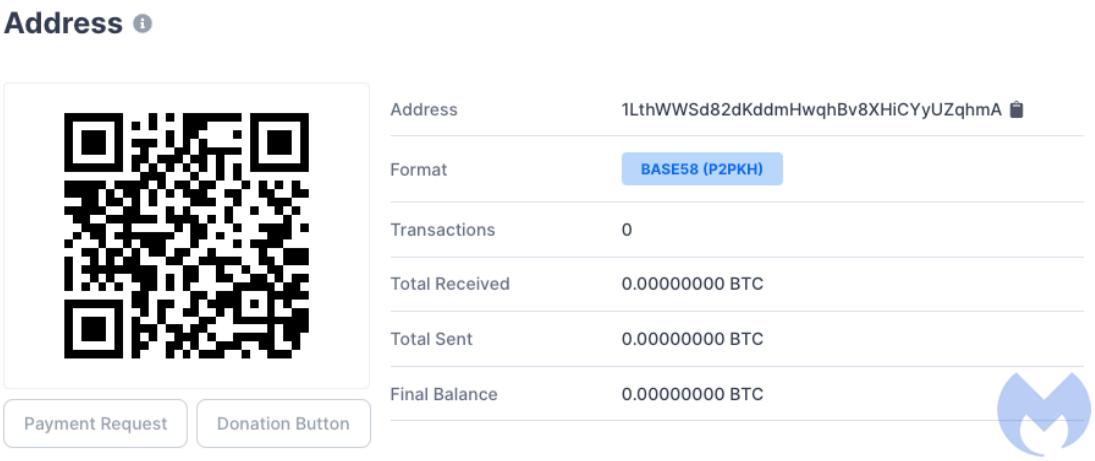
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

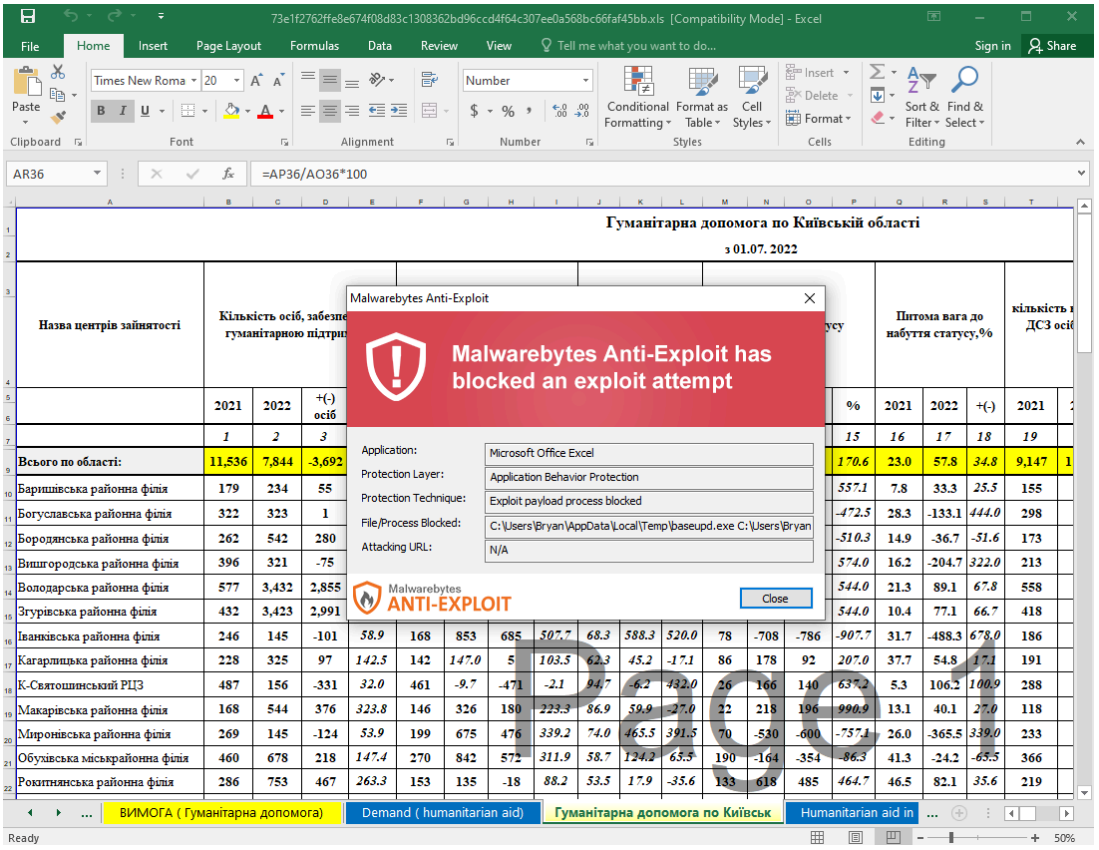
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

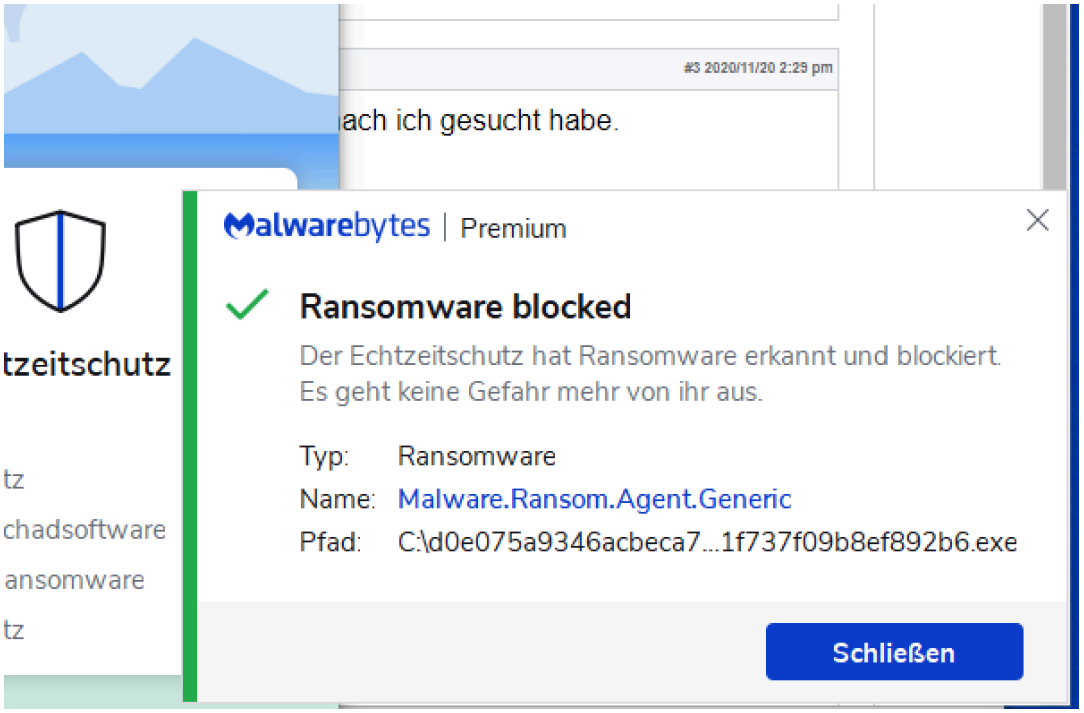
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

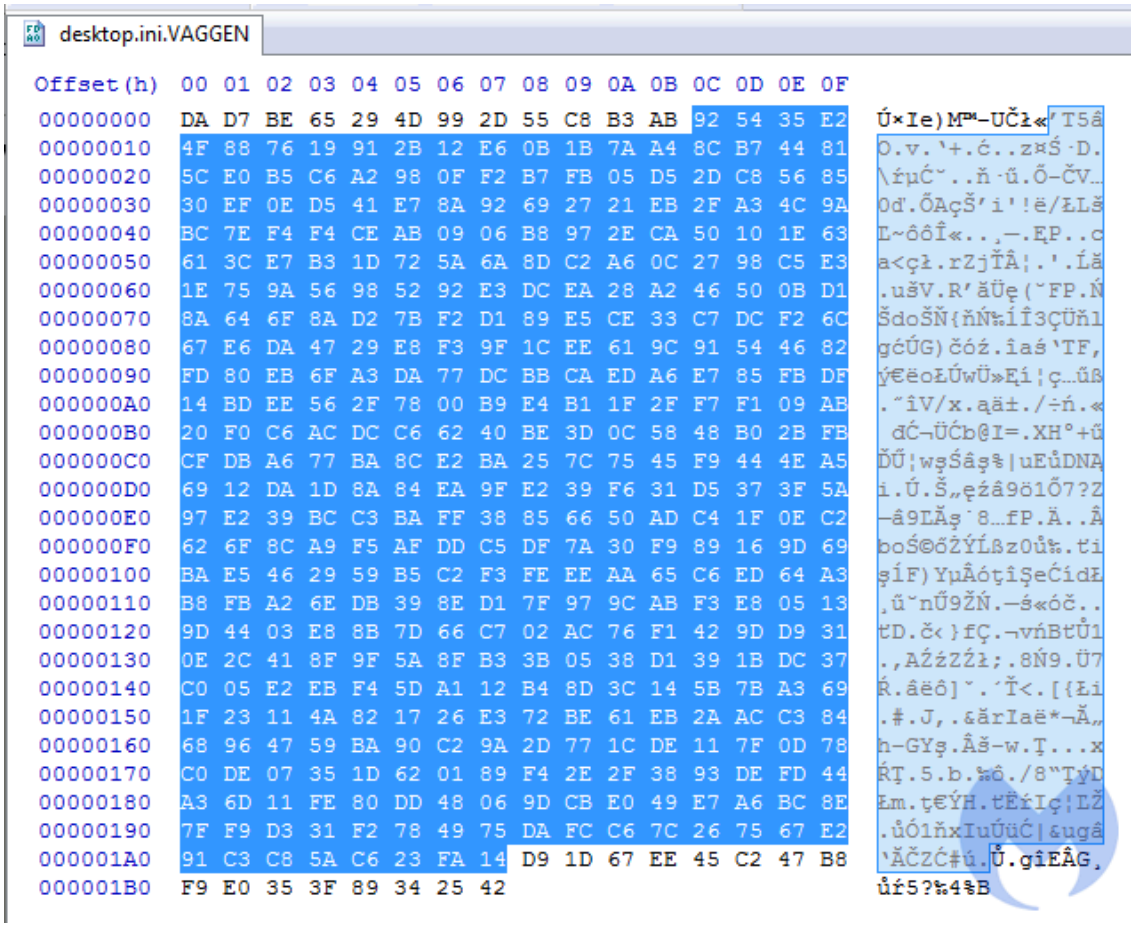
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

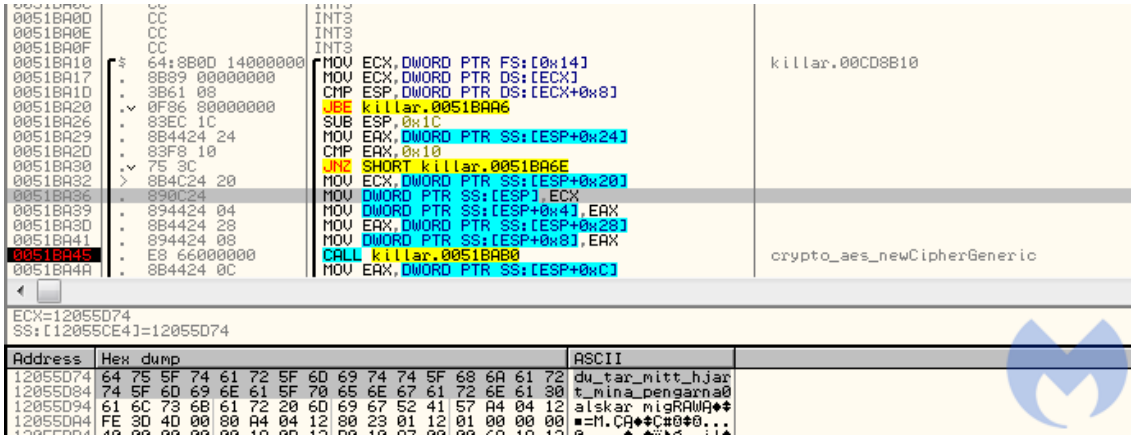


The content of the output file (with .VAGGEN extension) contains:

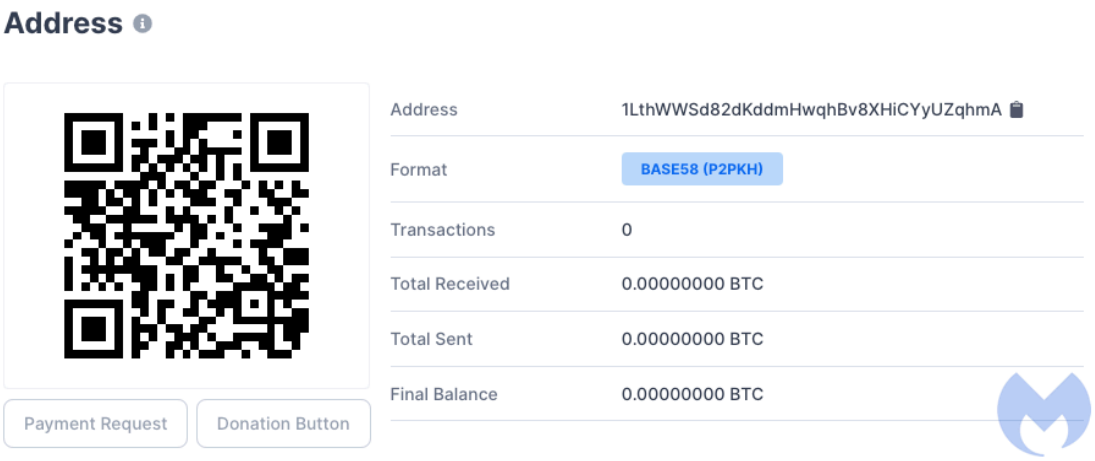
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

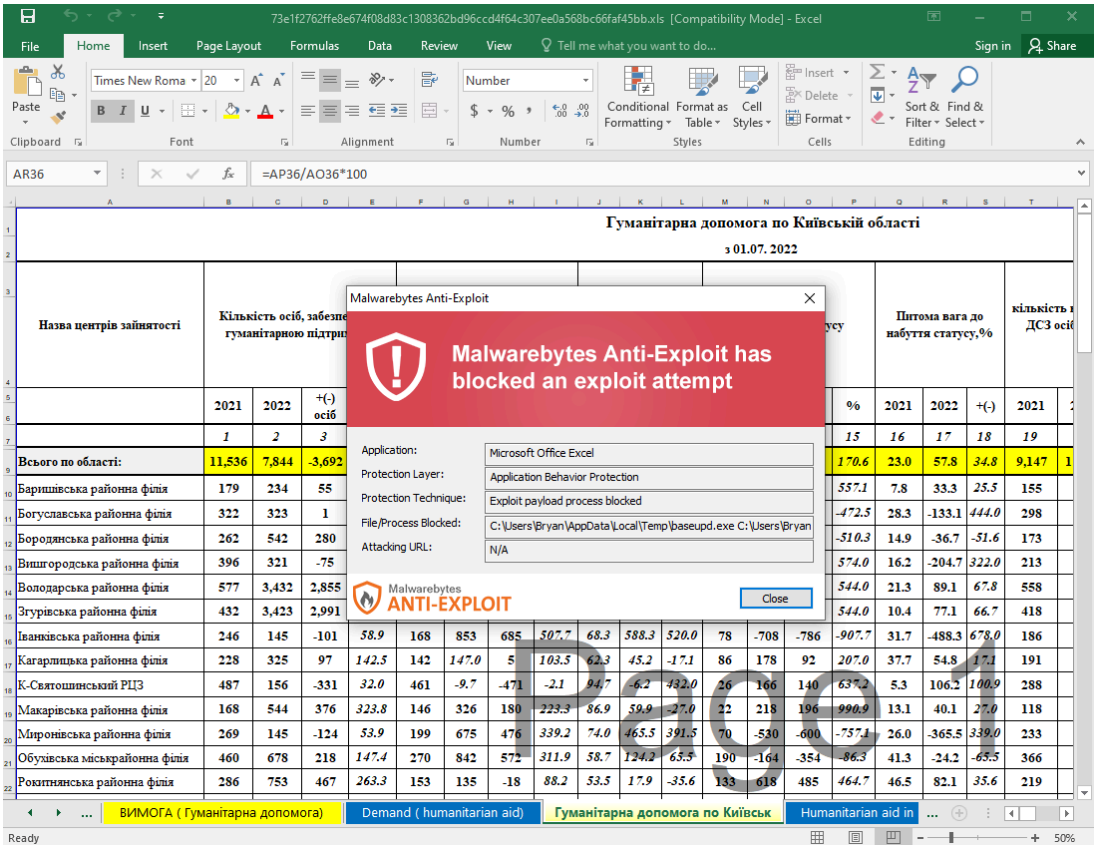
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

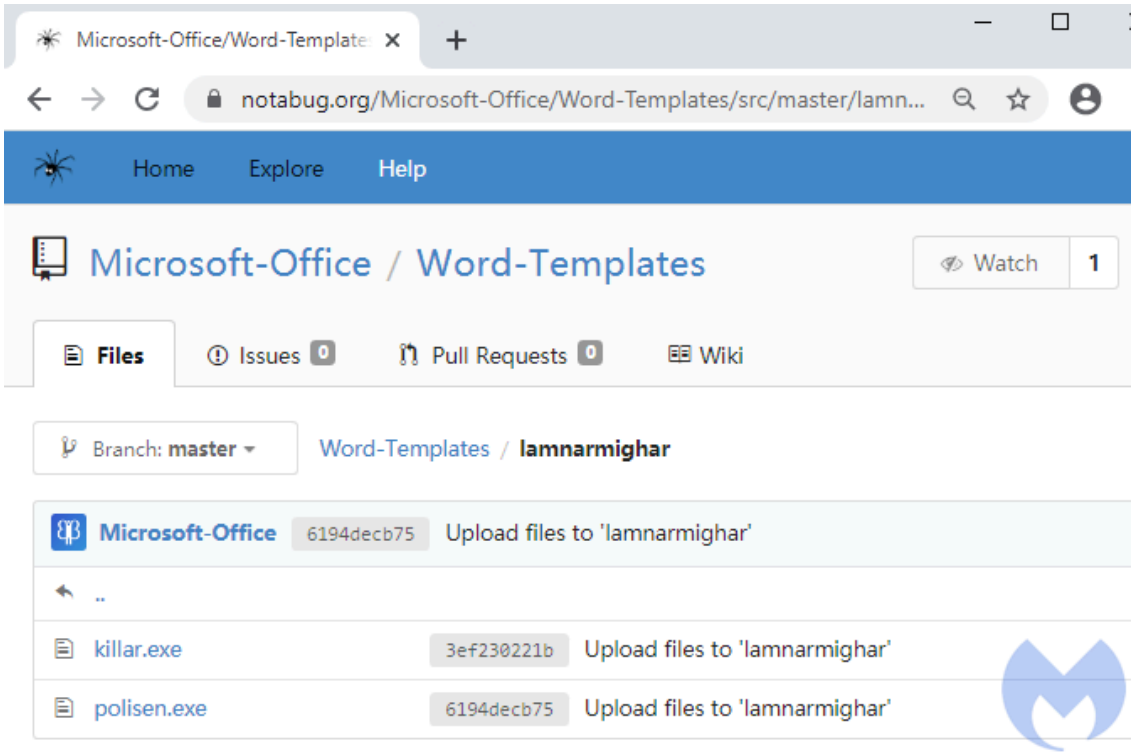
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



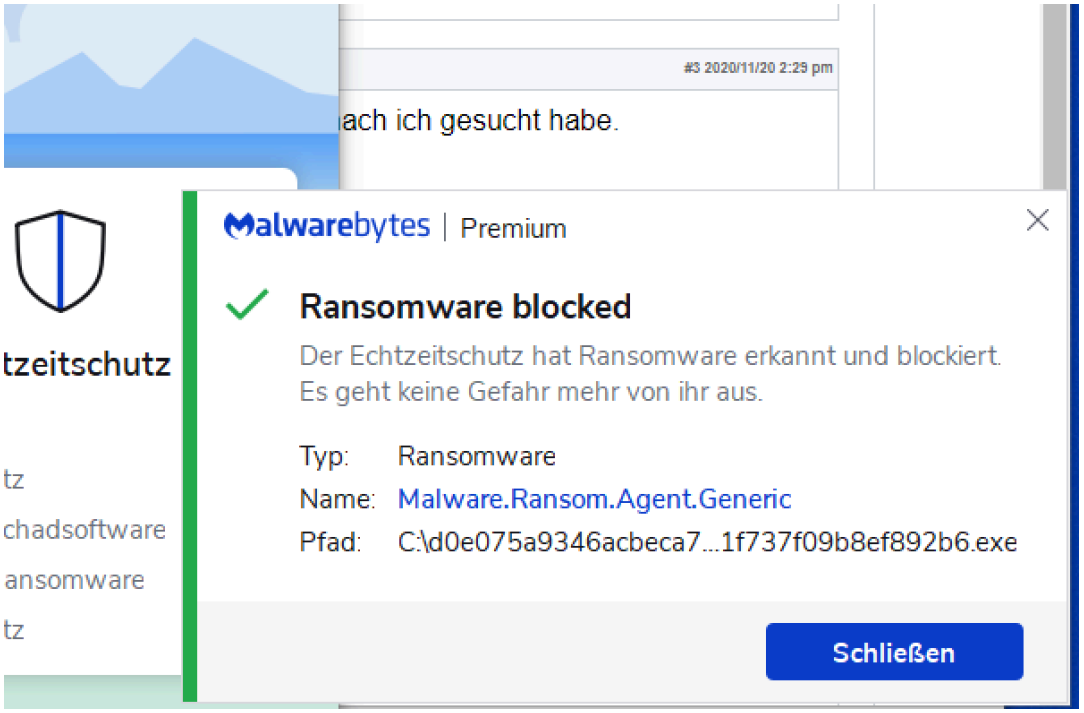
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

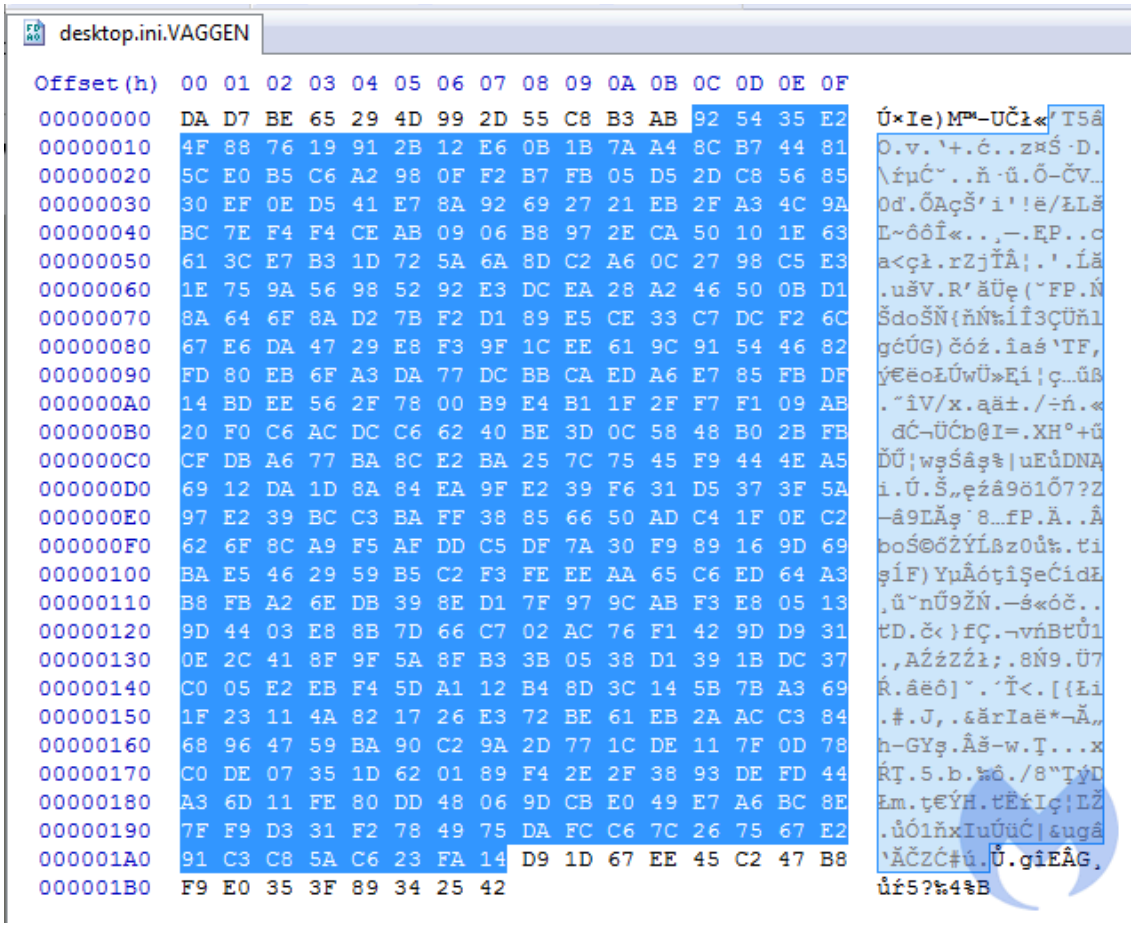
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

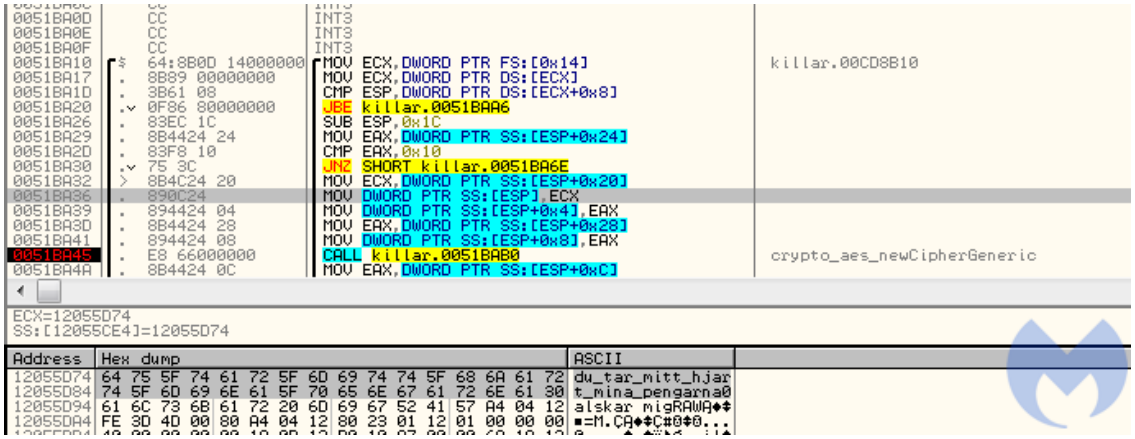


The content of the output file (with .VAGGEN extension) contains:

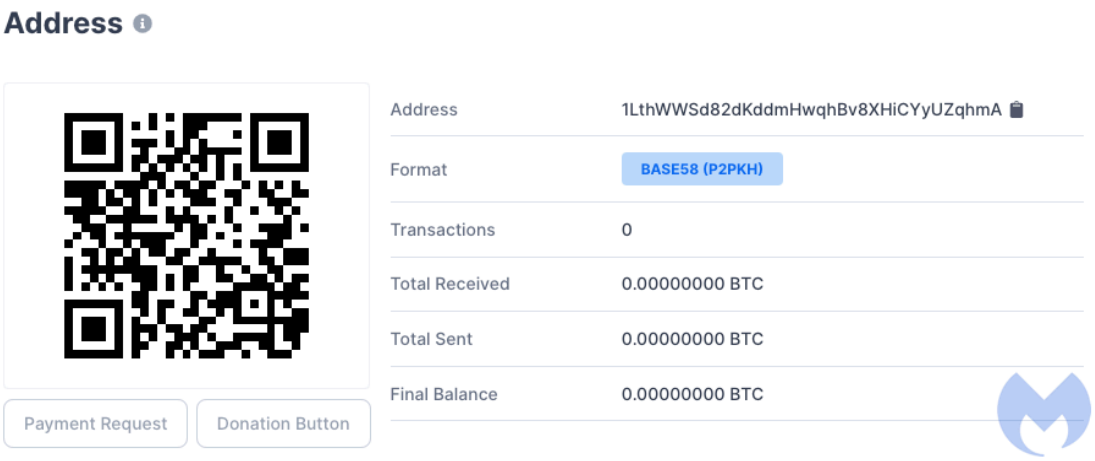
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

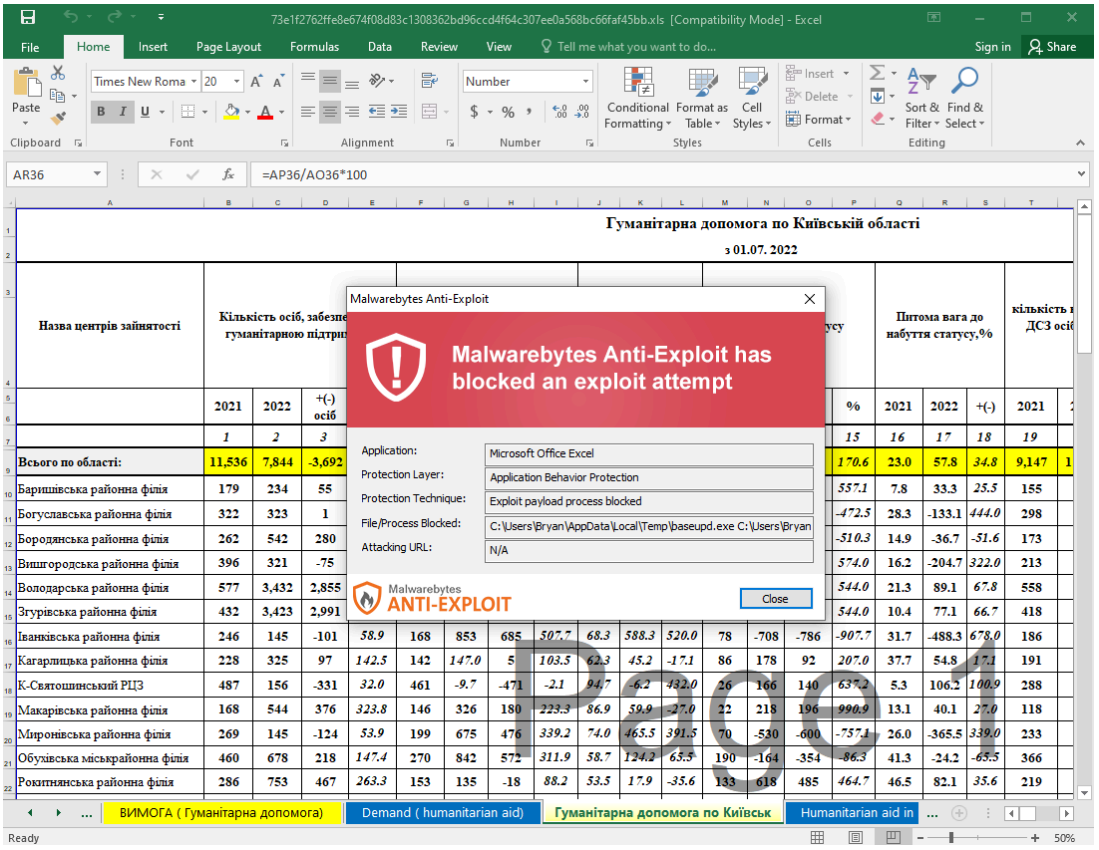
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

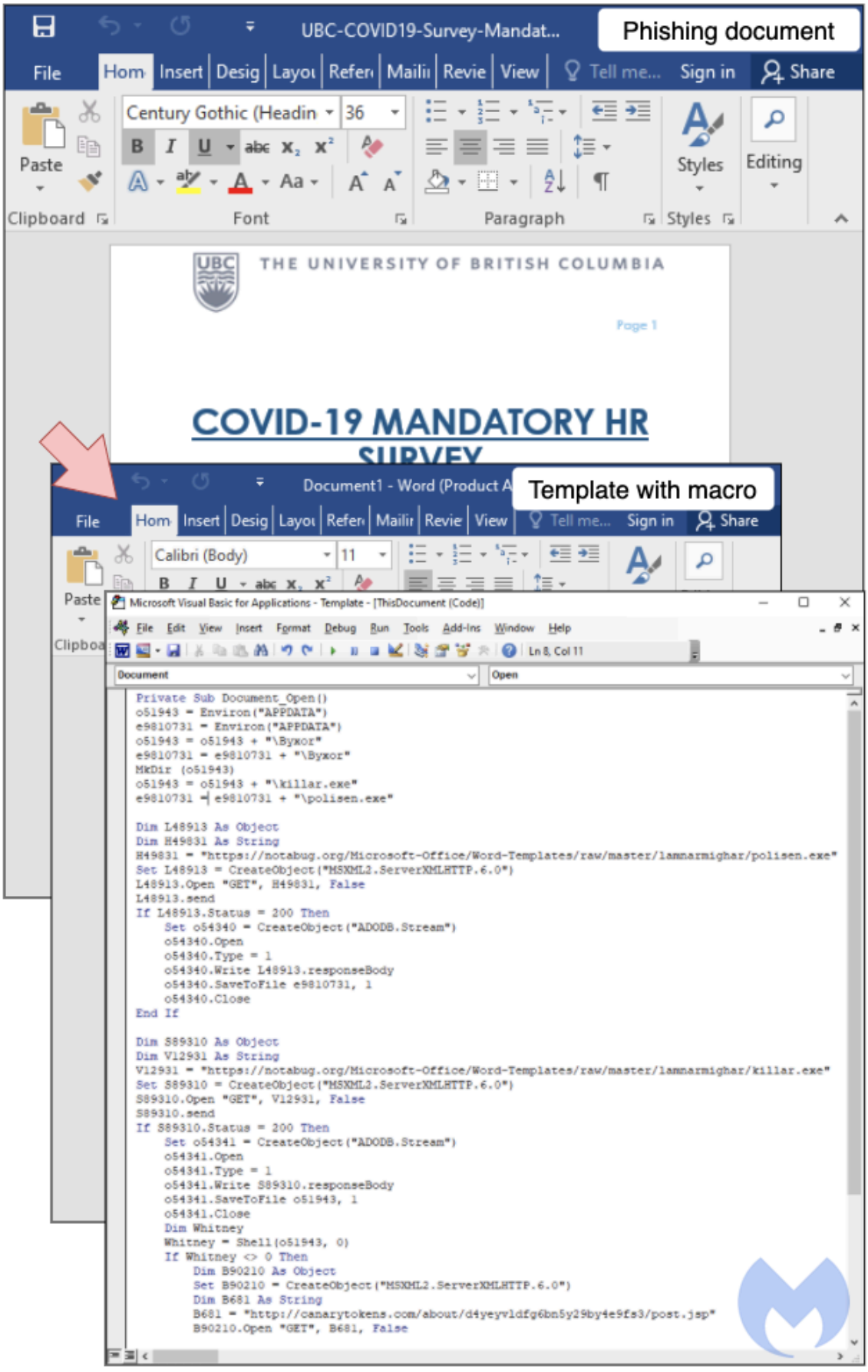
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

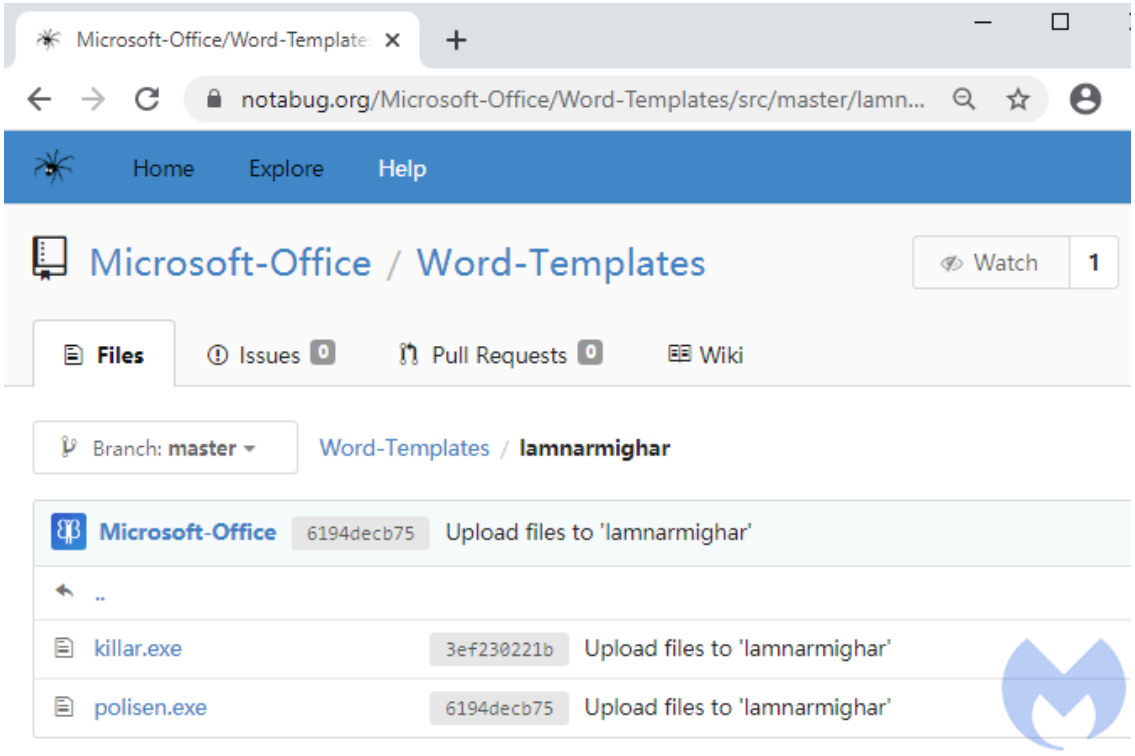
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



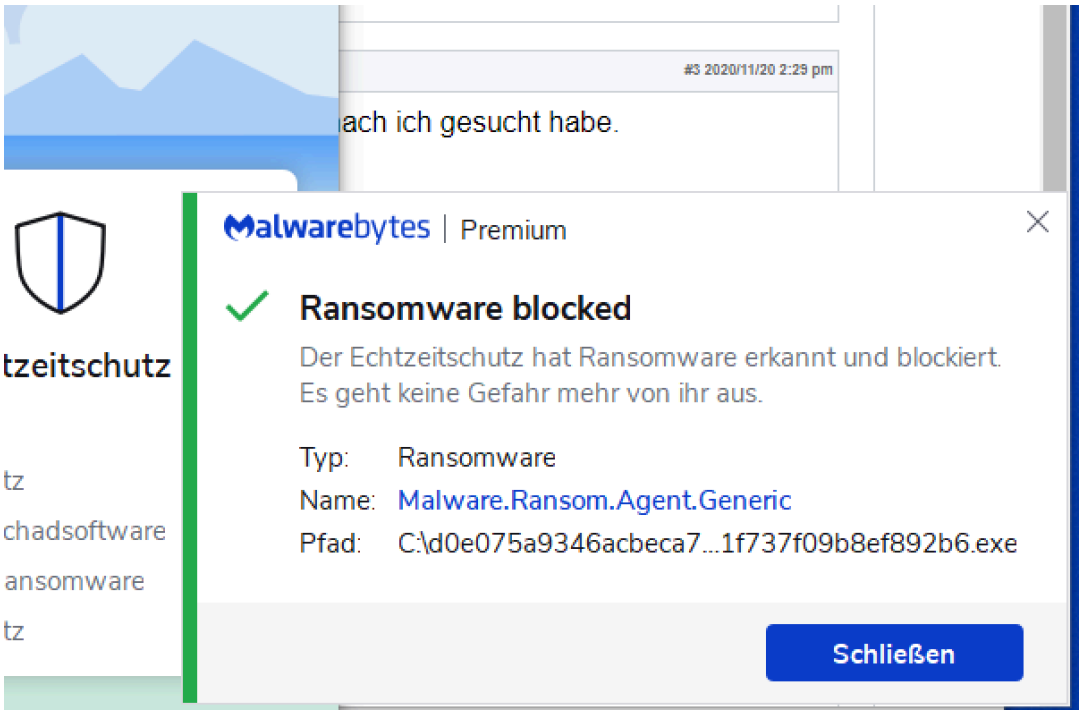
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

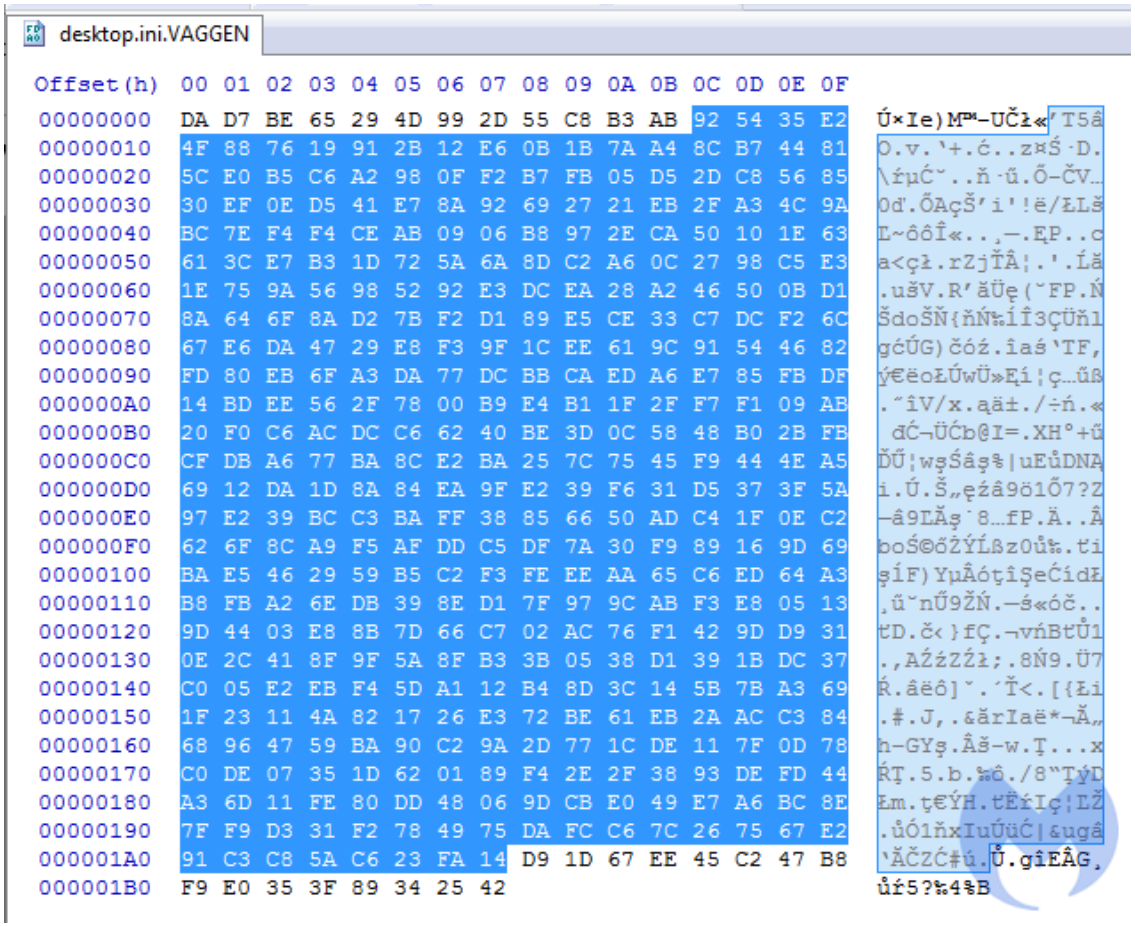
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

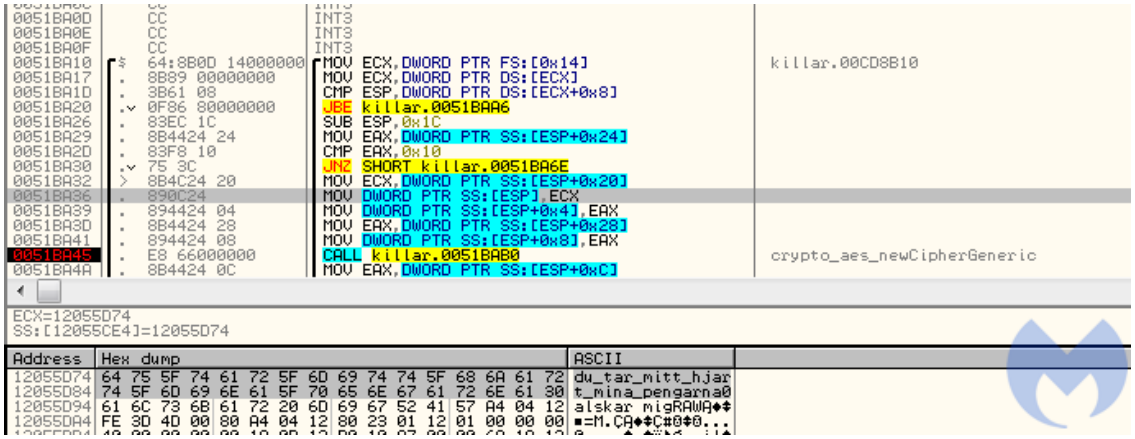


The content of the output file (with .VAGGEN extension) contains:

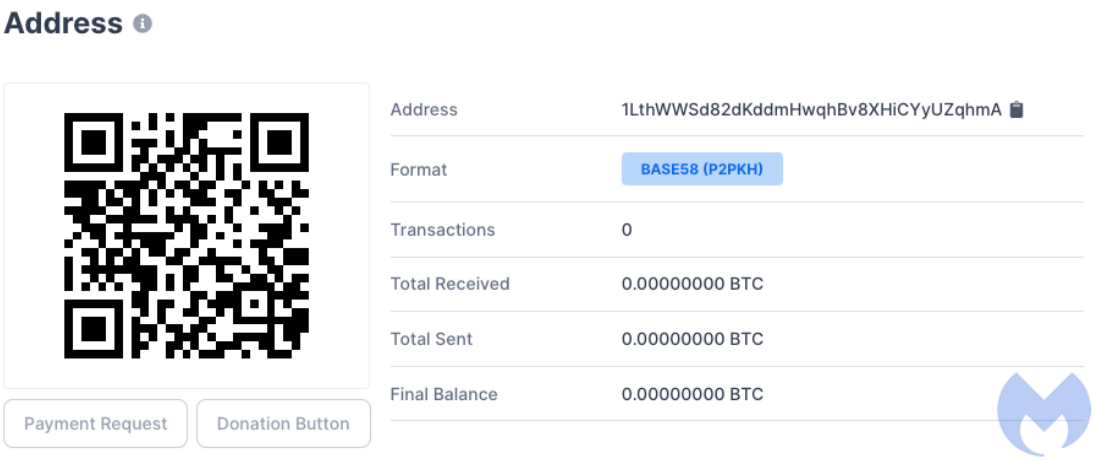
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

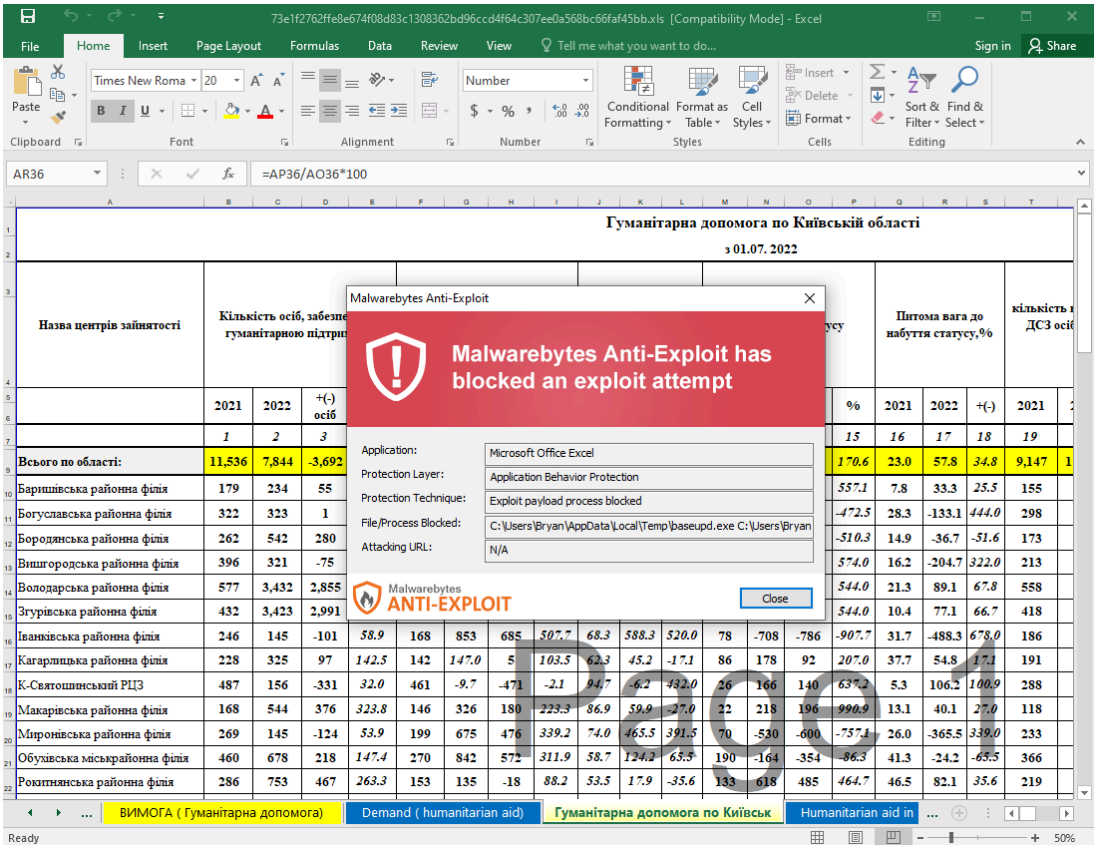
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

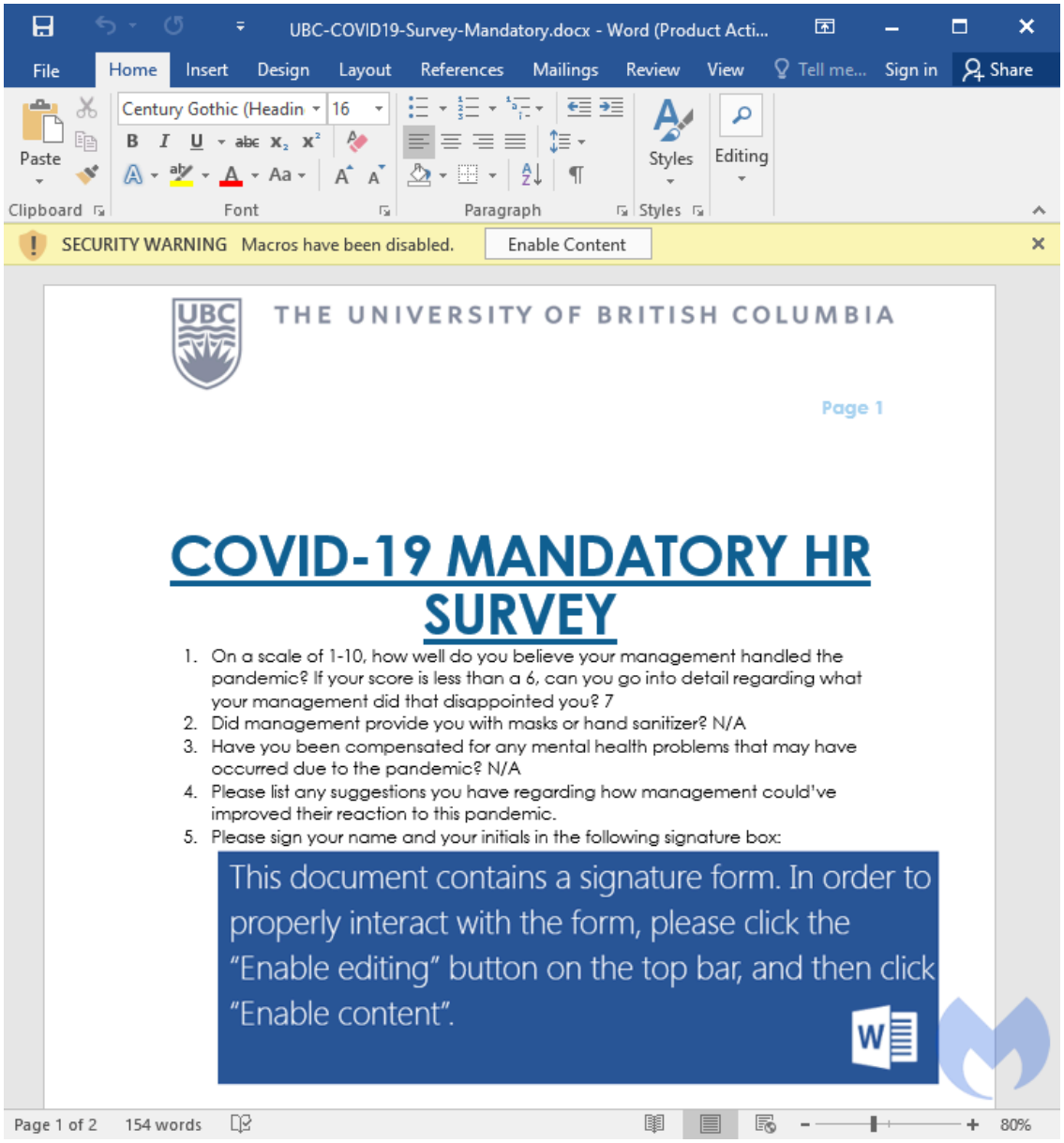
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

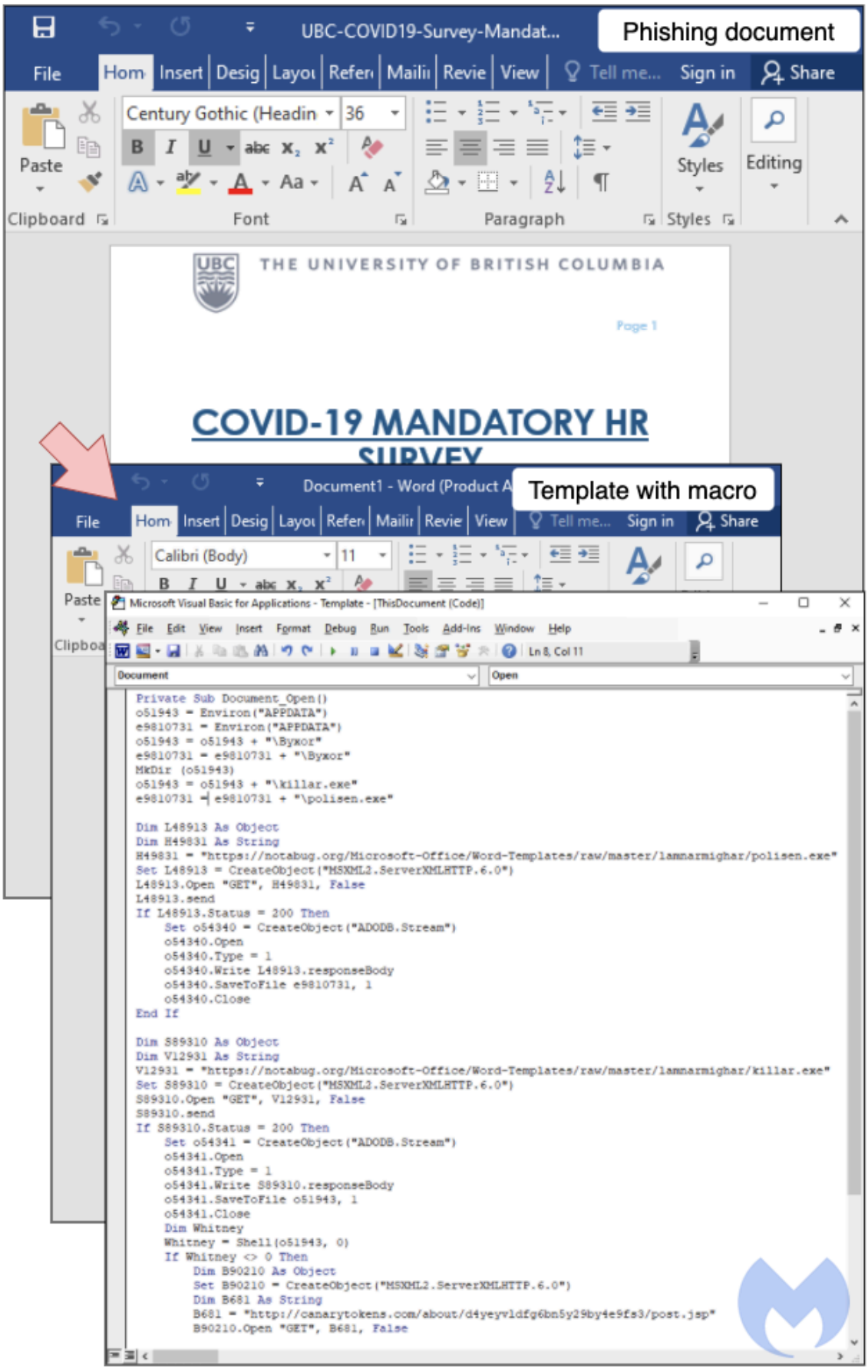
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

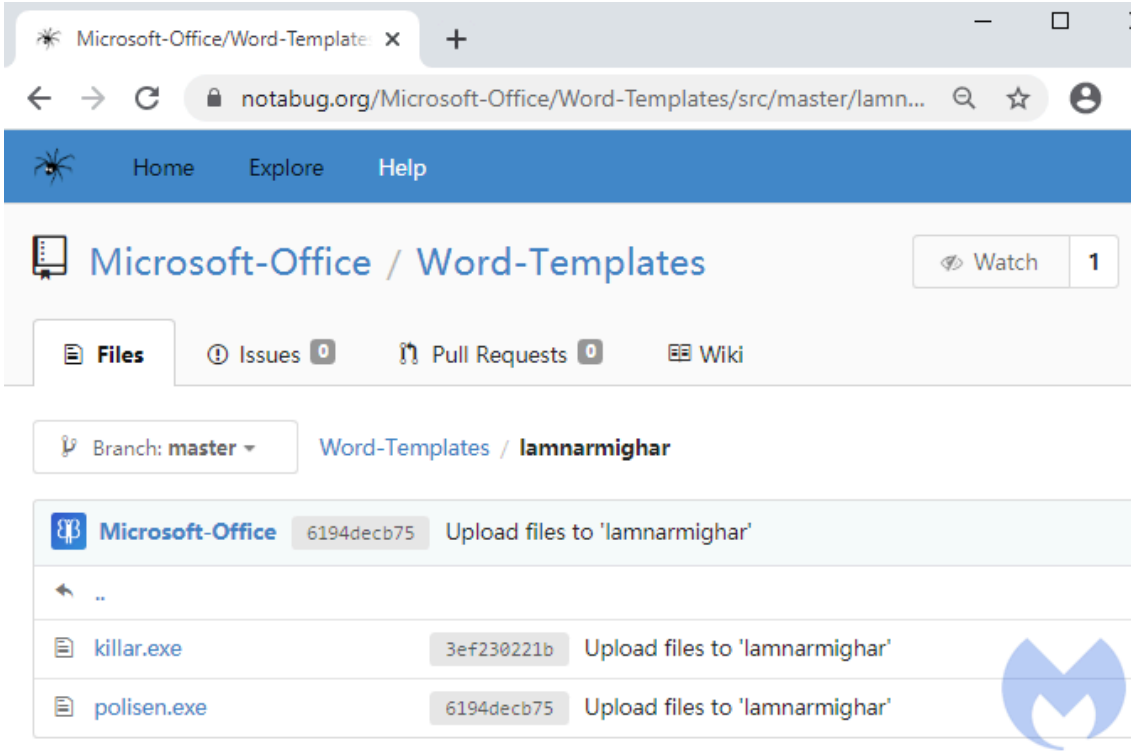
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



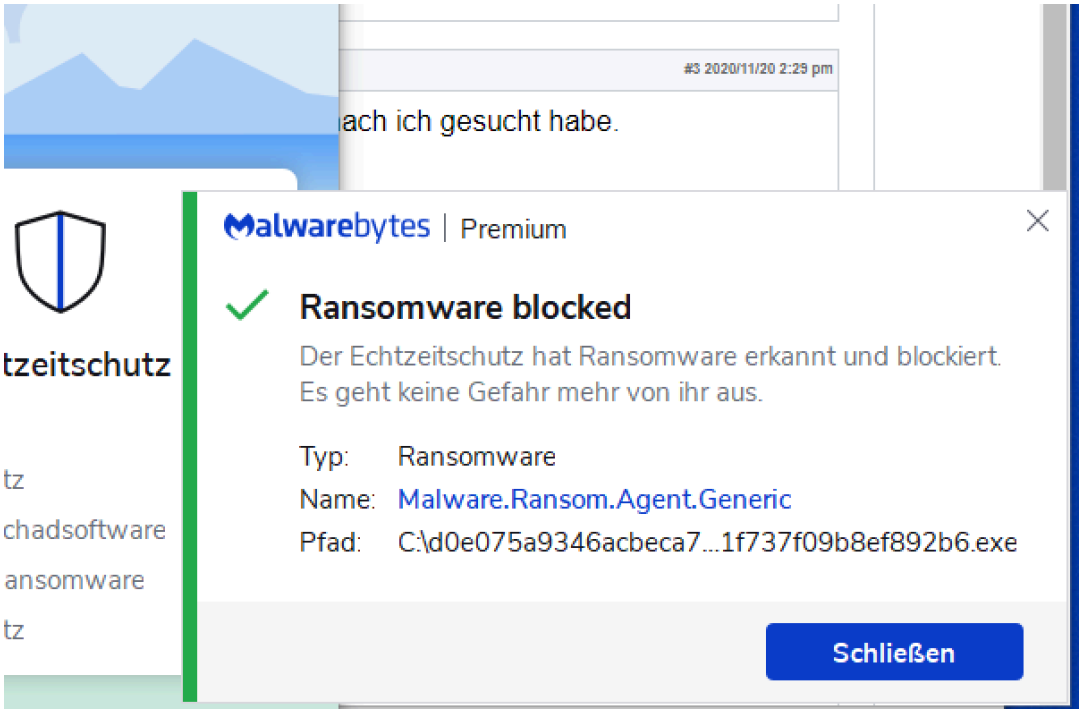
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

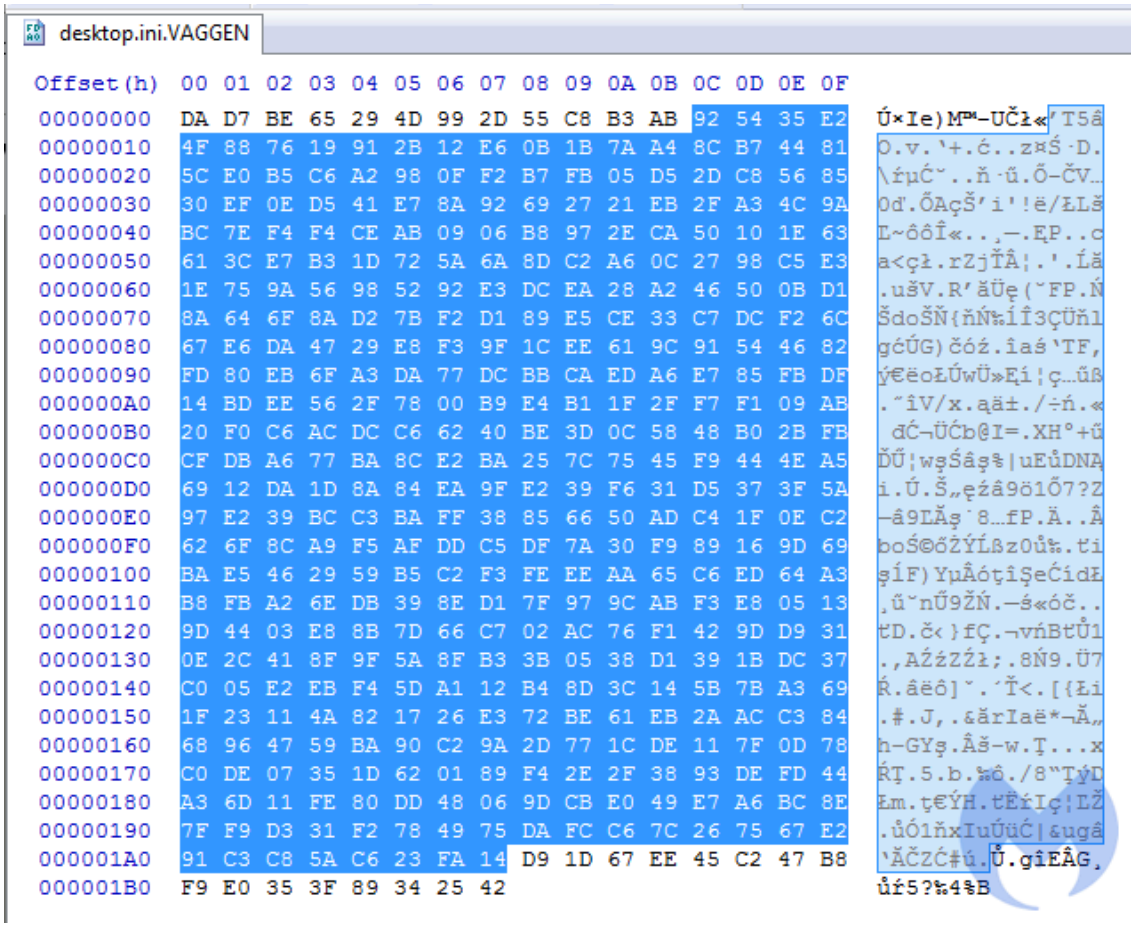
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

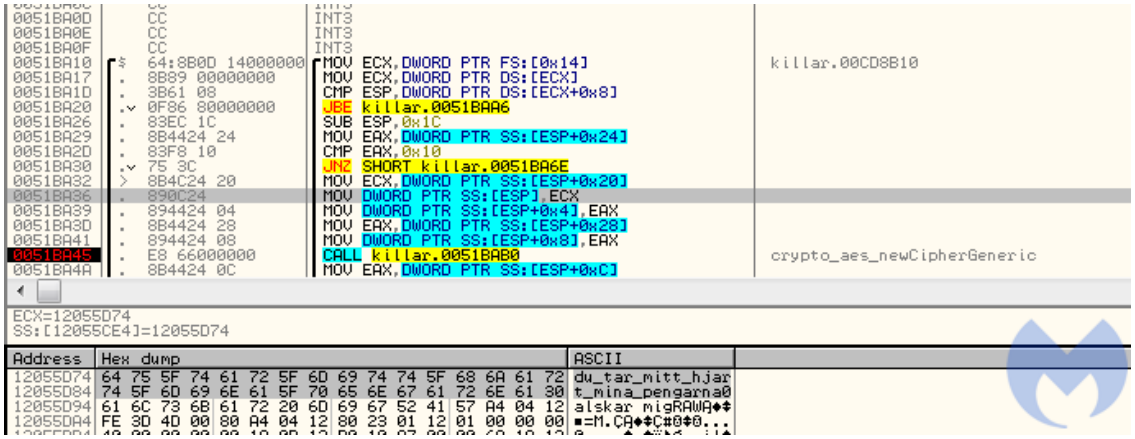


The content of the output file (with .VAGGEN extension) contains:

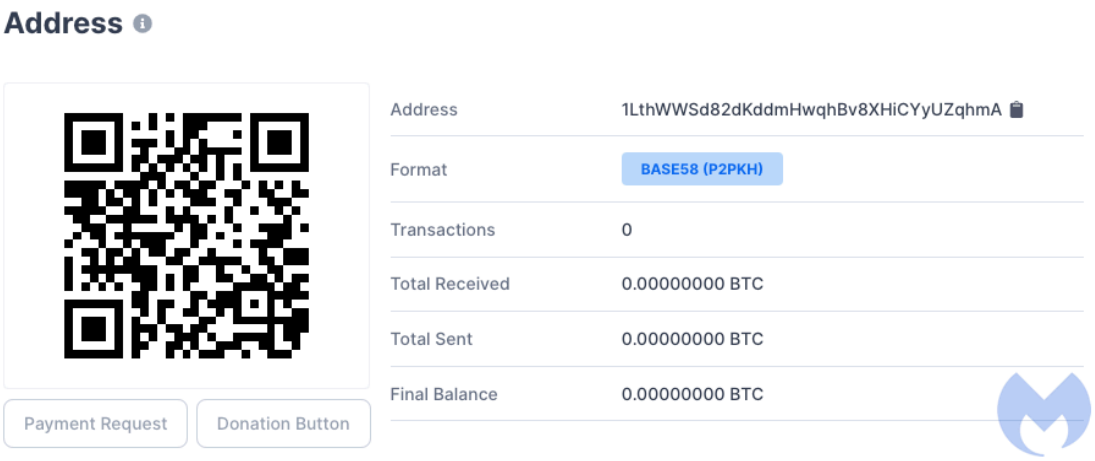
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

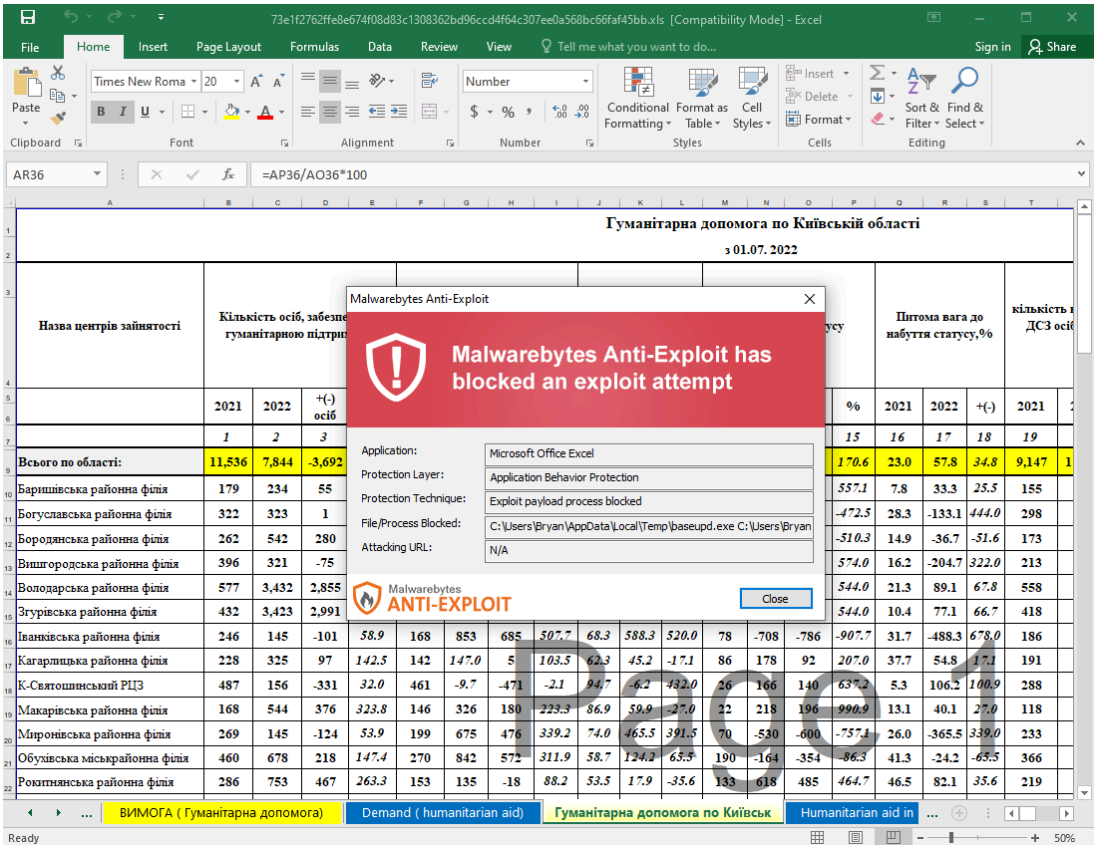
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

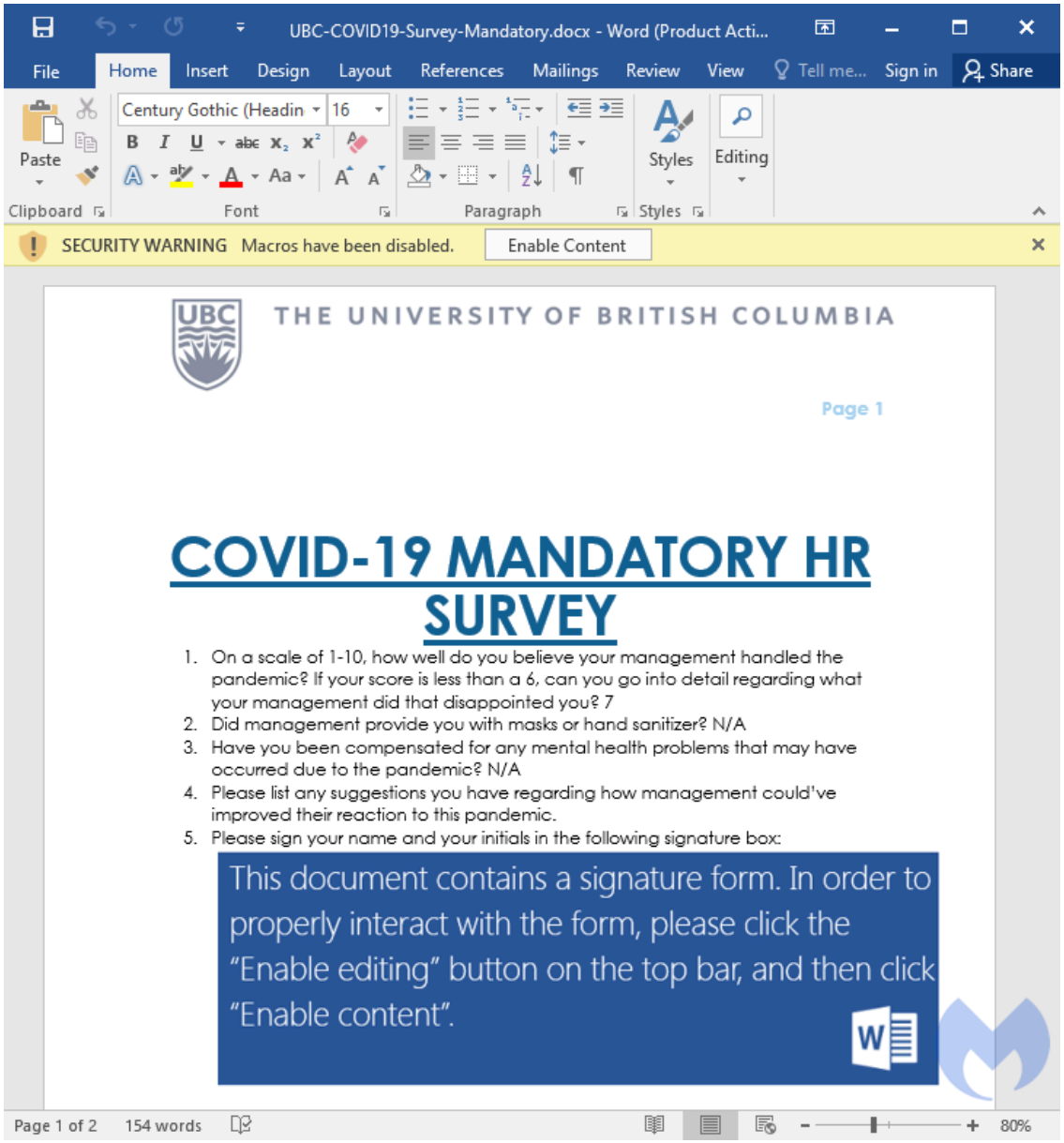
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

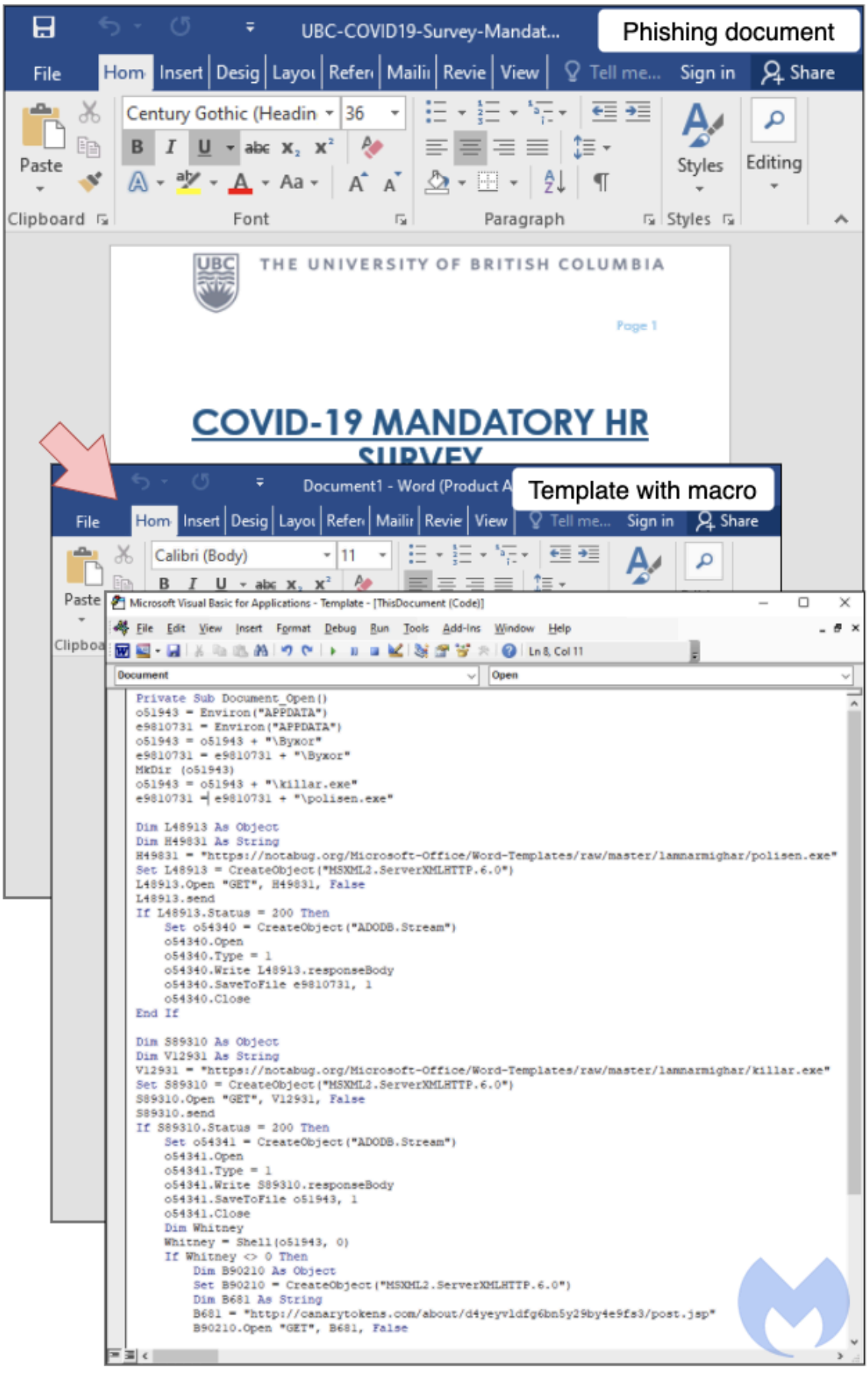
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

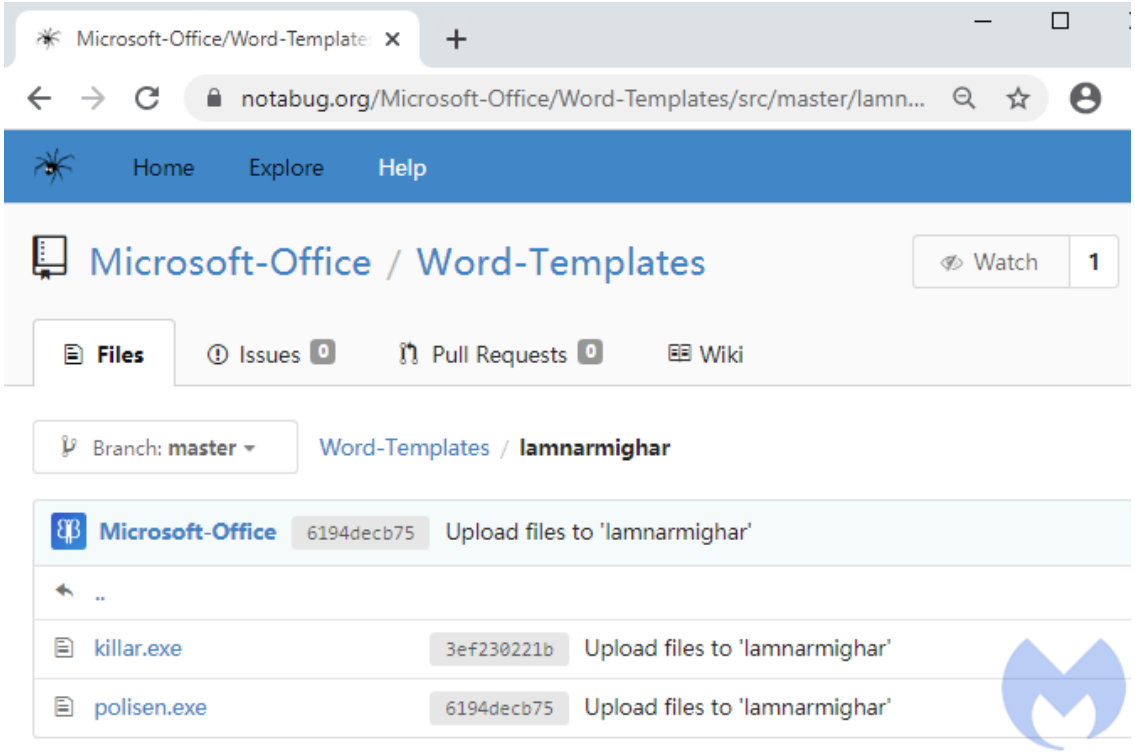
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



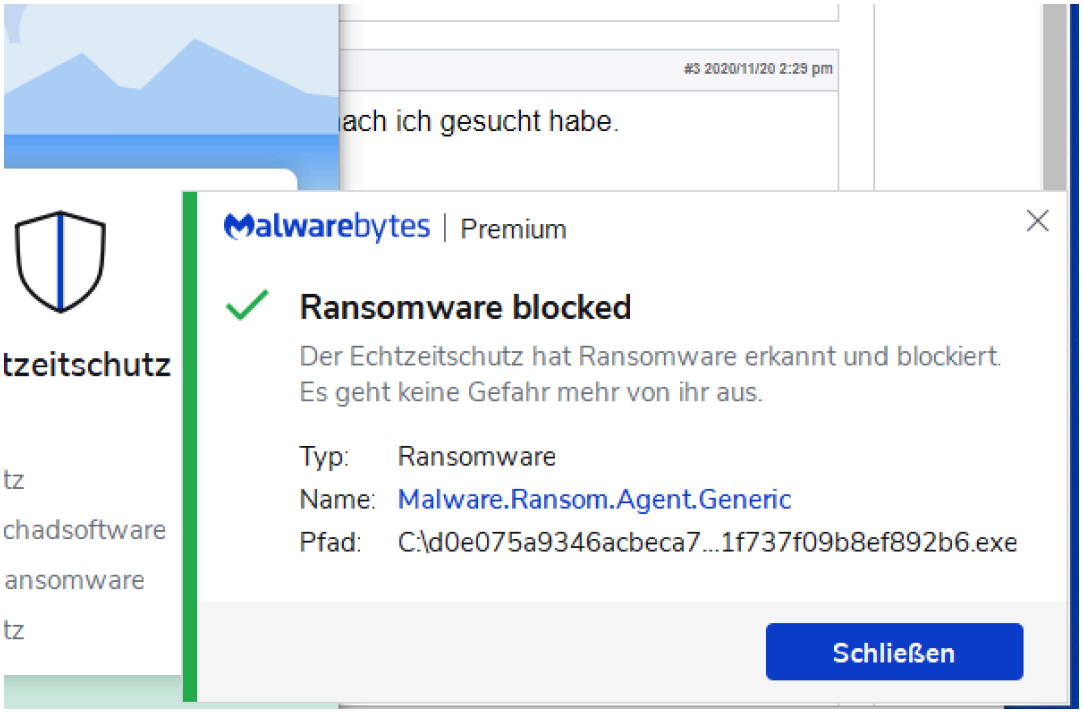
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

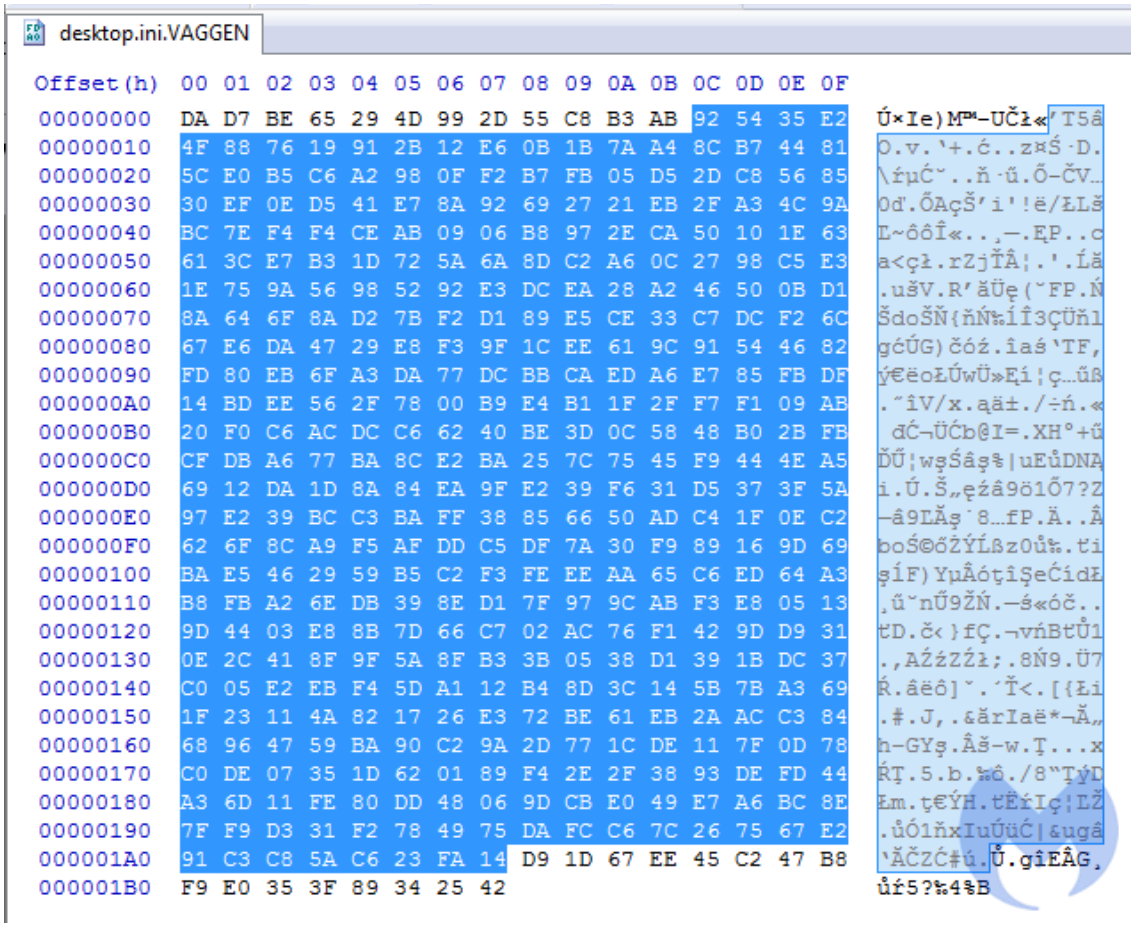
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

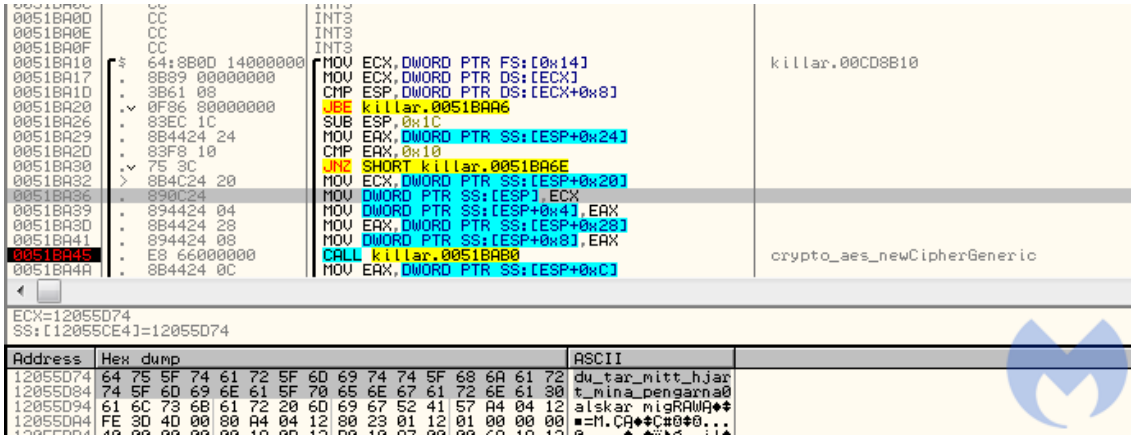


The content of the output file (with .VAGGEN extension) contains:

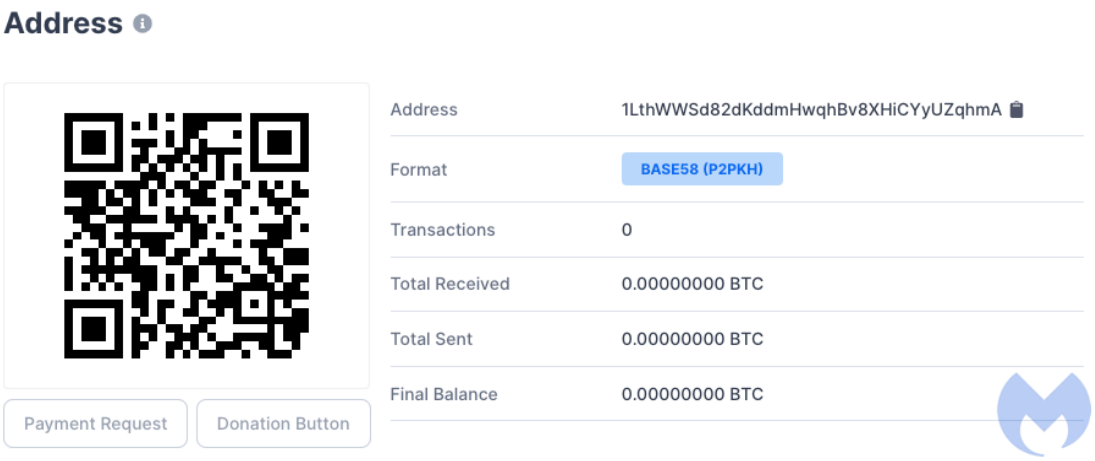
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

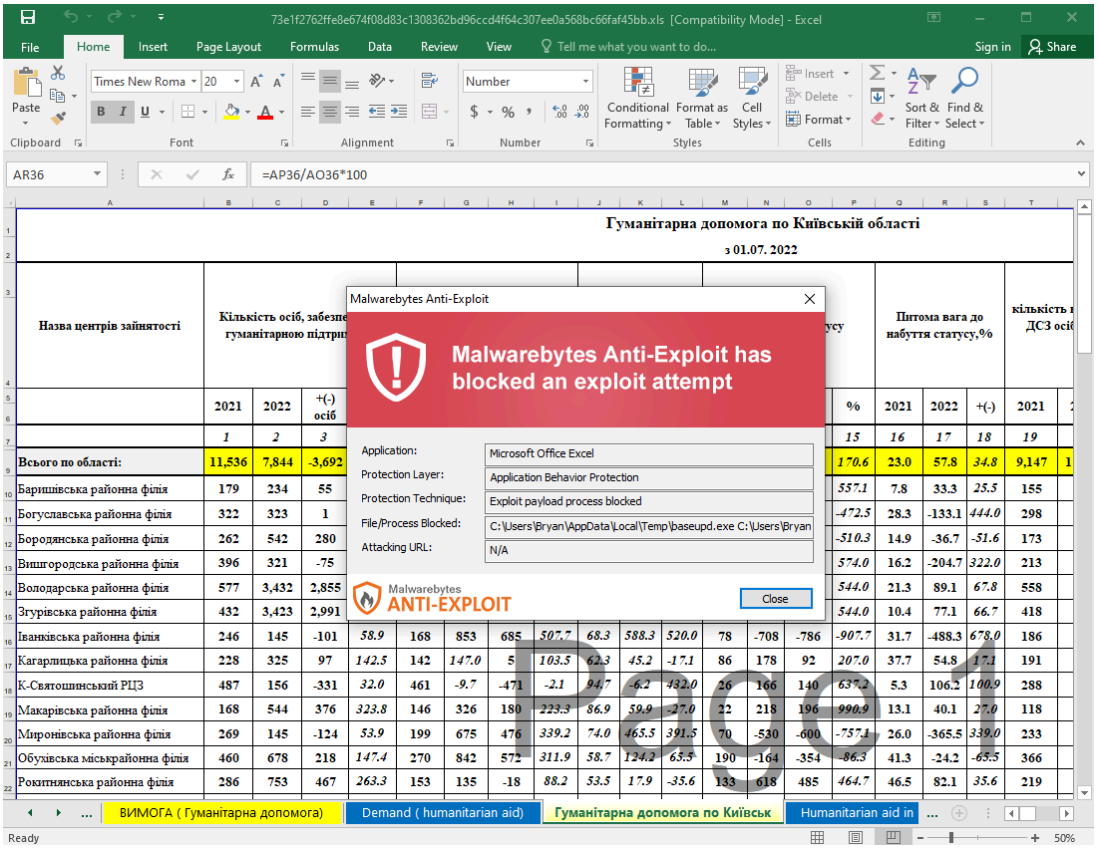
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

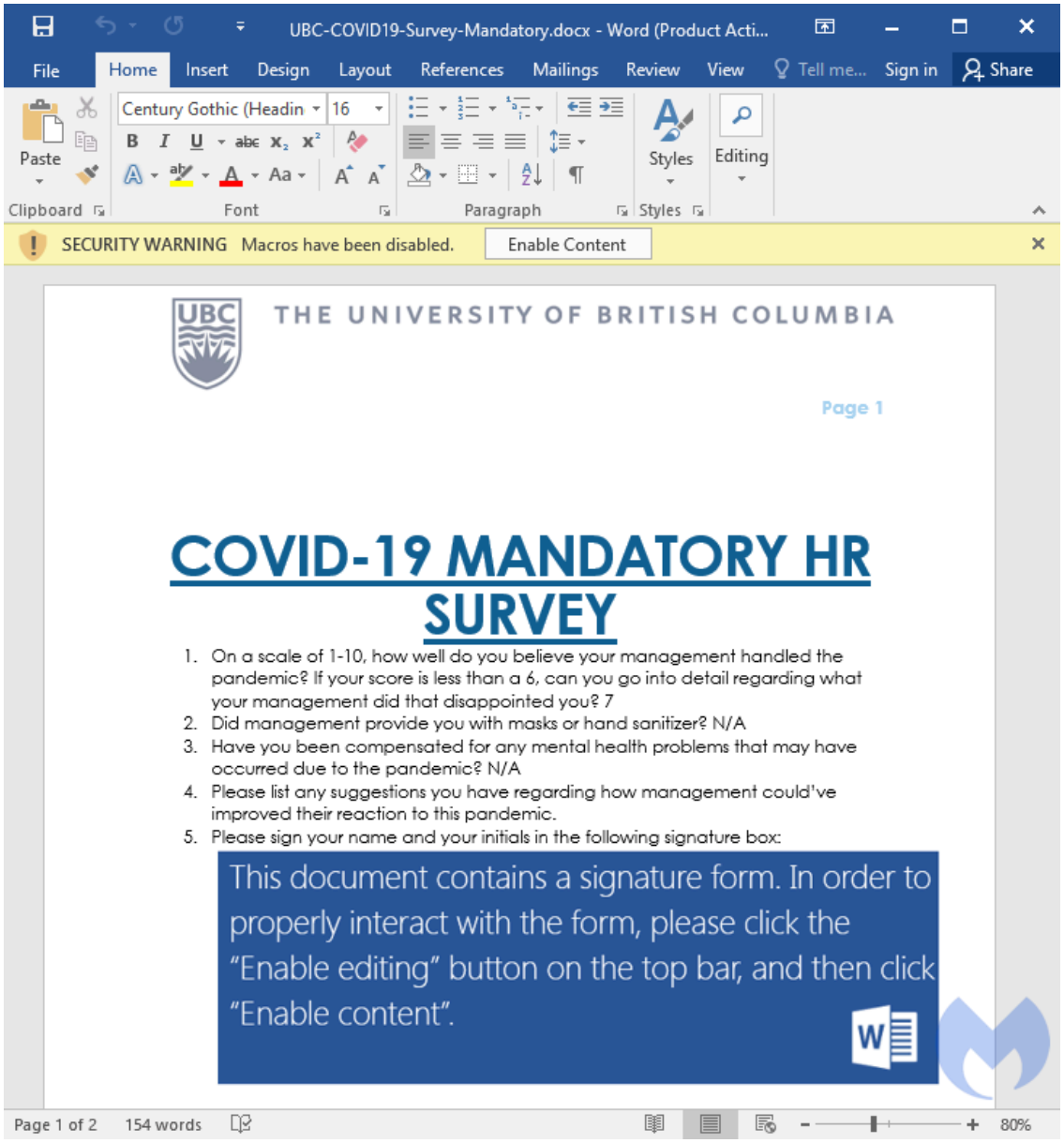
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

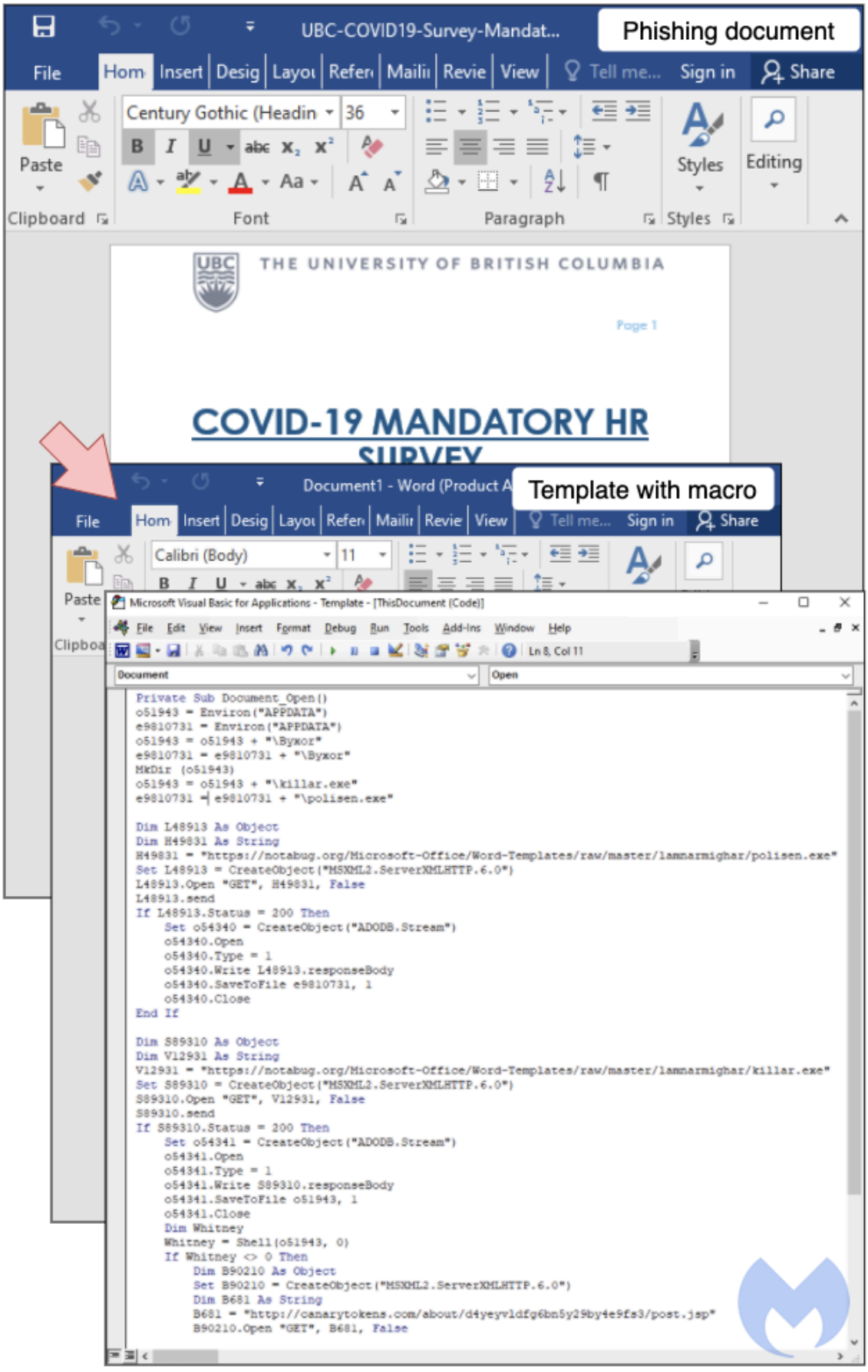
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

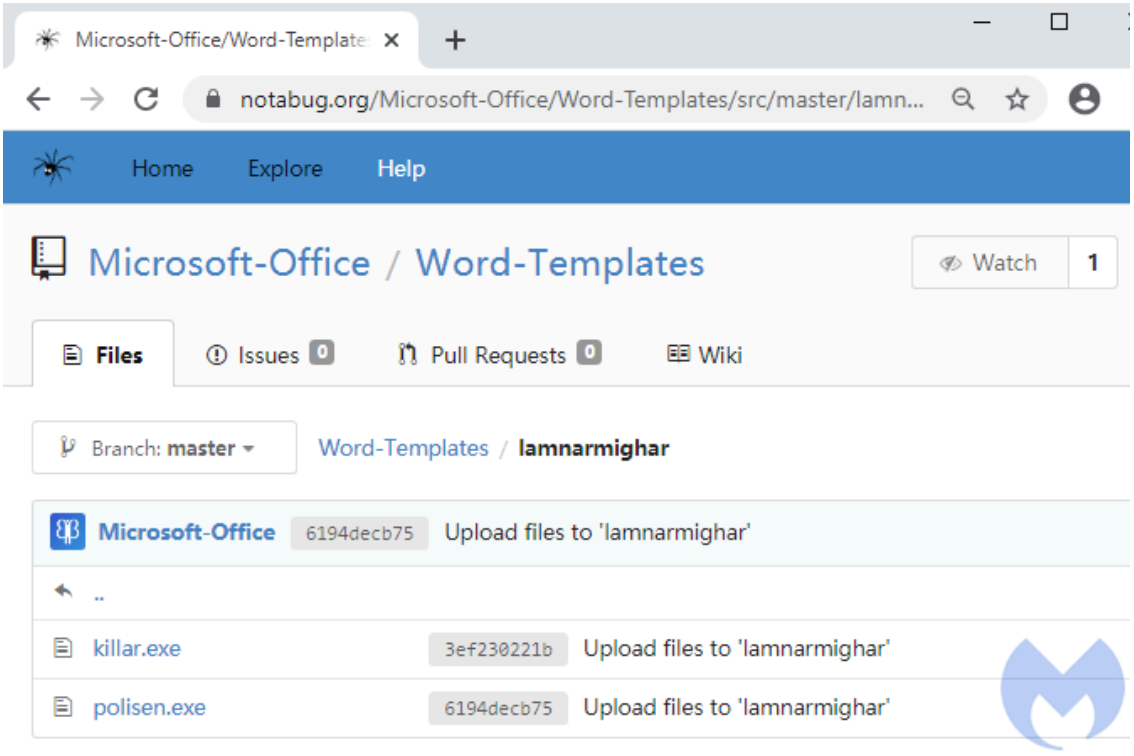
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



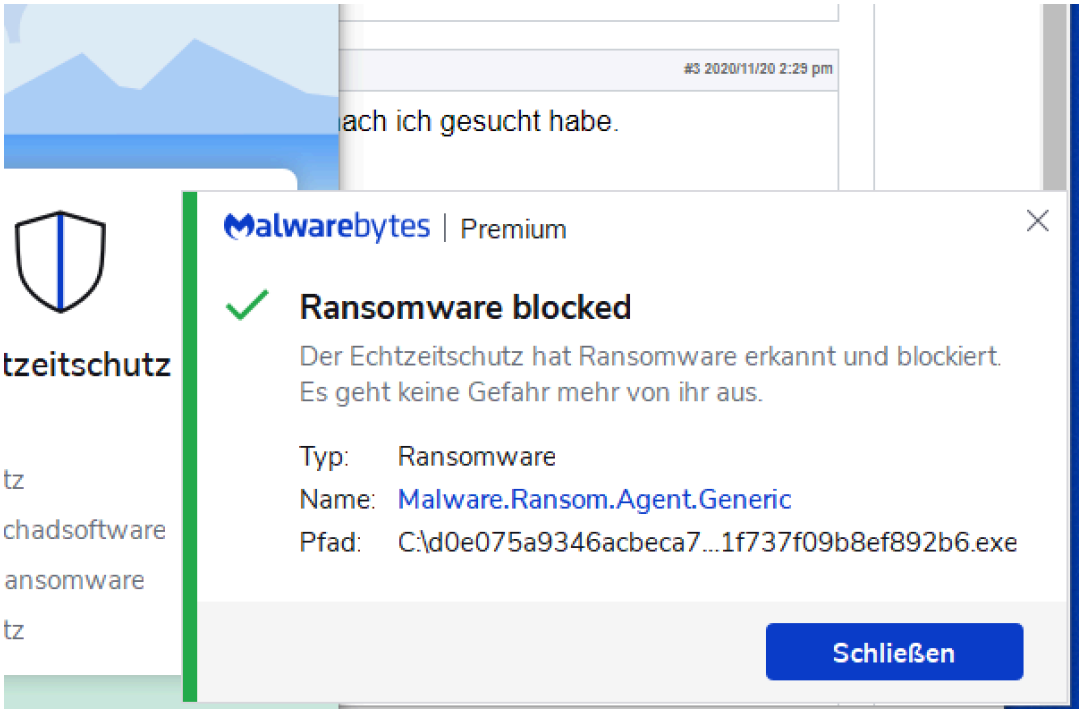
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```



Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

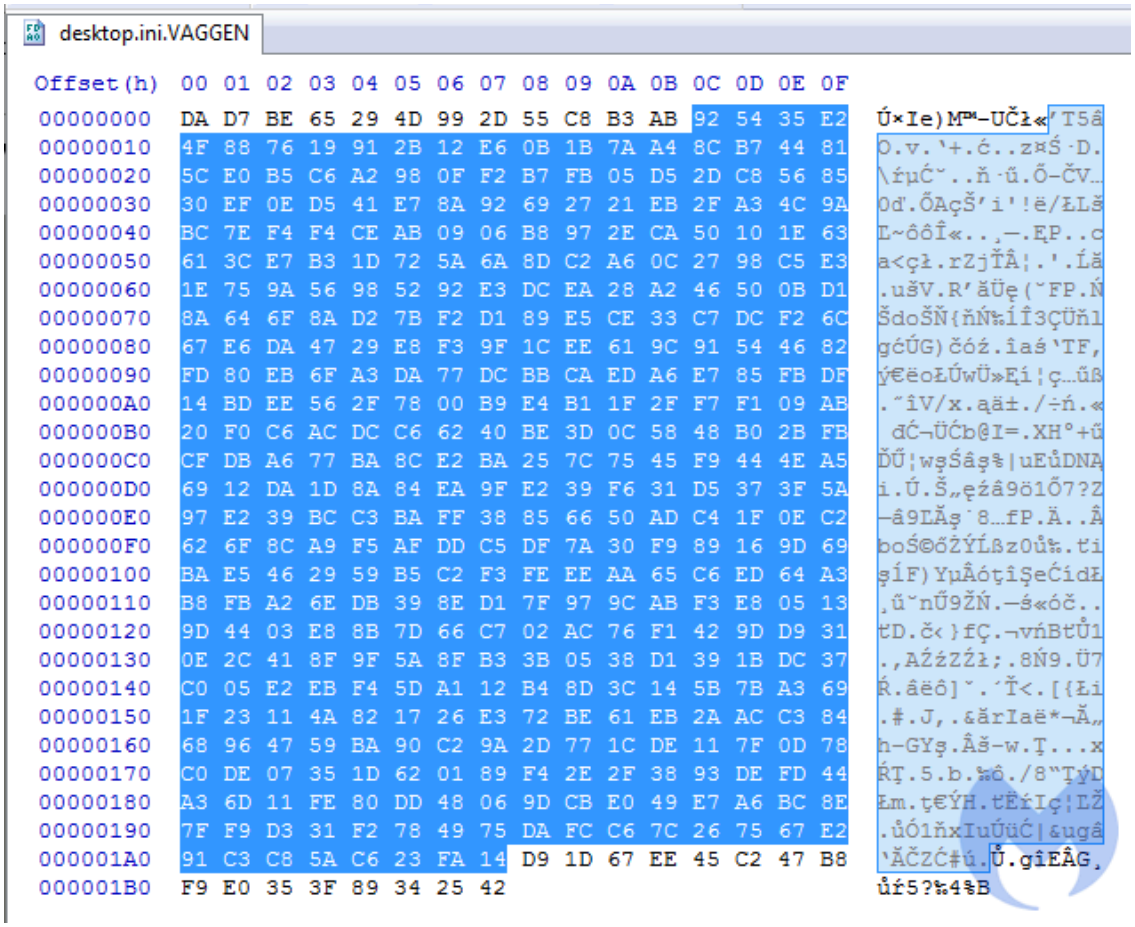


The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

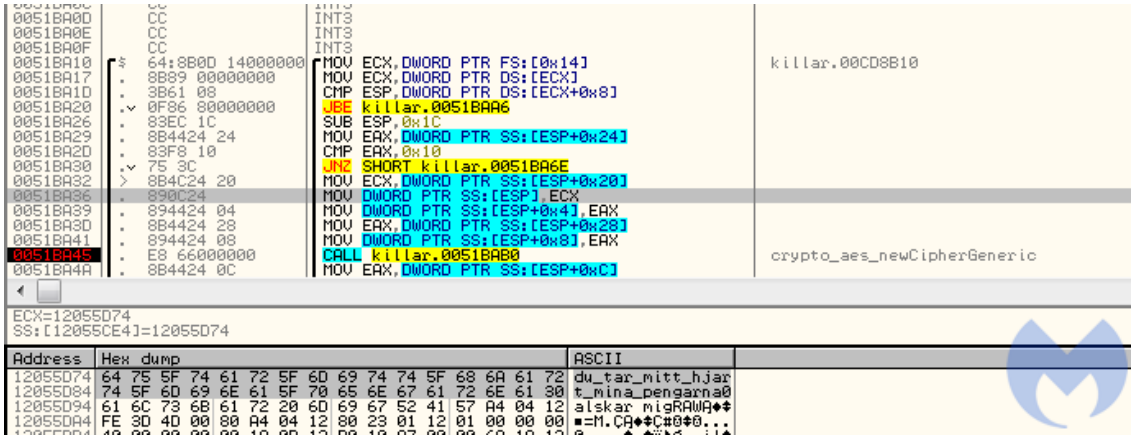


The content of the output file (with .VAGGEN extension) contains:

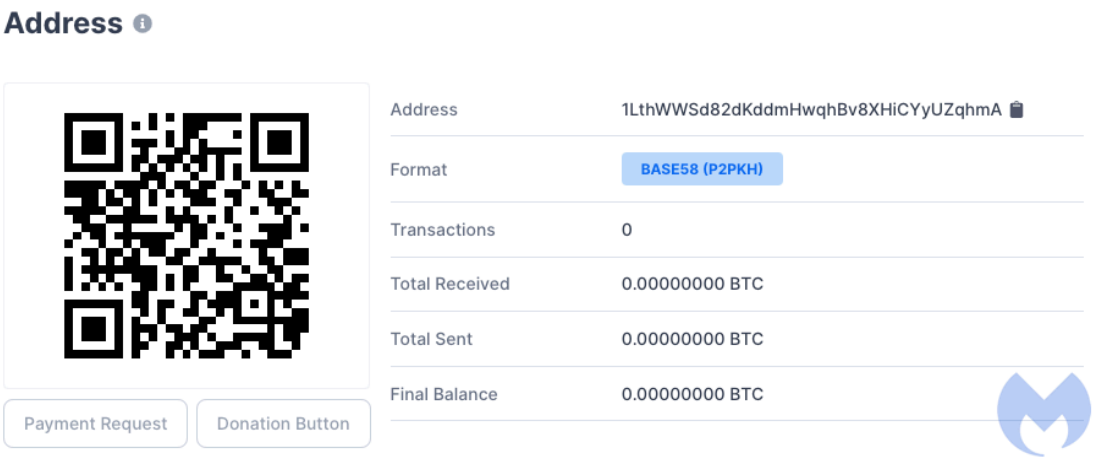
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

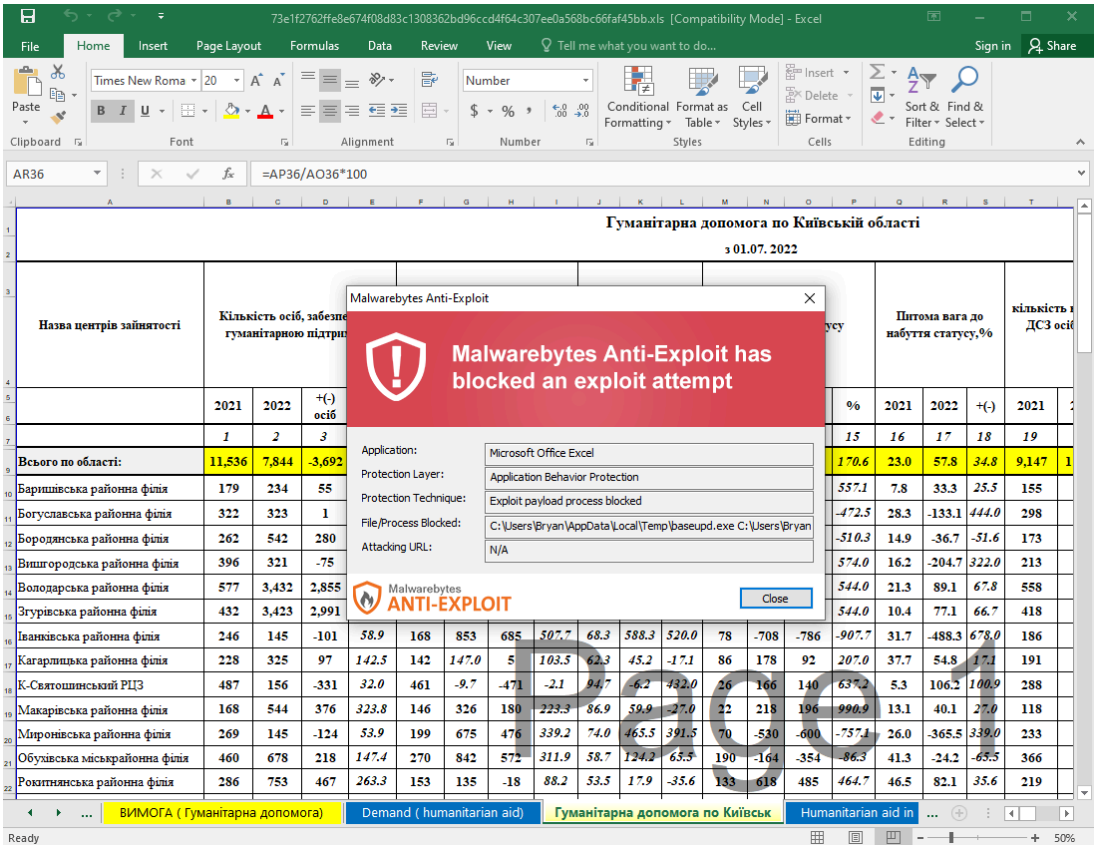
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
46  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
47    runtime_morestack_noctxt();
48  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
49  ((void (*)(void))loc_45CDA0)();
50  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
51  v38 = v4;
52  v30 = v5;
53  v31 = v6;
54  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
55  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
56  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
57  v32 = v10;
58  v11 = *(void (__cdecl **)(int))(v10 + 16);
59  v39 = v12;
60  v11(v12);
61  v29 = v13;
62  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
63  v40 = v15;
64  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
65  v17 = v27;
66  if ( v27 )
67  {
68    if ( v27 )
69      v17 = *(_DWORD *)(v27 + 4);
70    v43 = v17;
71    v44 = v28;
72    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
73  }
```

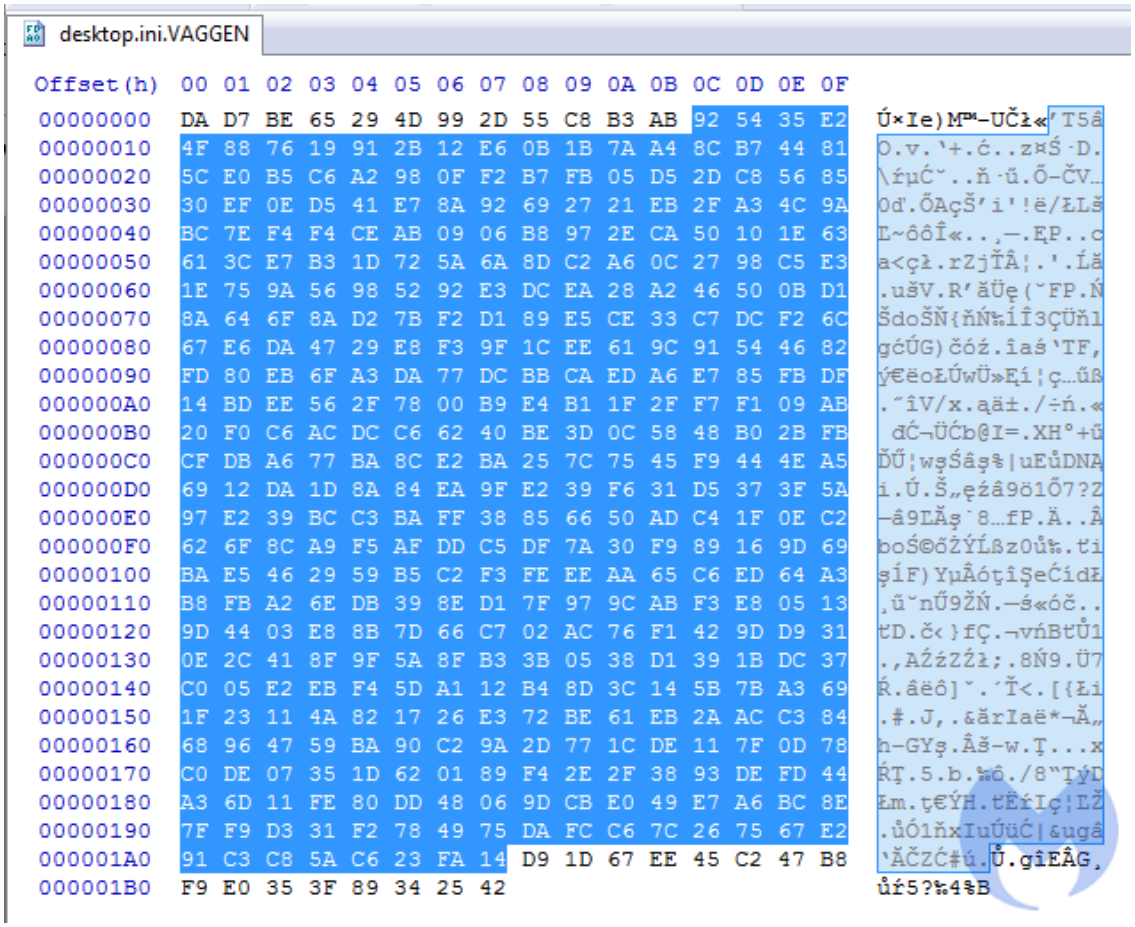
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



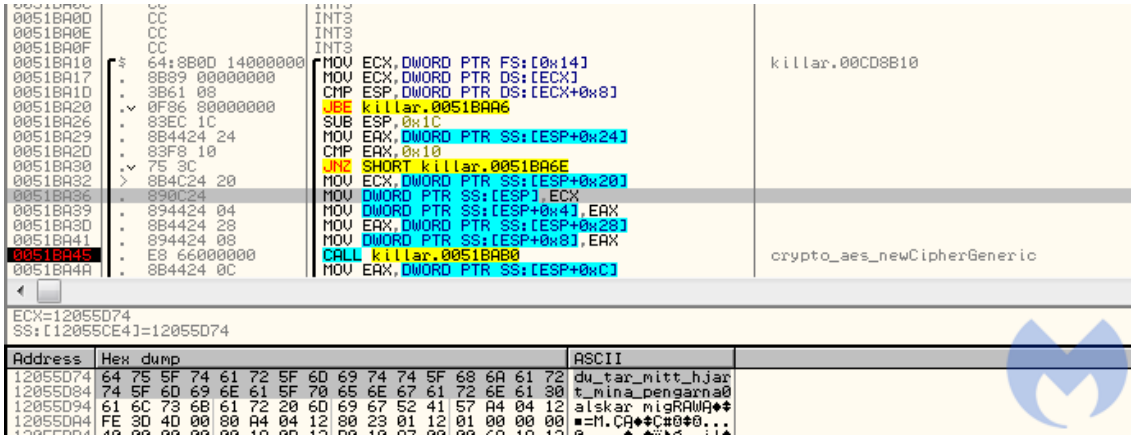
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

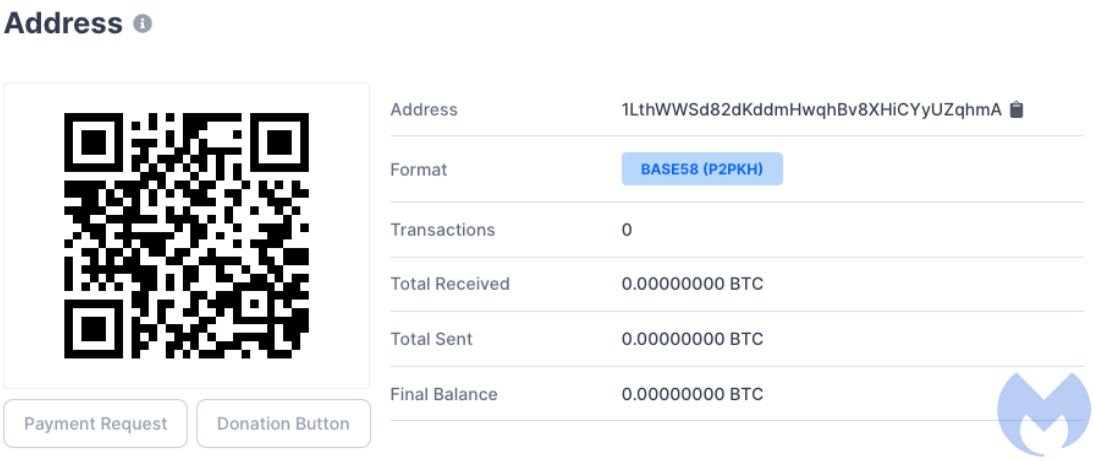
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

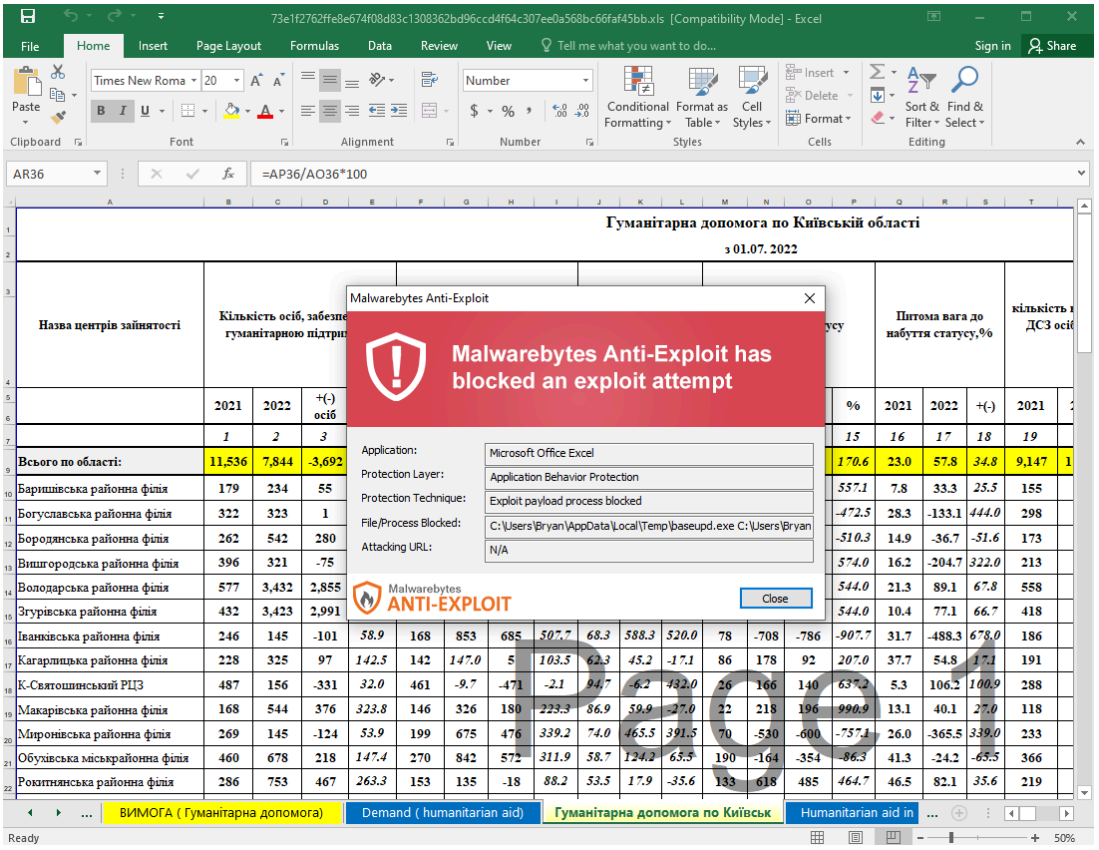
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

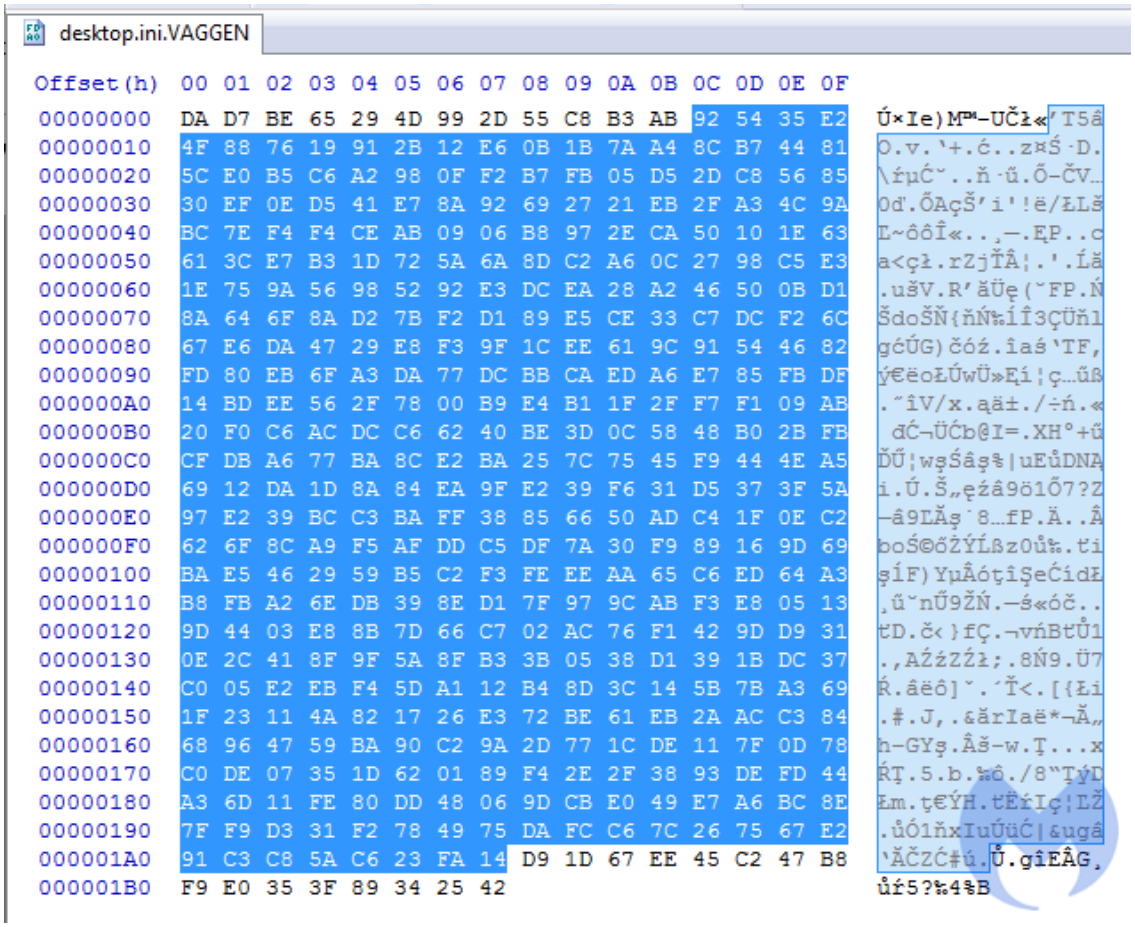
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

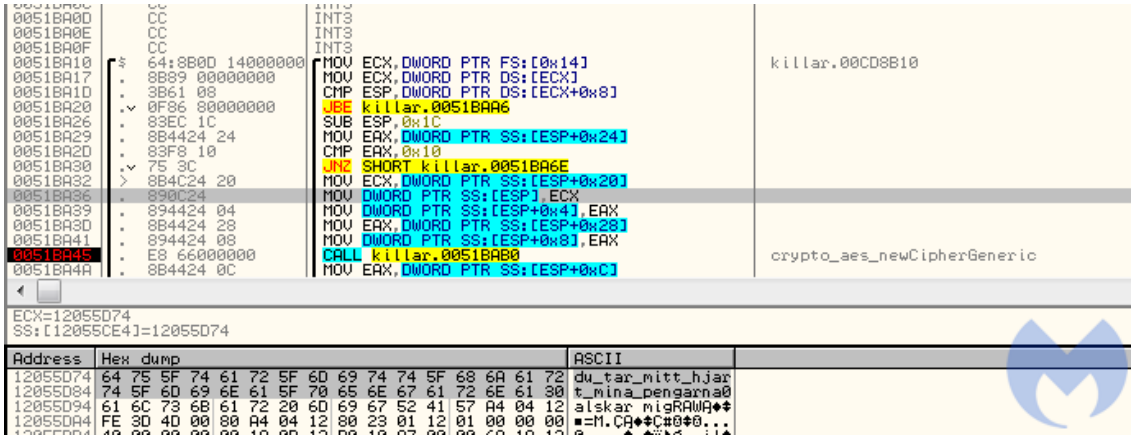


The content of the output file (with .VAGGEN extension) contains:

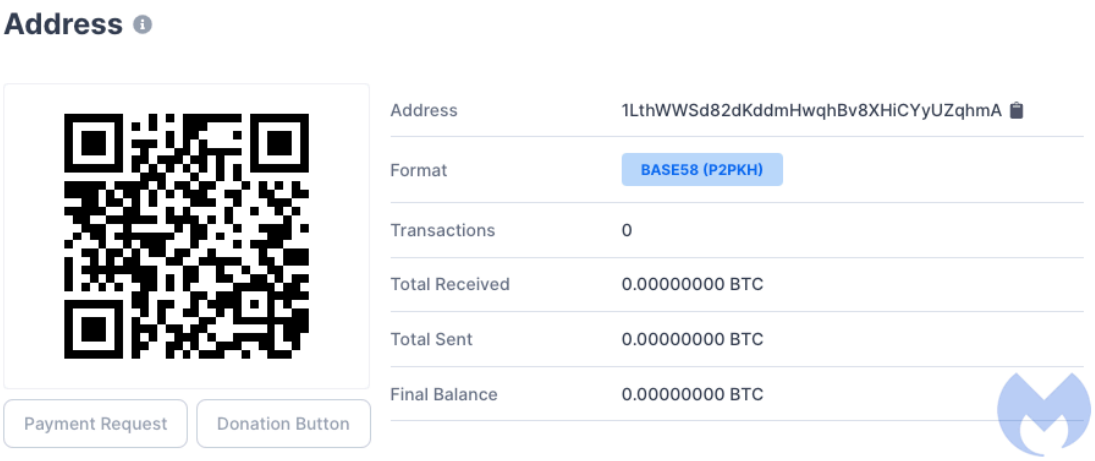
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

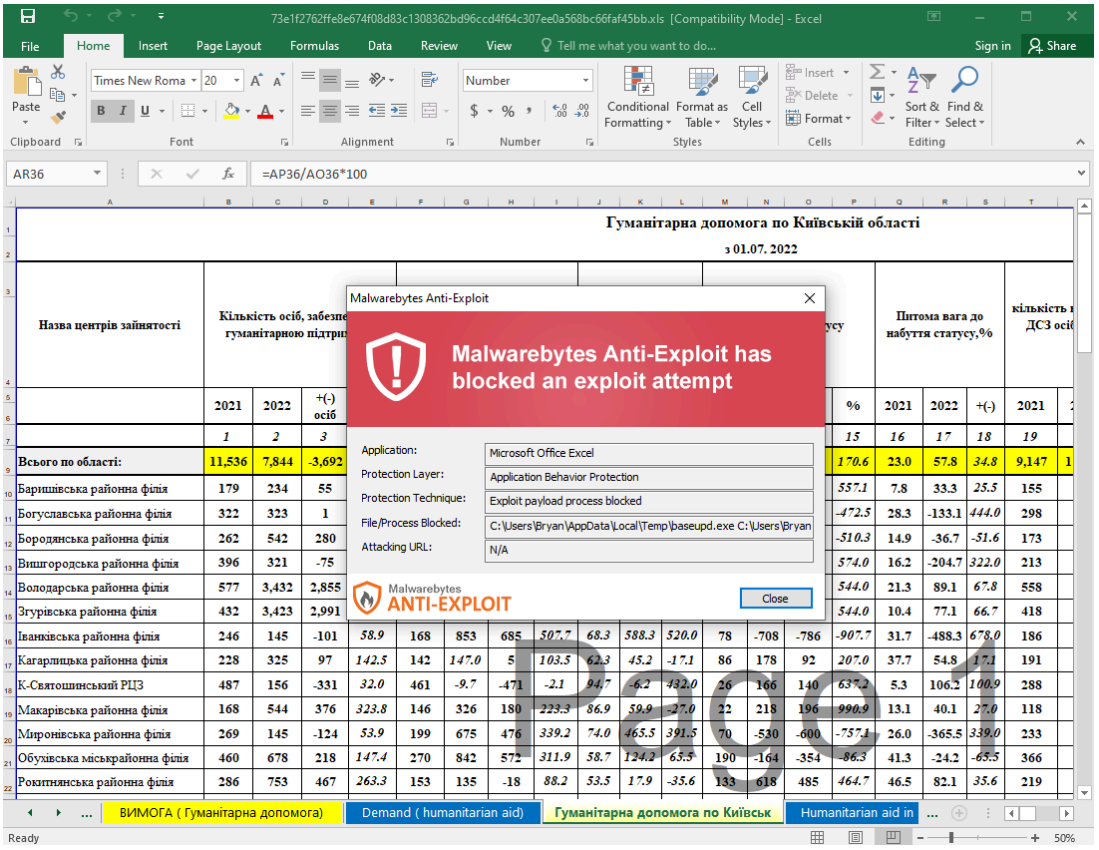
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

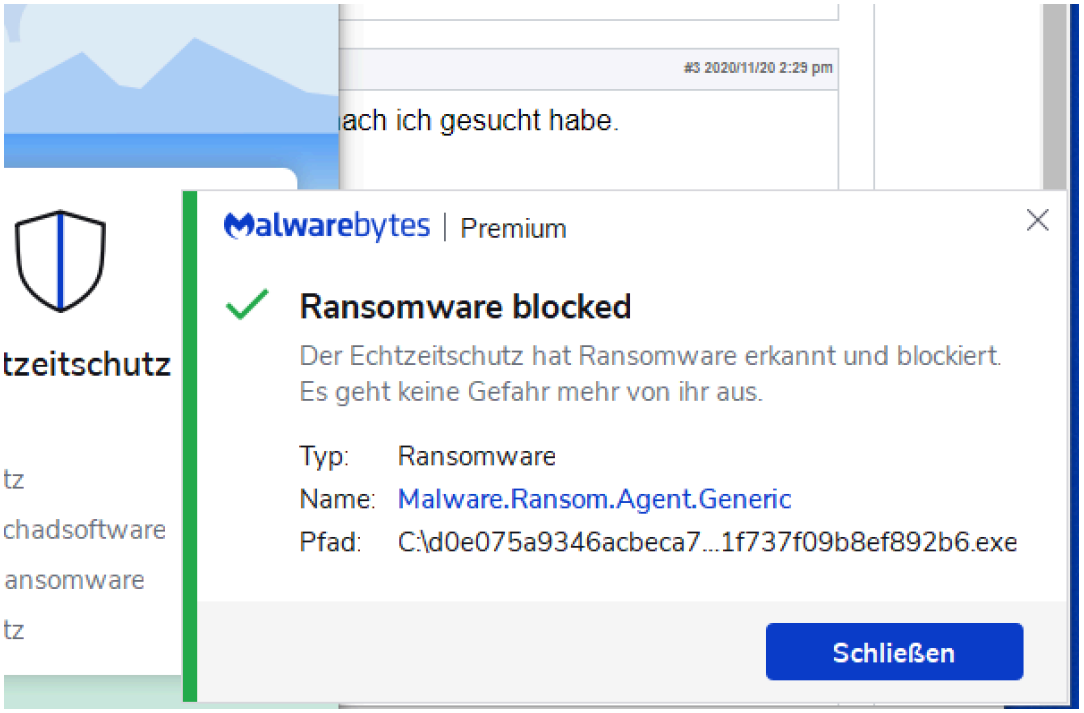
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

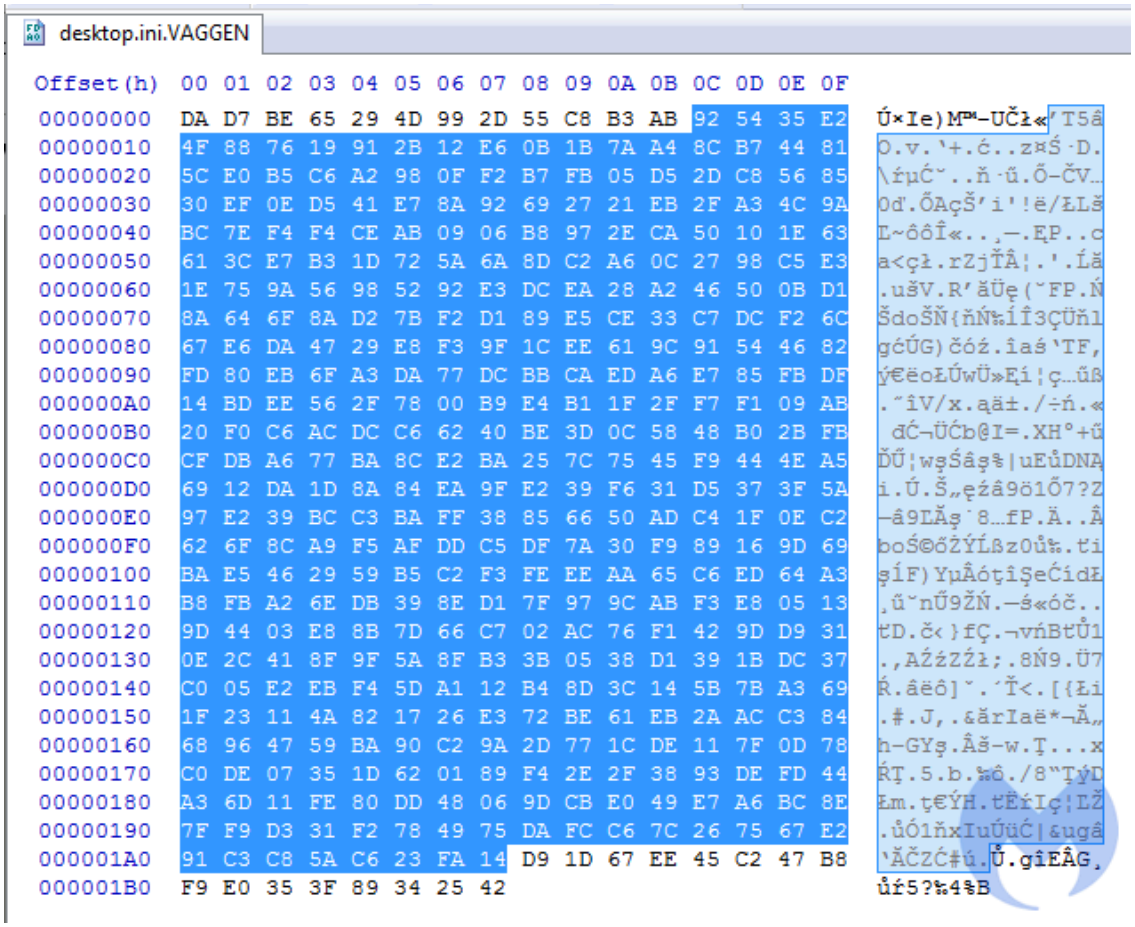
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



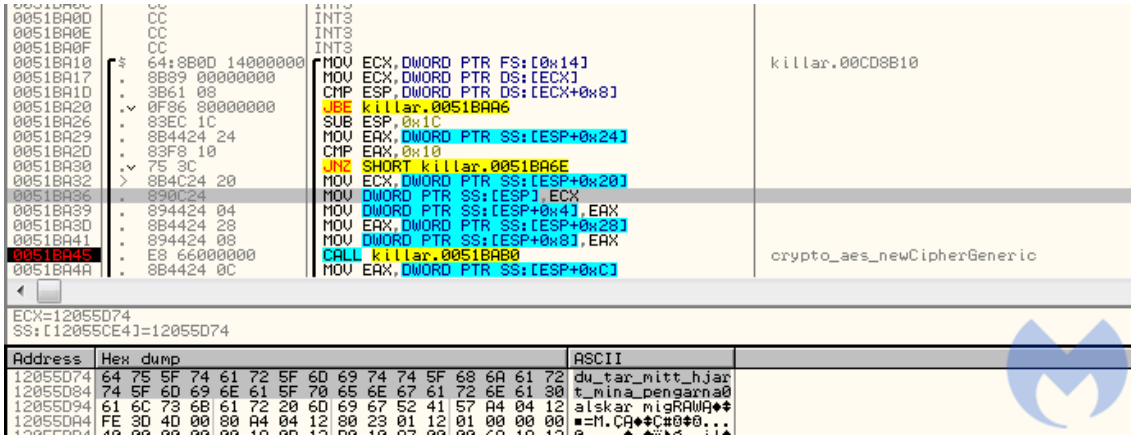
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

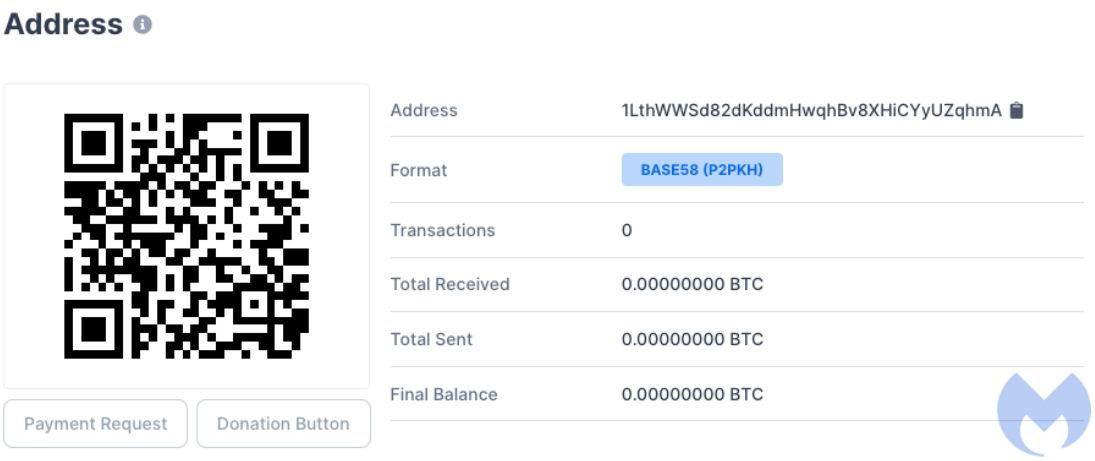
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

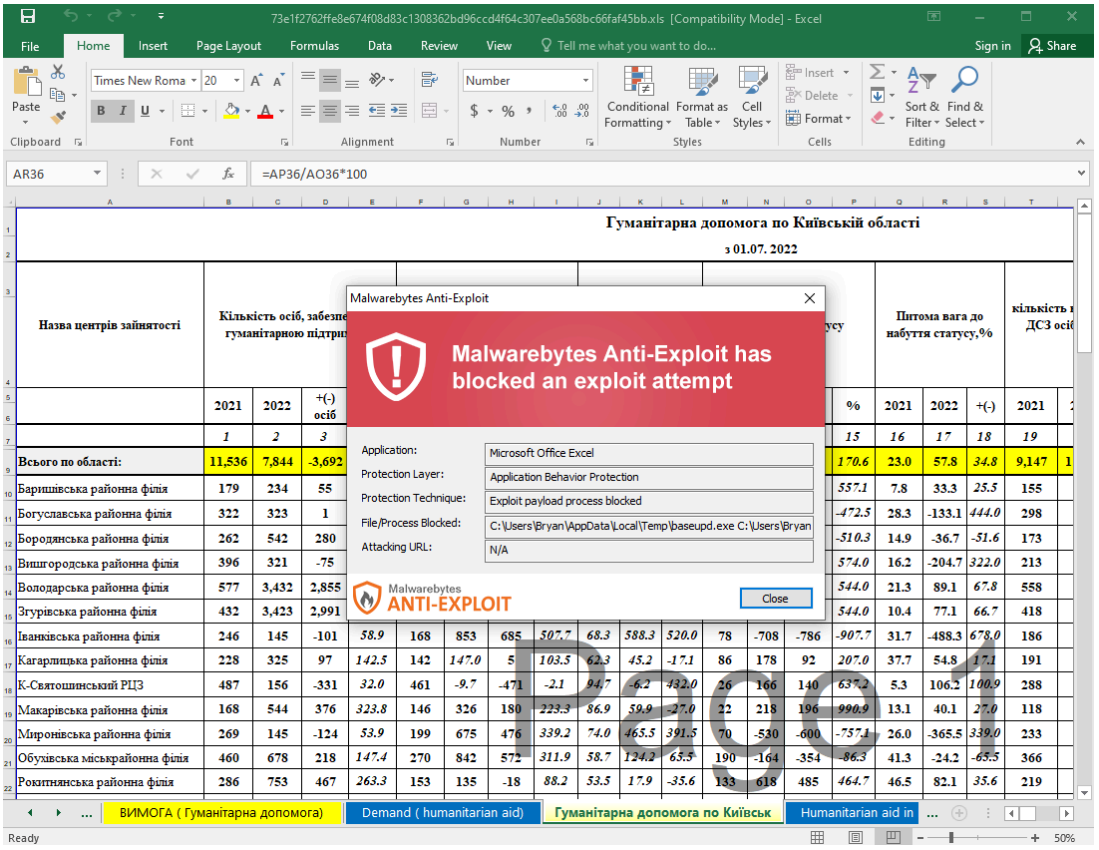
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

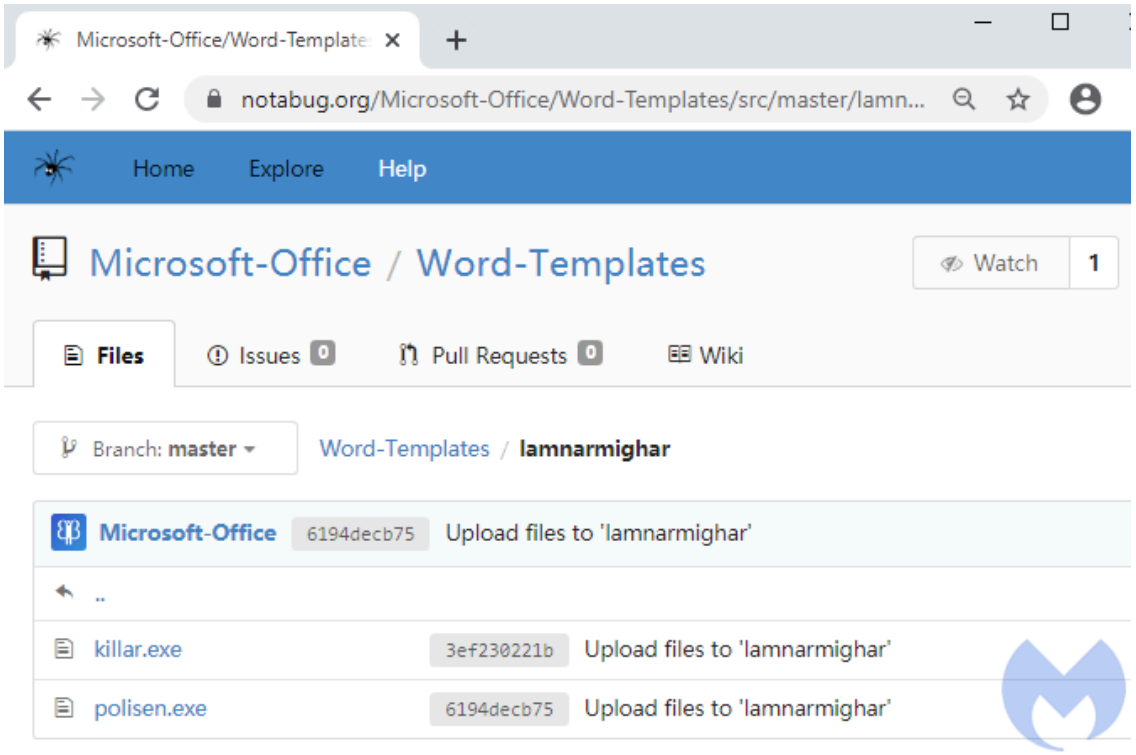
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



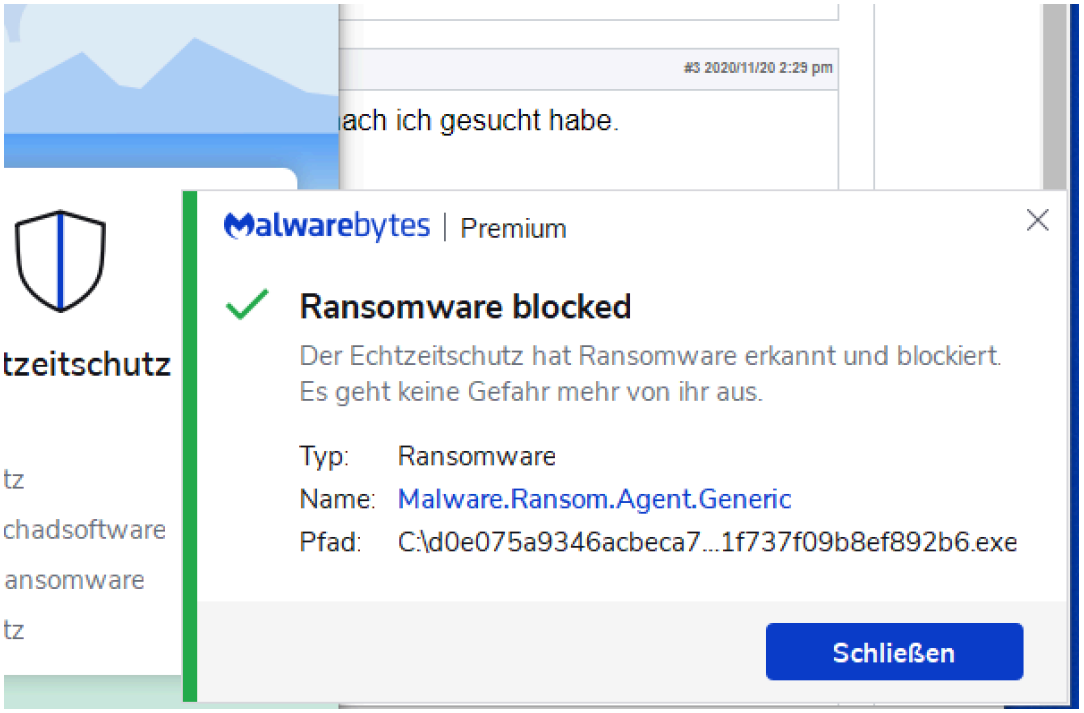
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

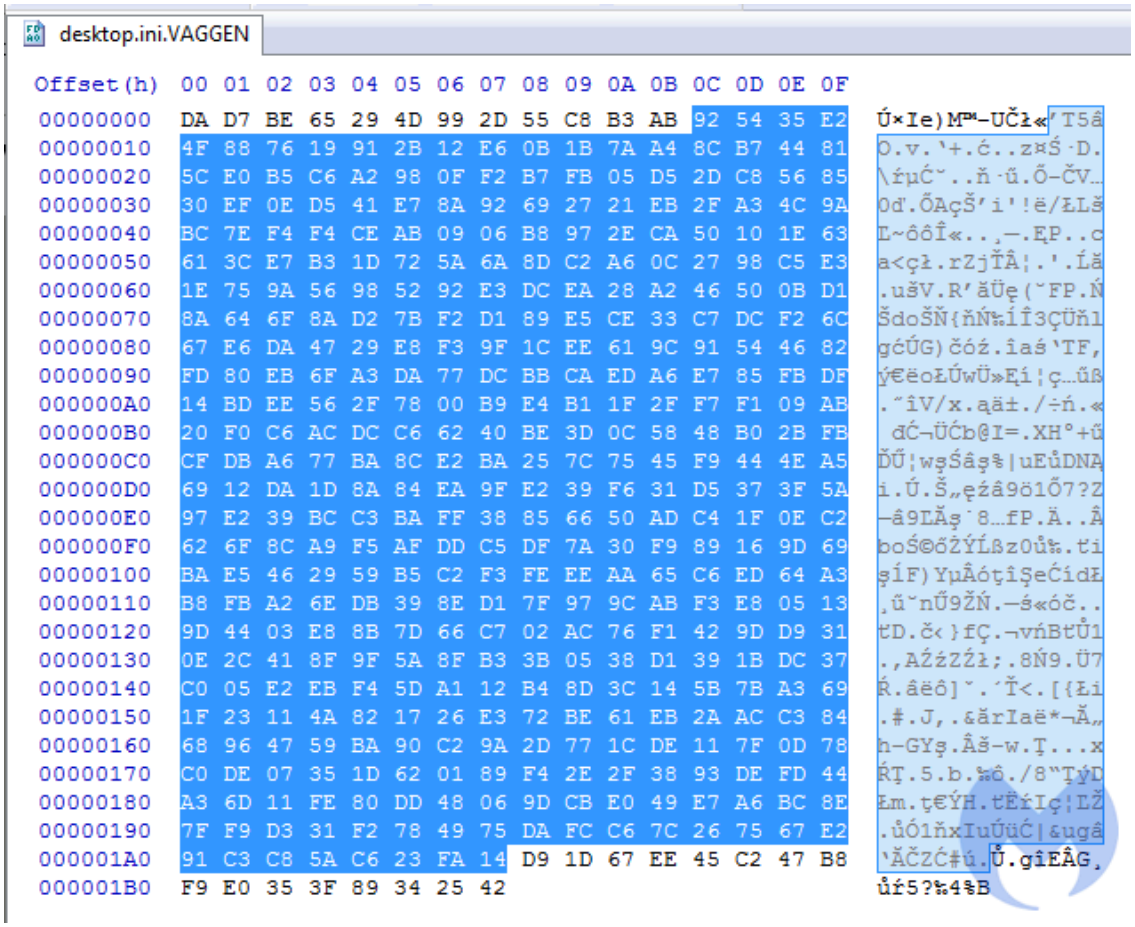
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



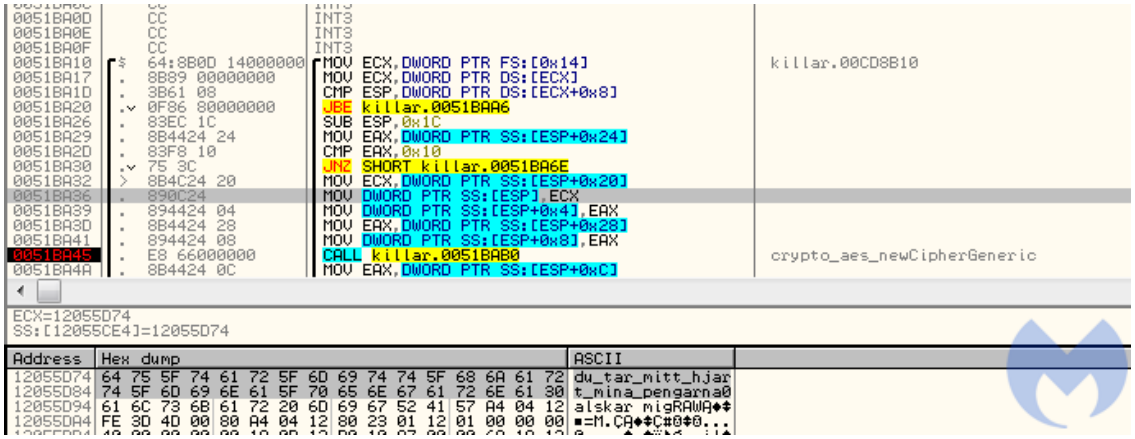
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

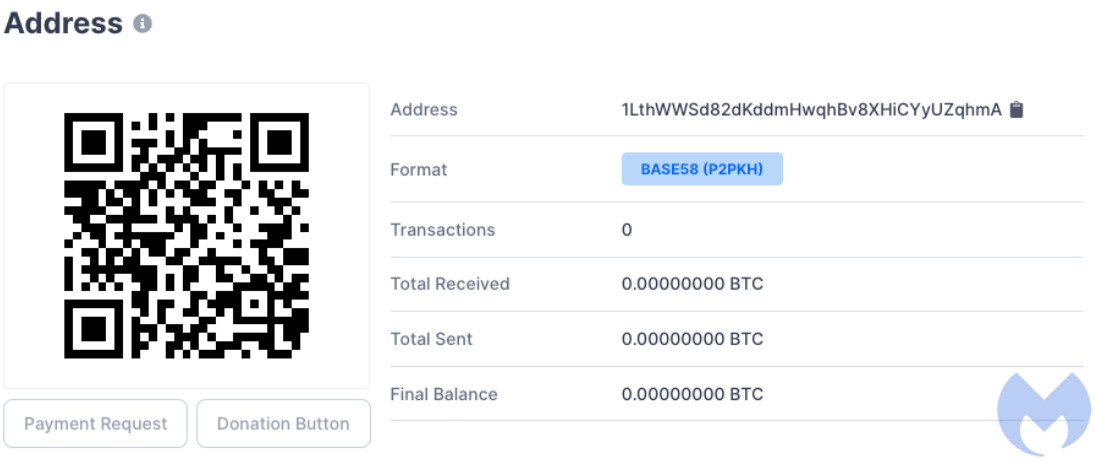
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

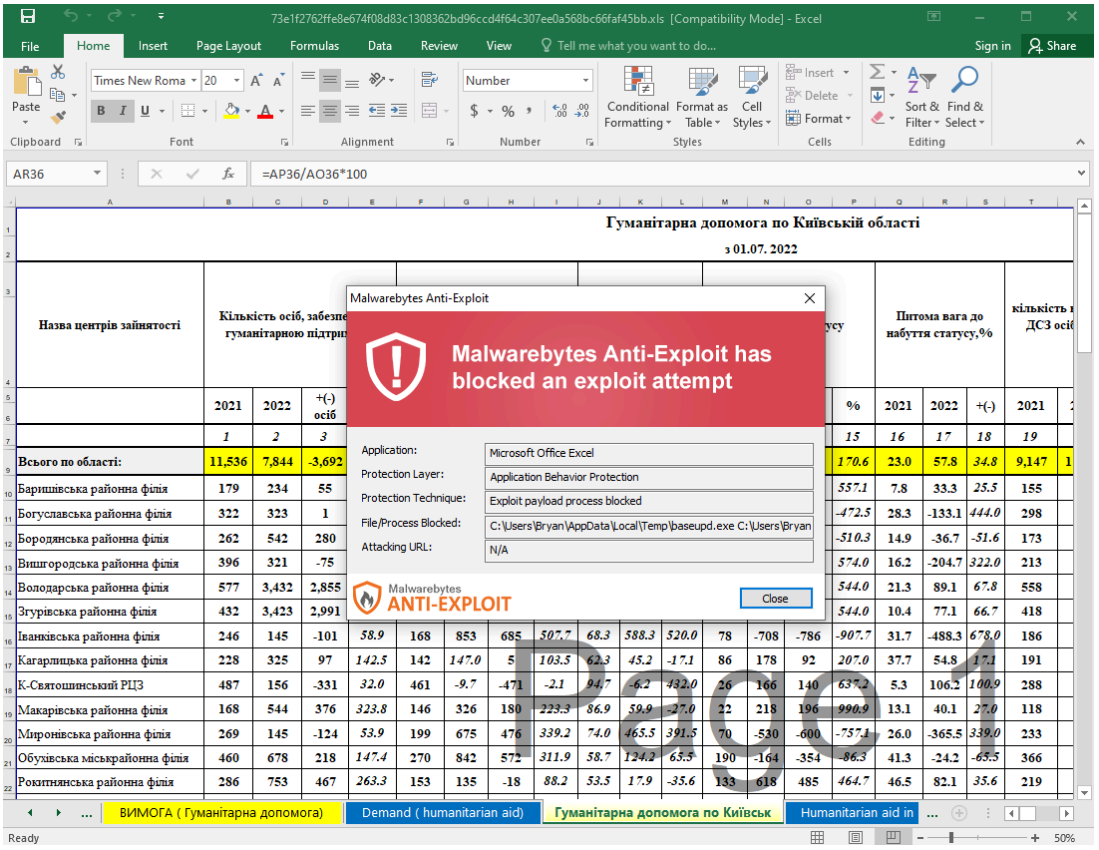
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

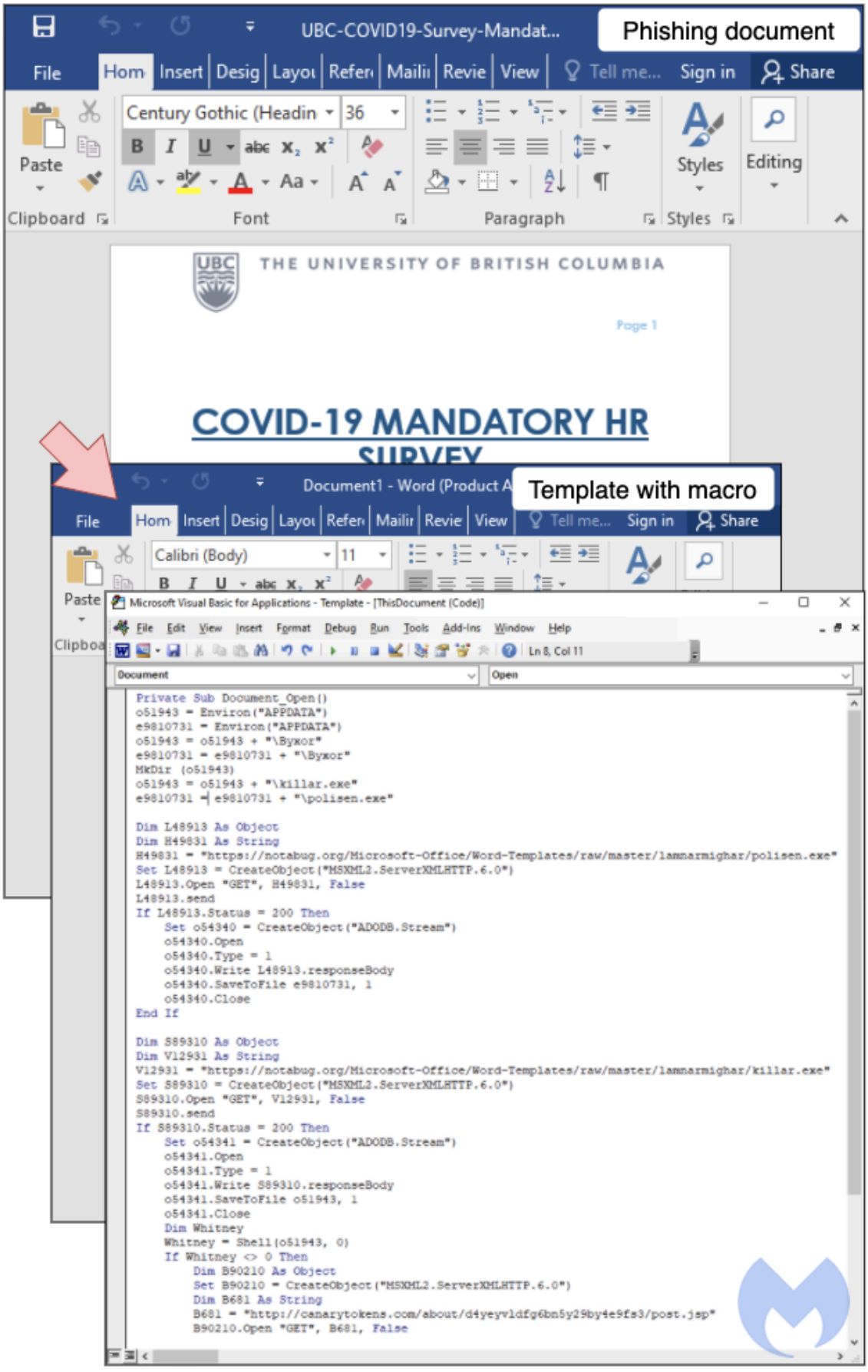
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

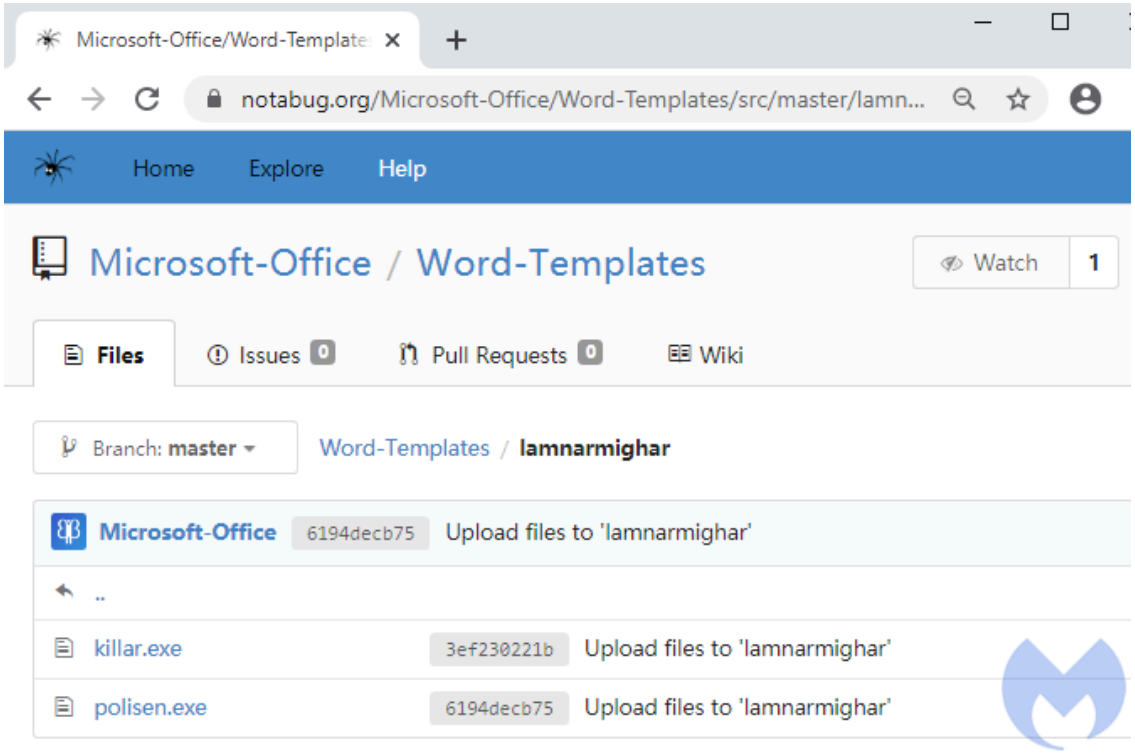
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



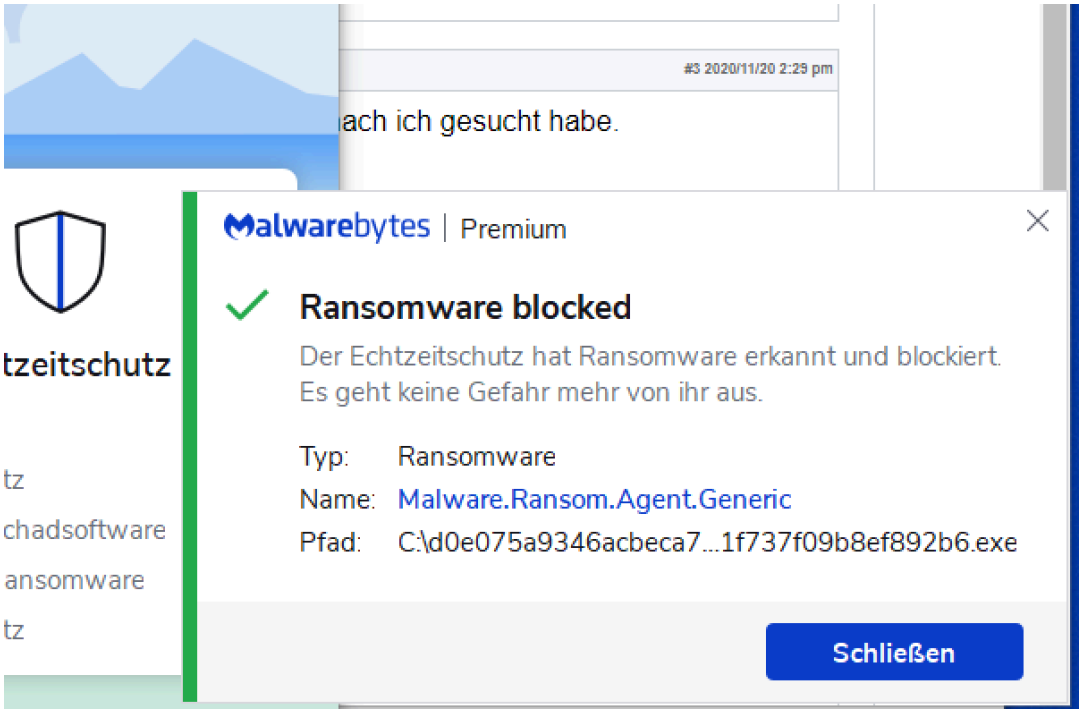
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

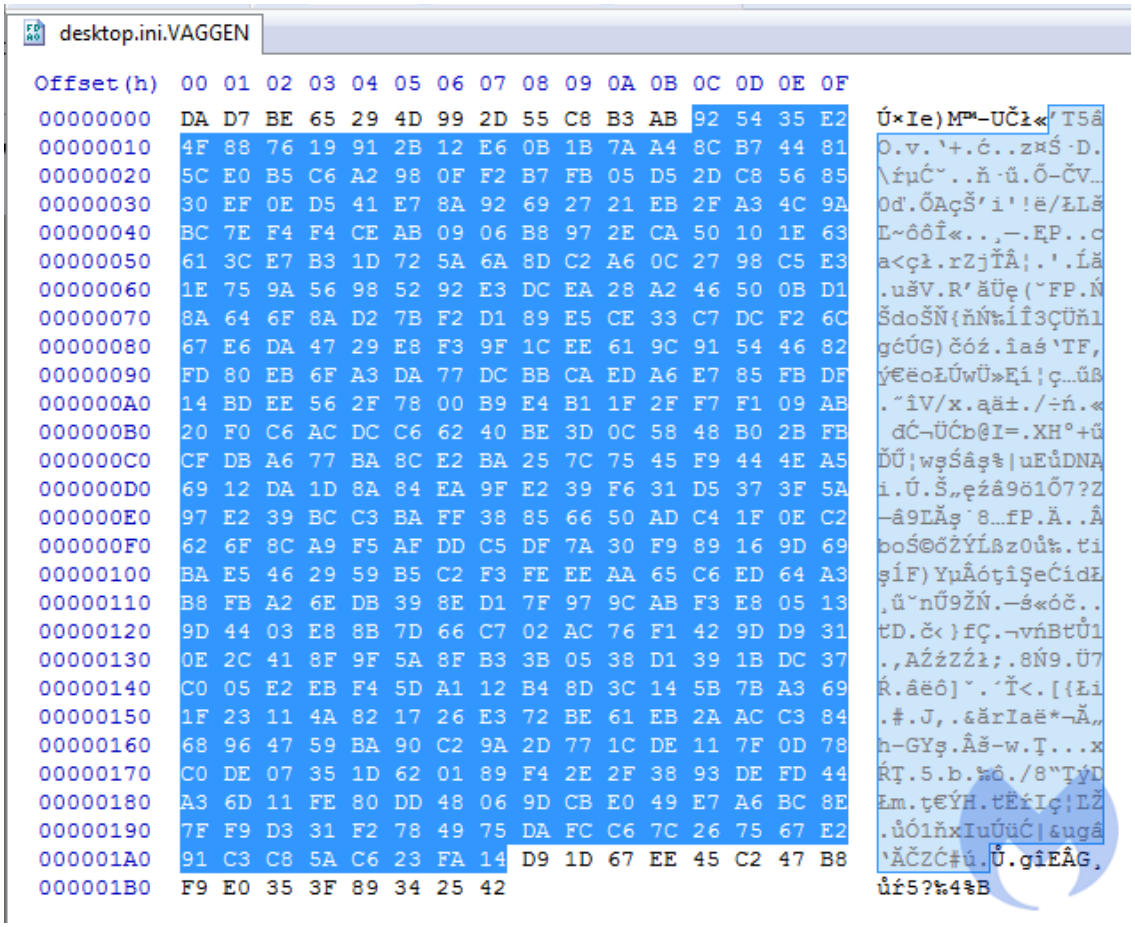
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



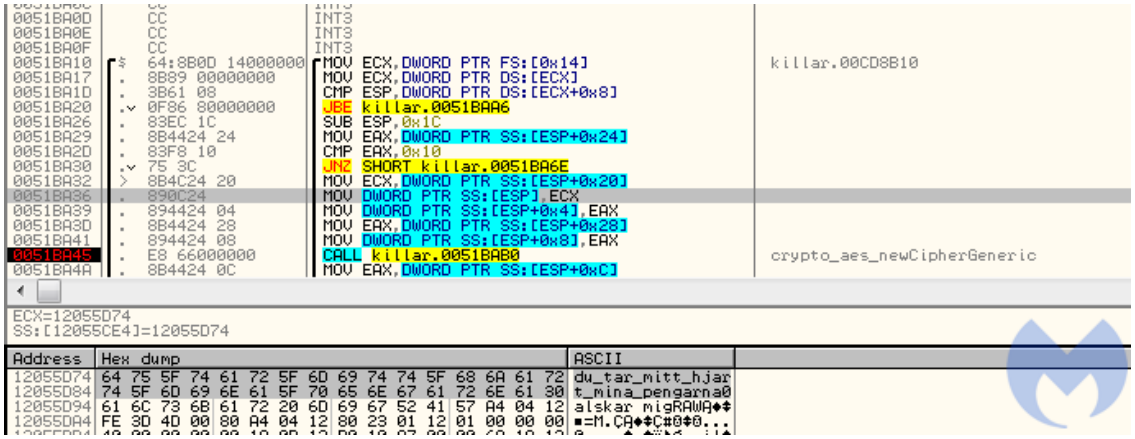
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

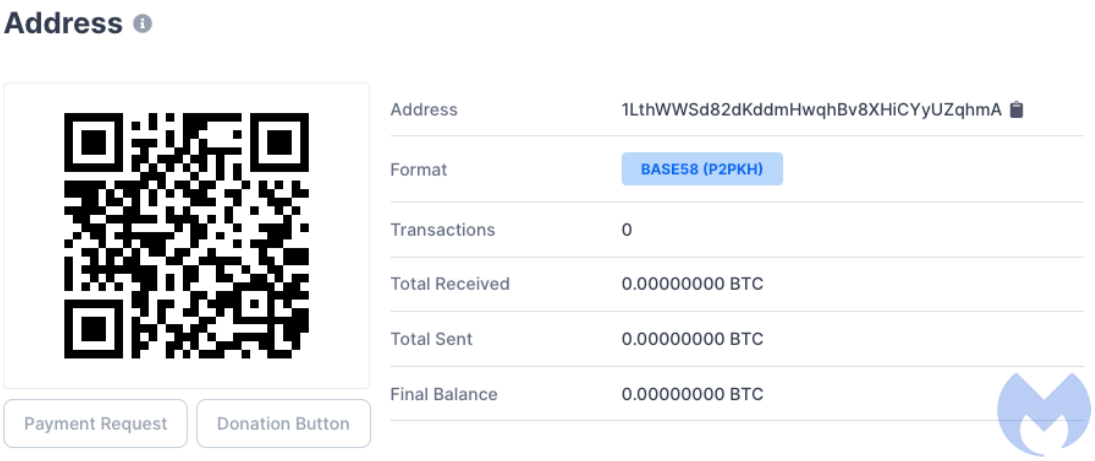
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

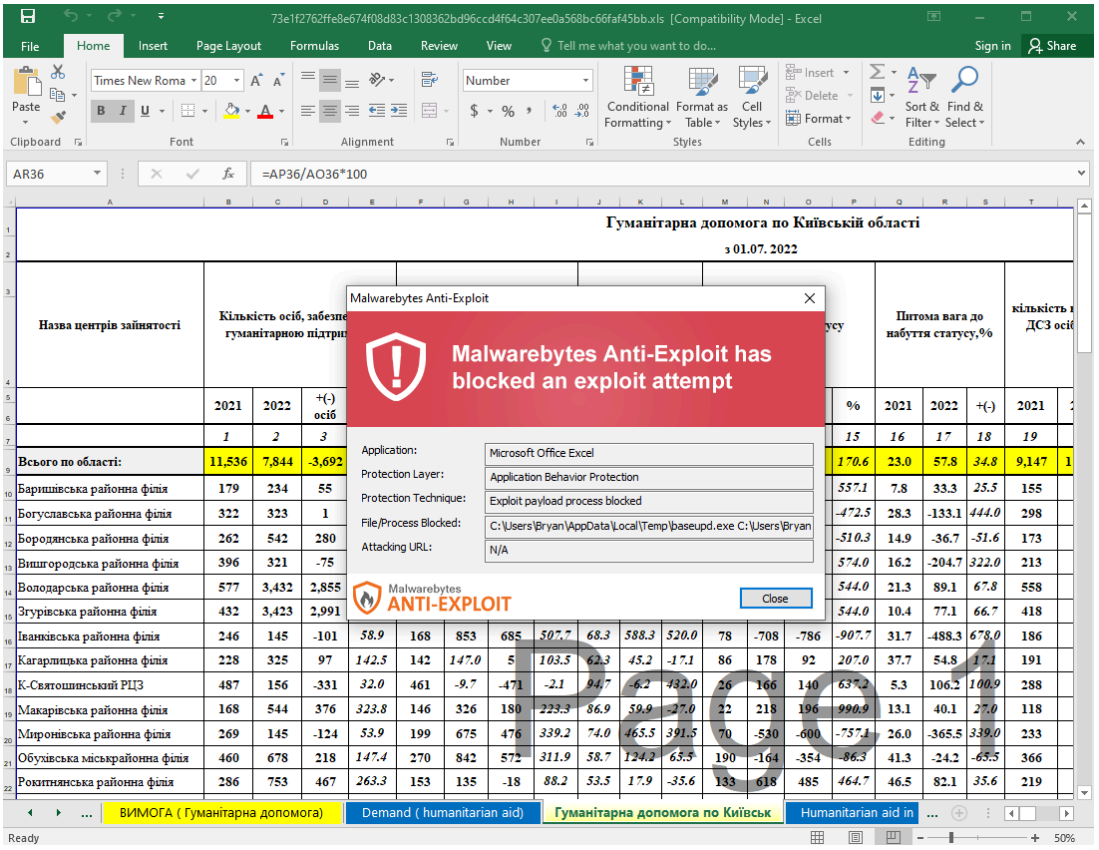
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

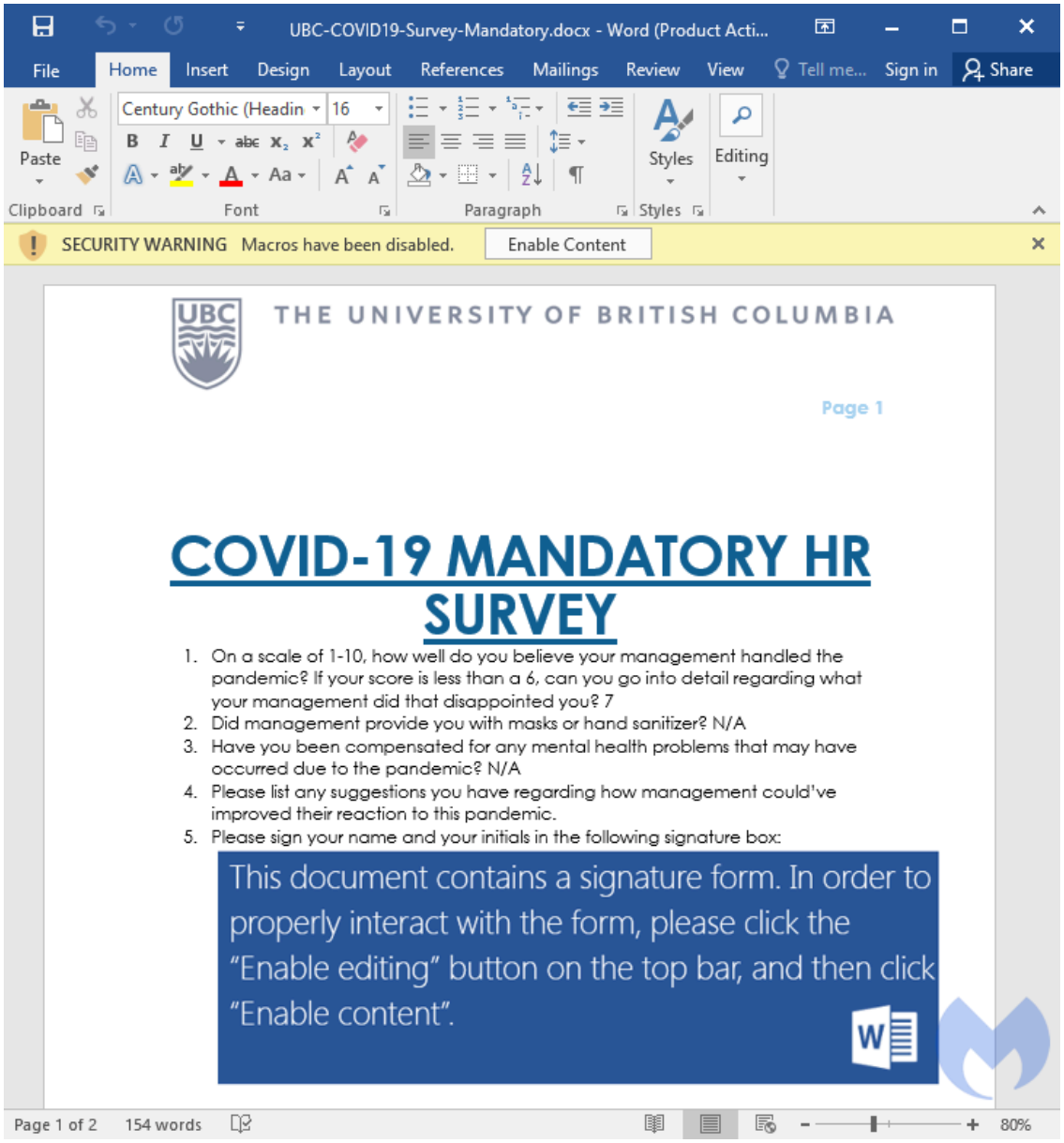
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

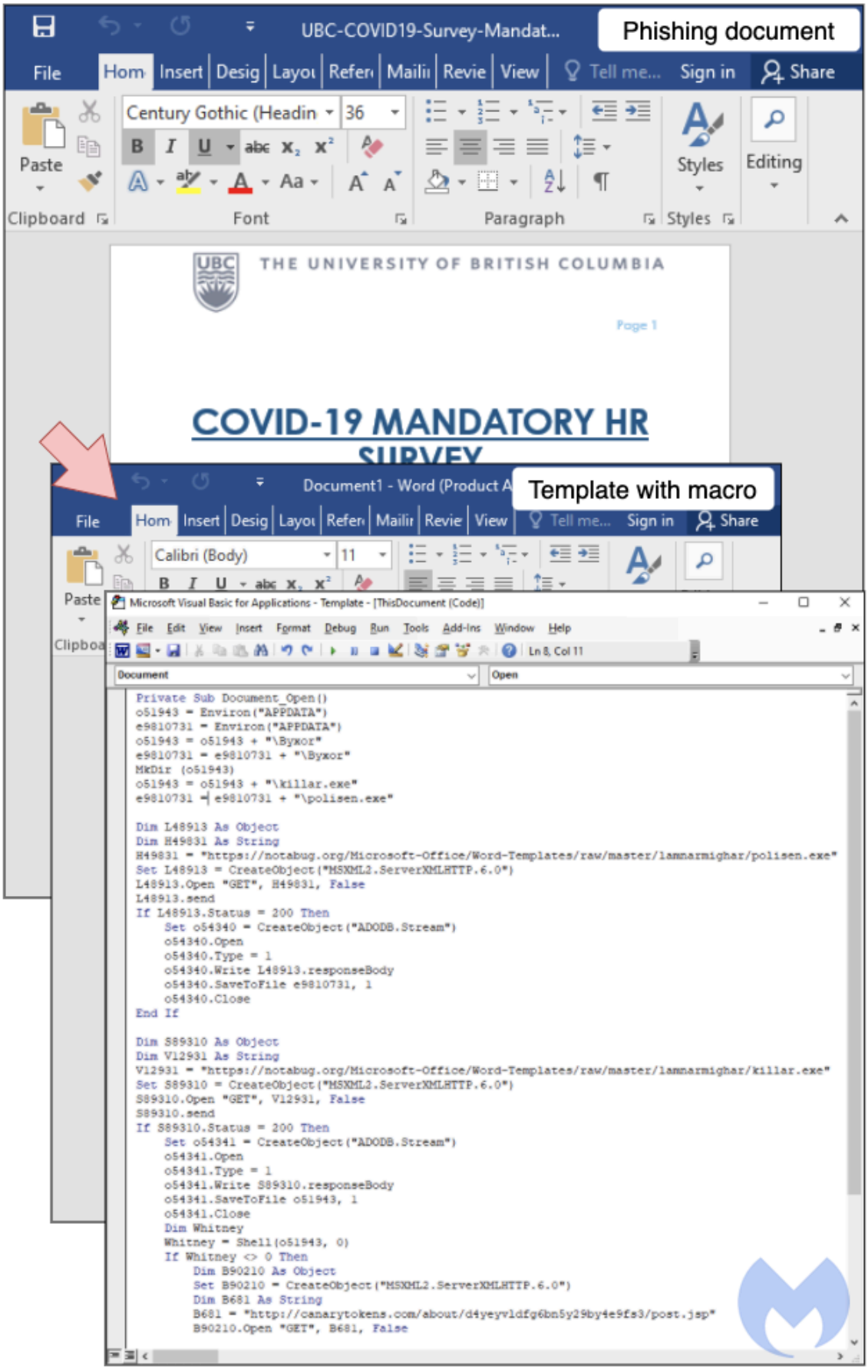
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

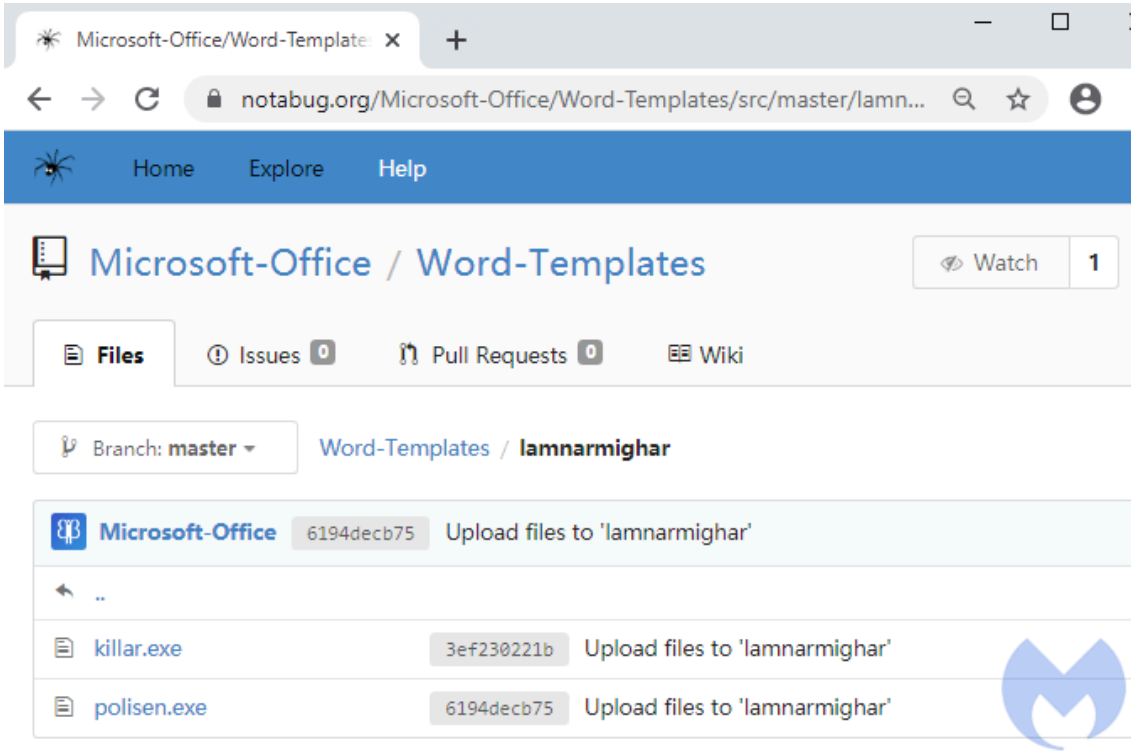
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



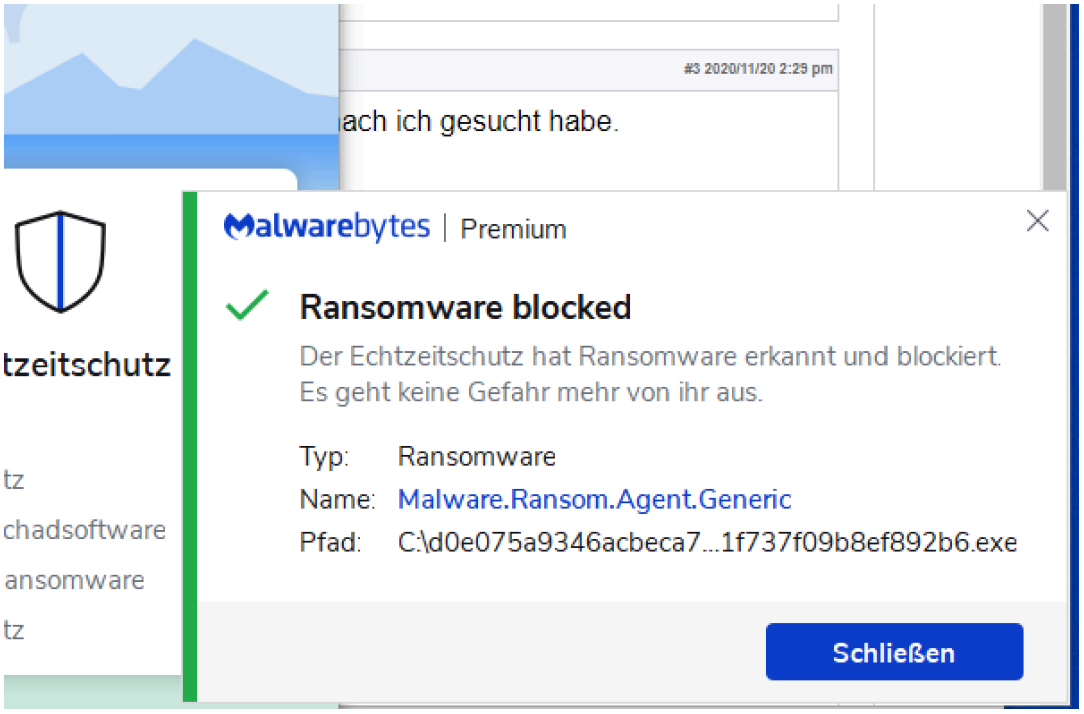
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

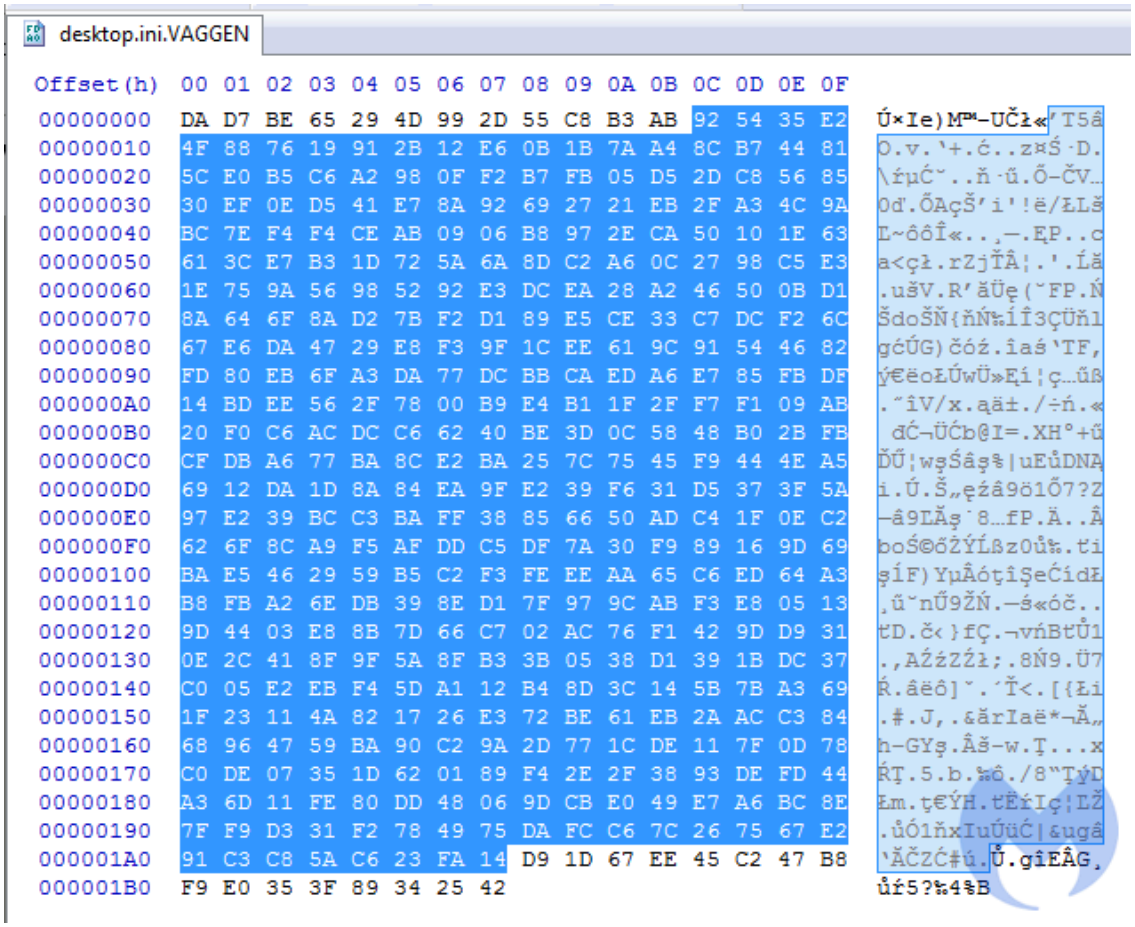
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

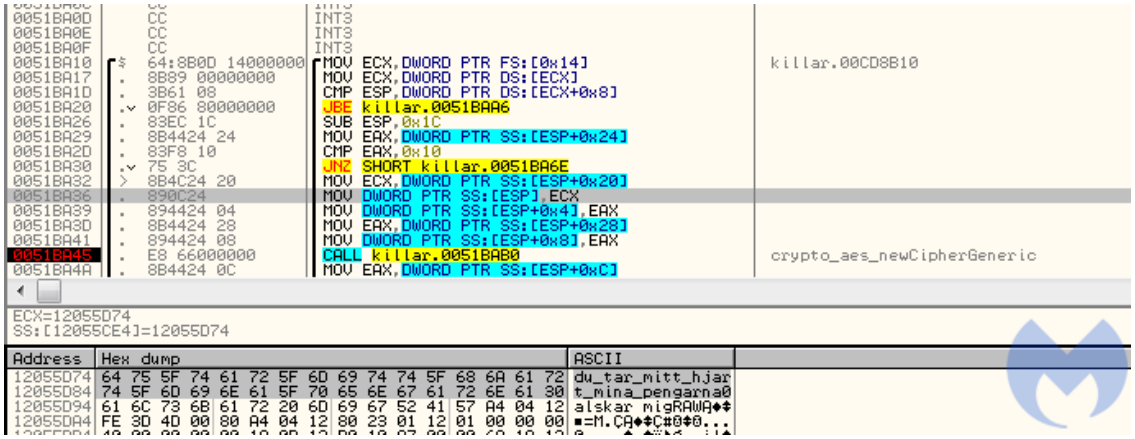


The content of the output file (with .VAGGEN extension) contains:

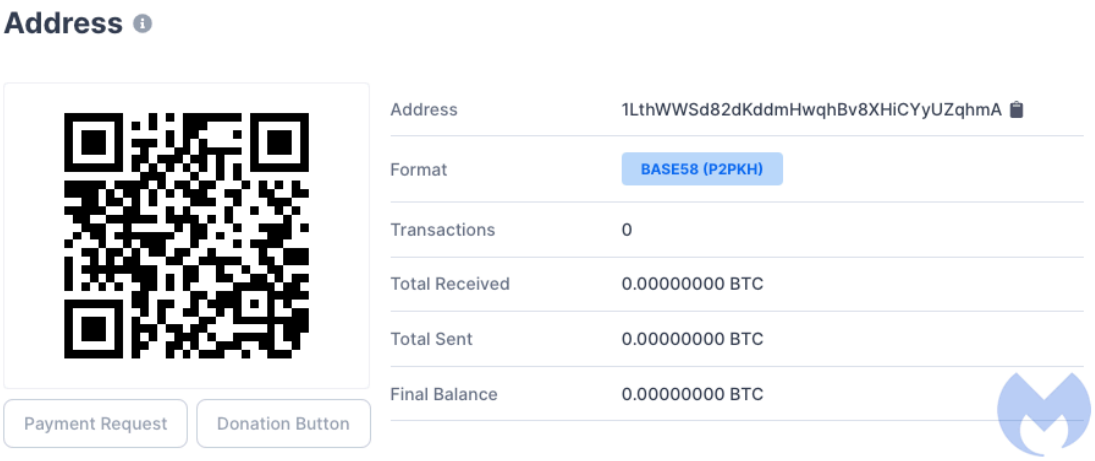
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

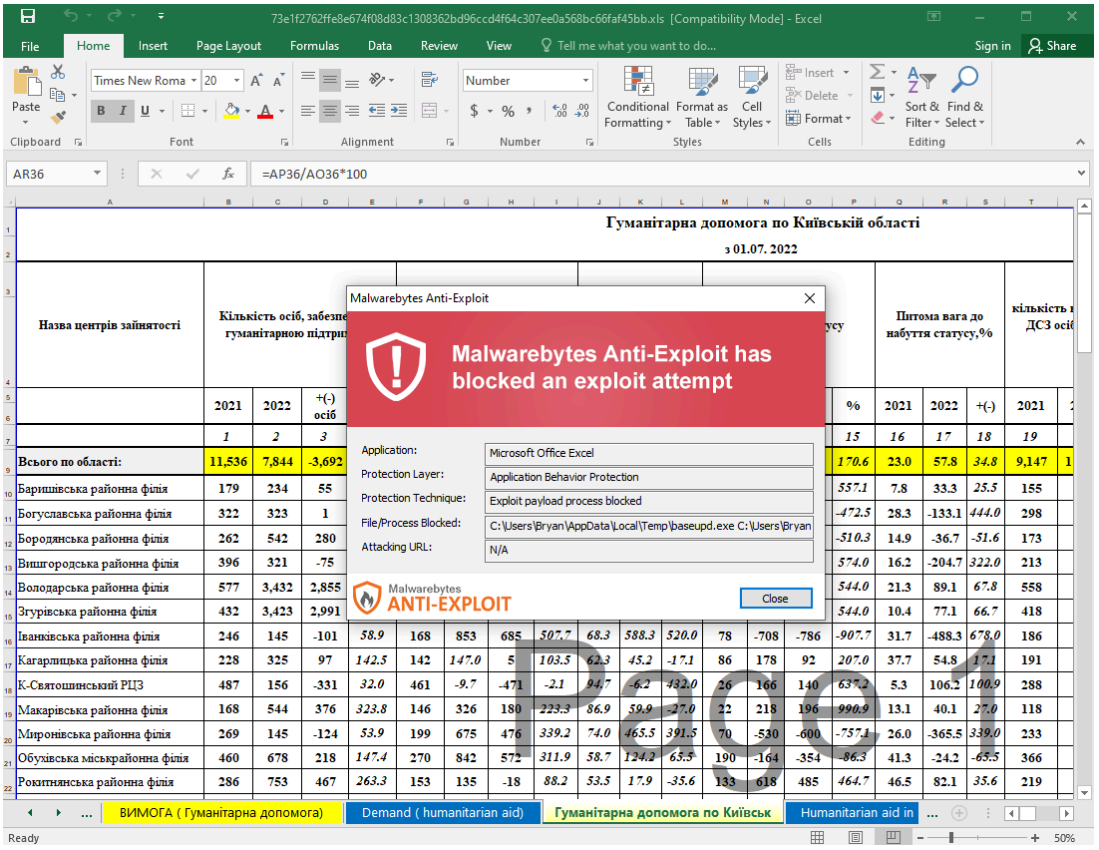
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

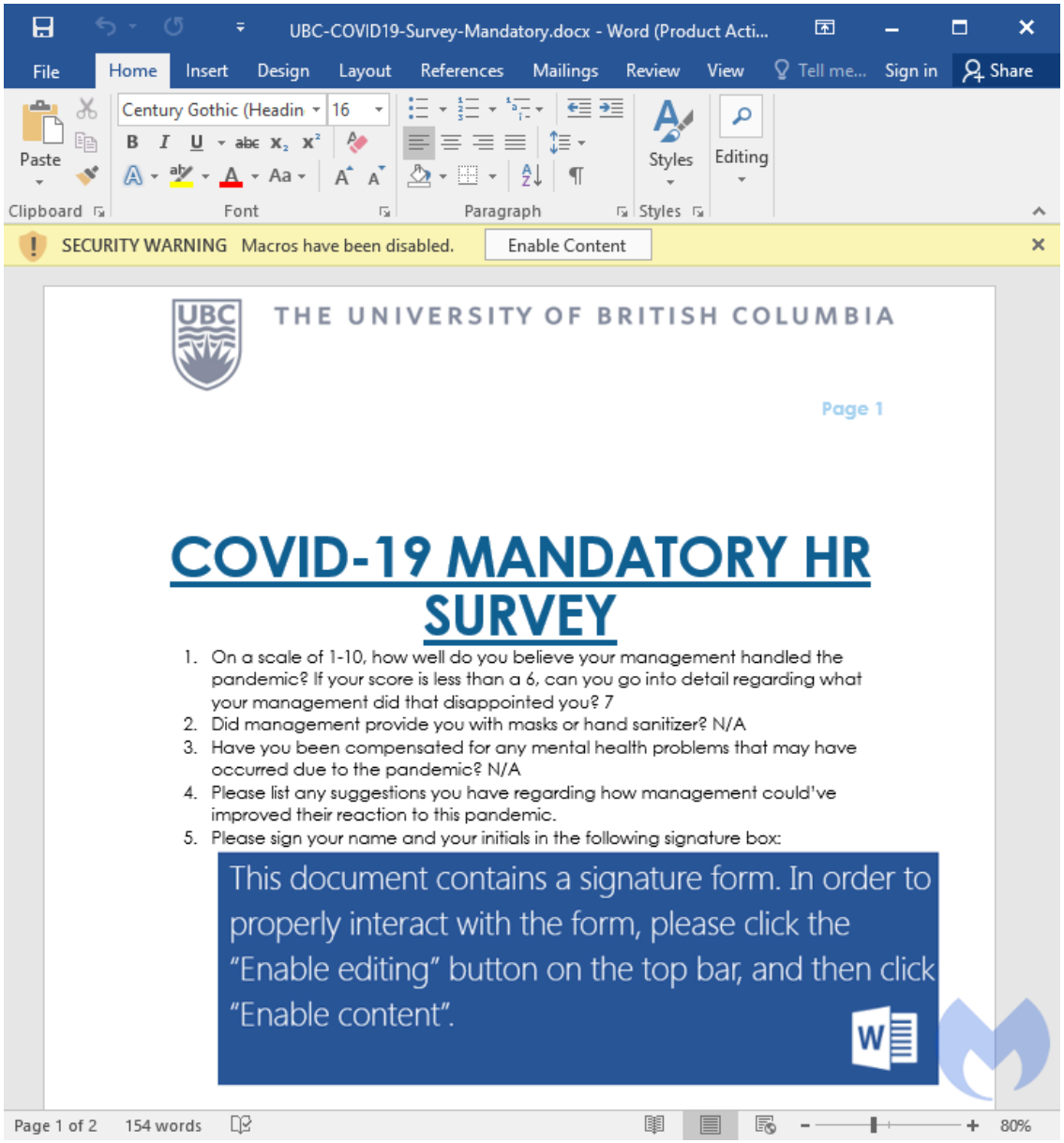
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

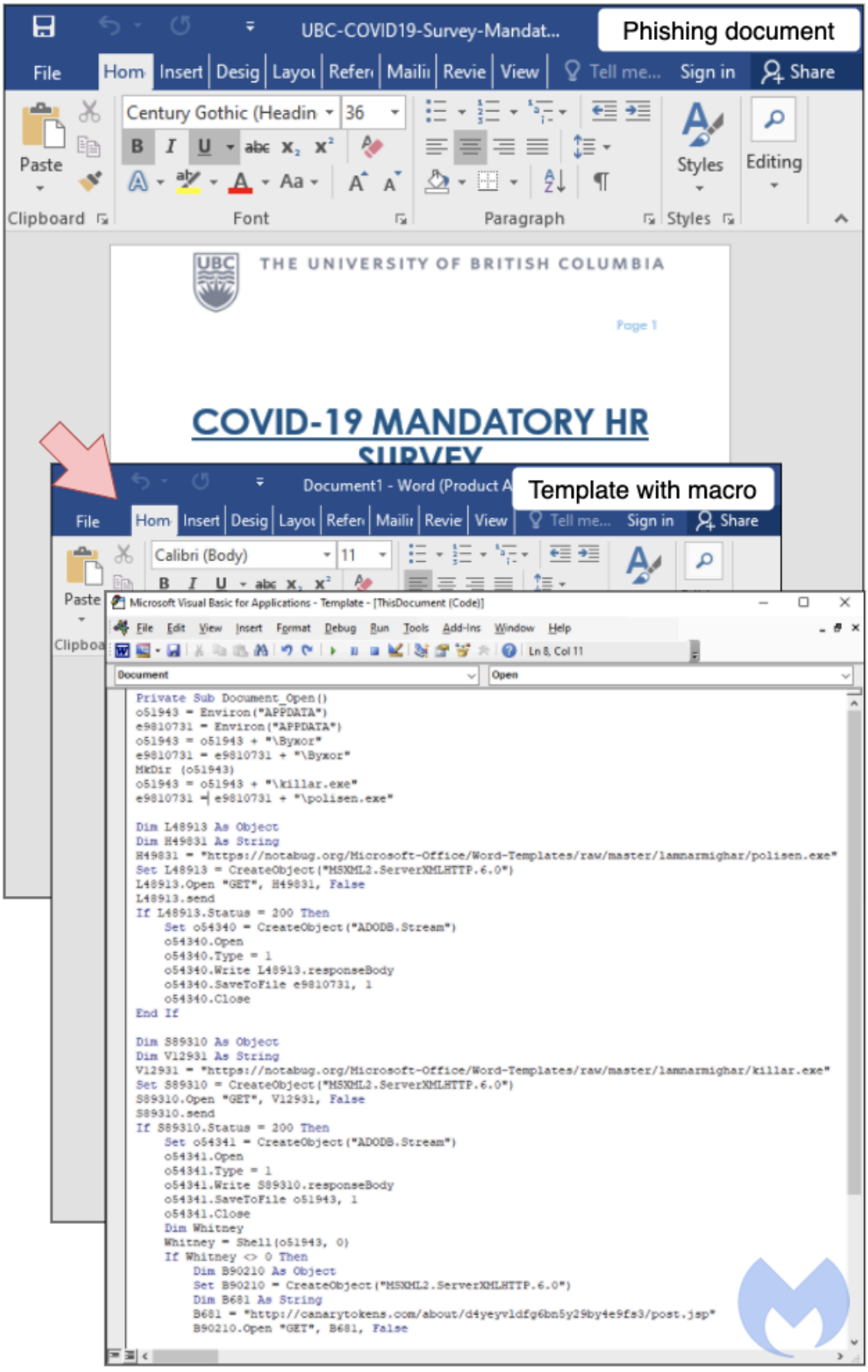
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

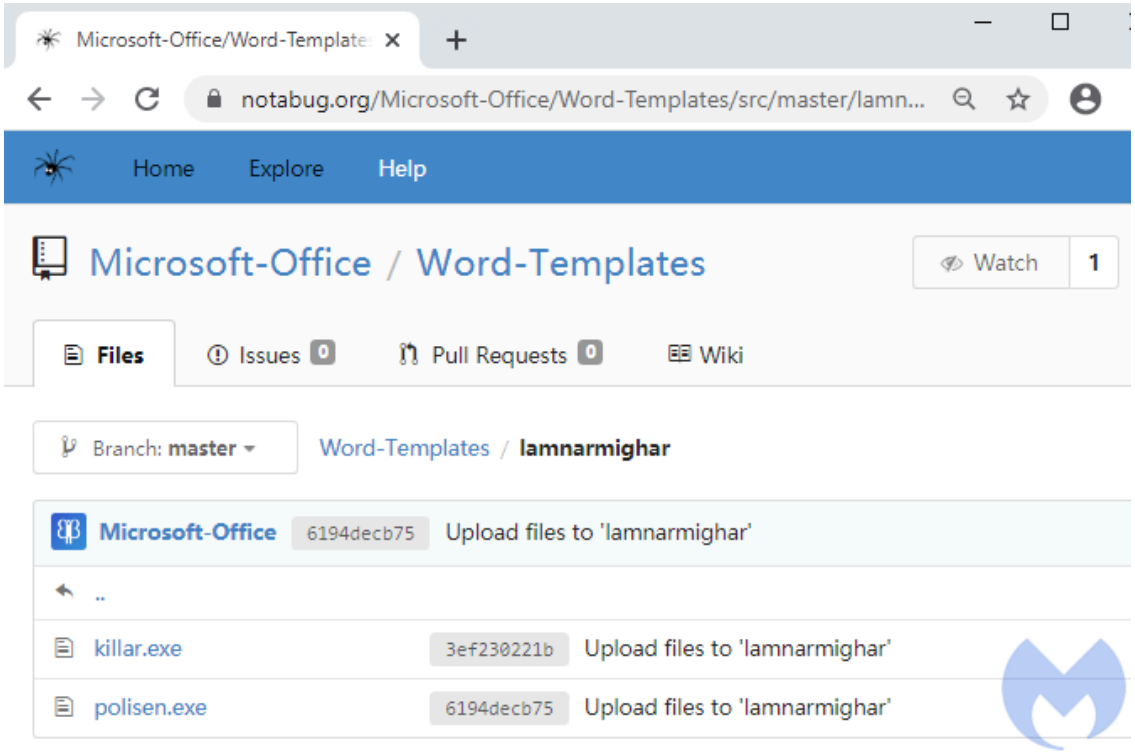
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



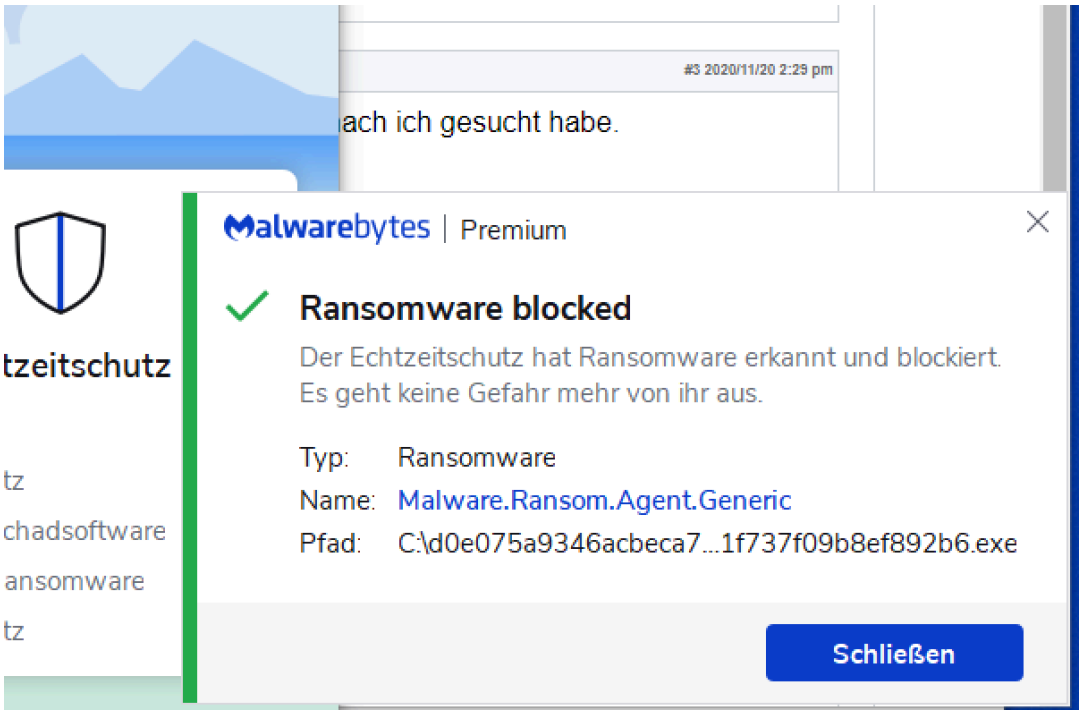
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

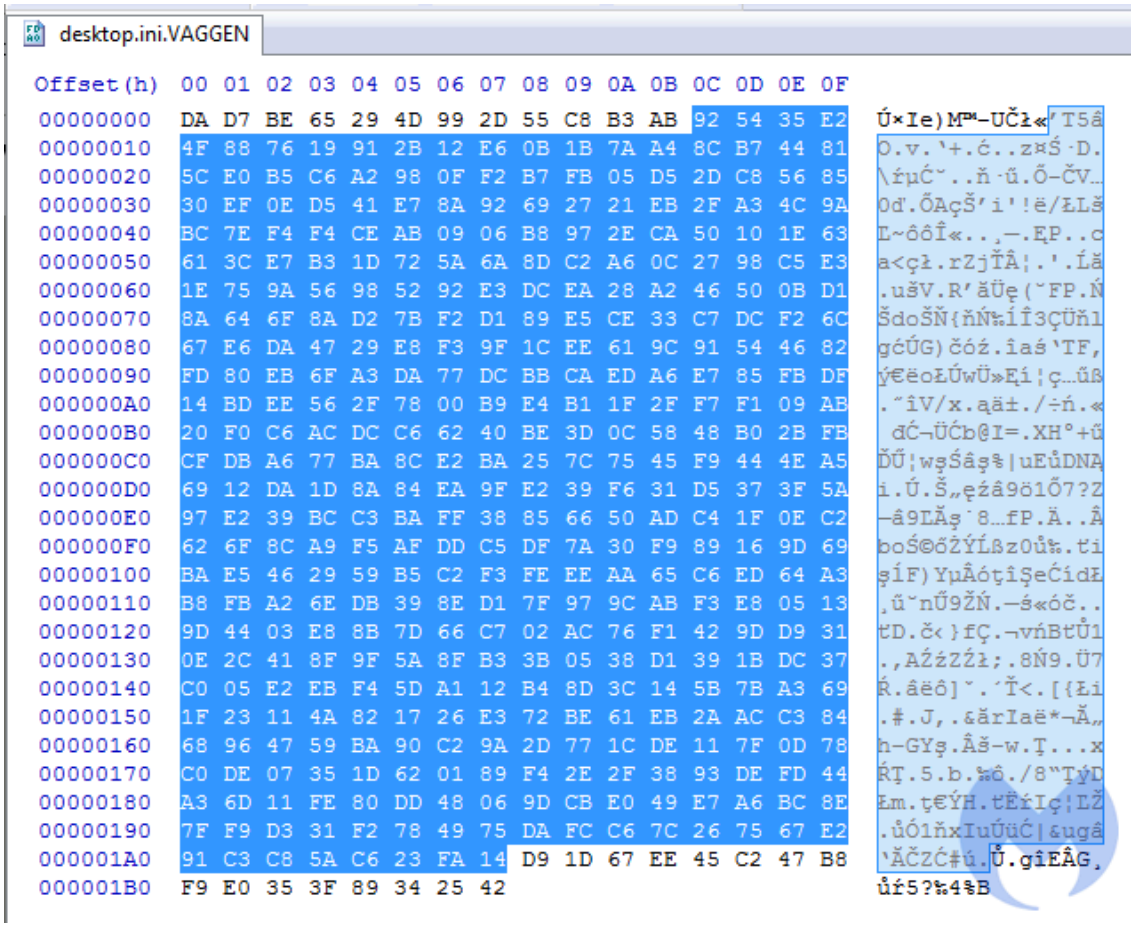
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

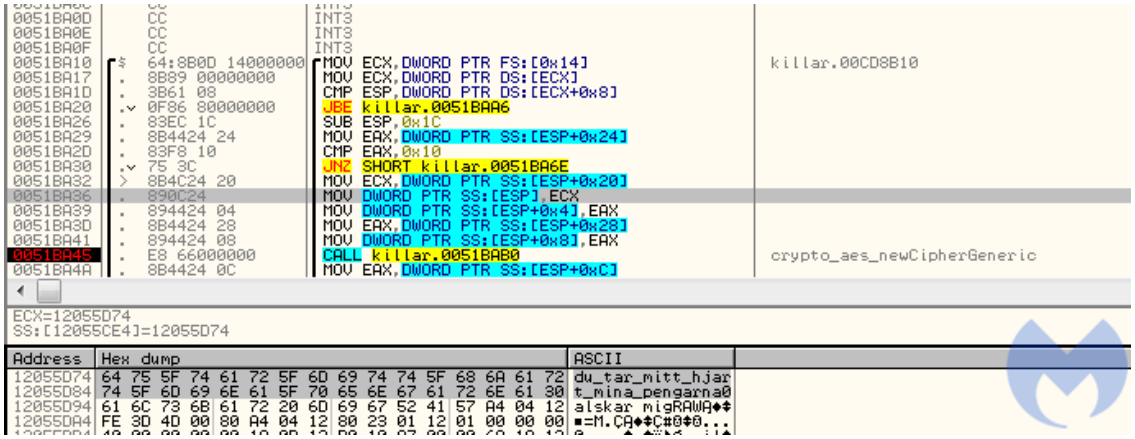


The content of the output file (with .VAGGEN extension) contains:

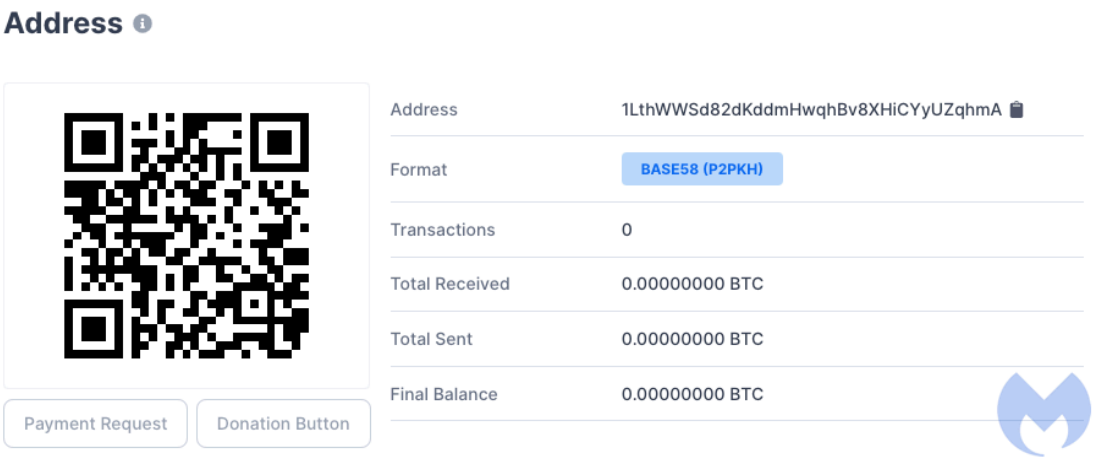
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

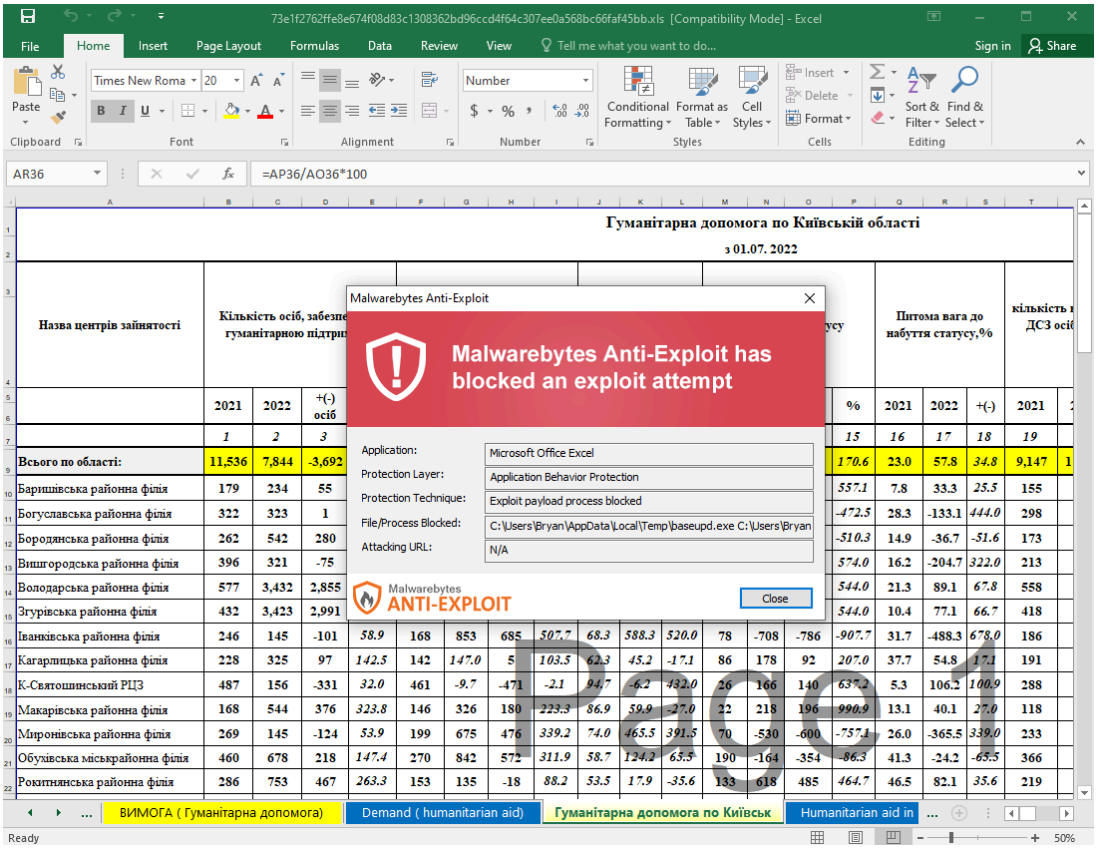
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

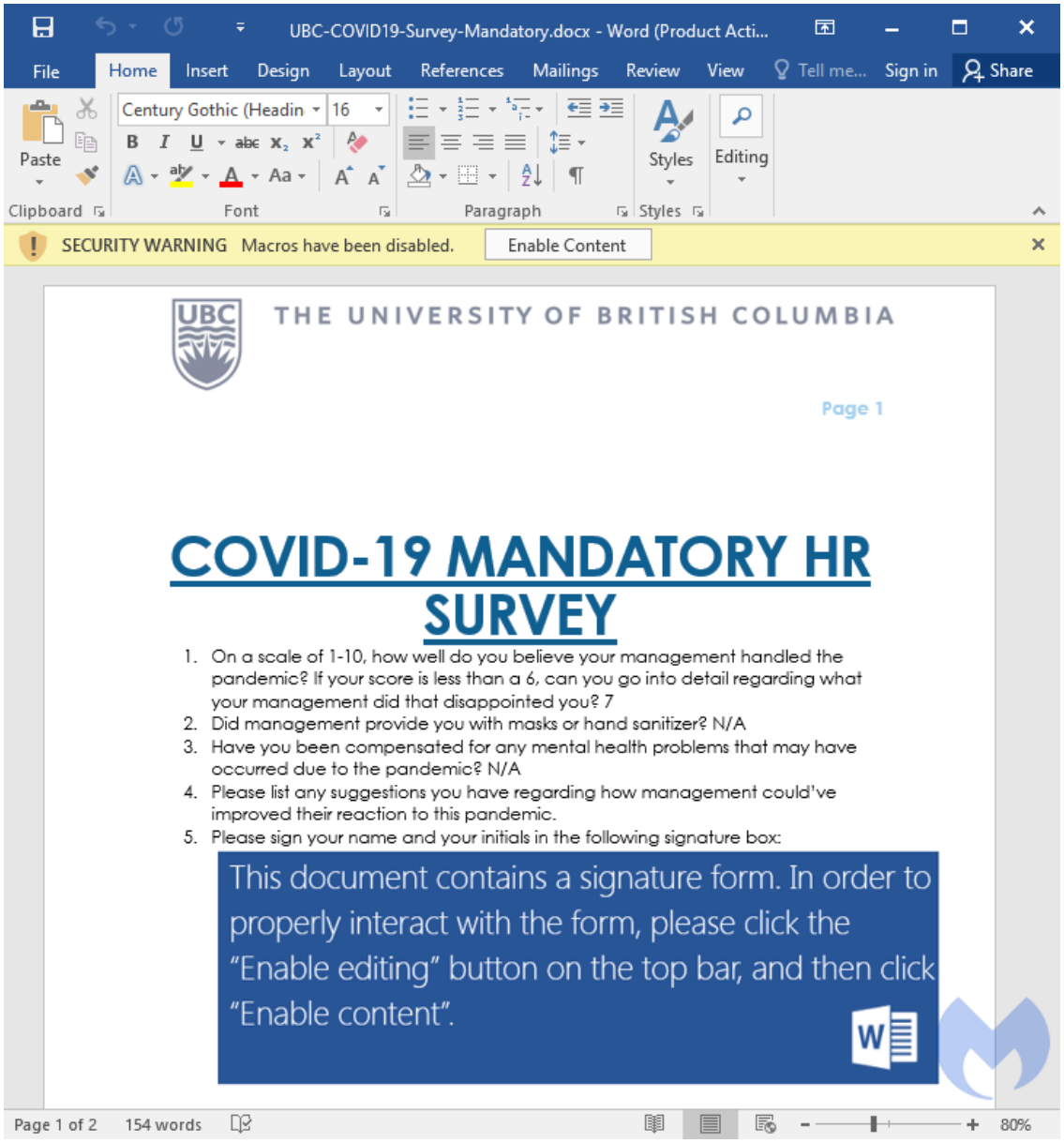
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

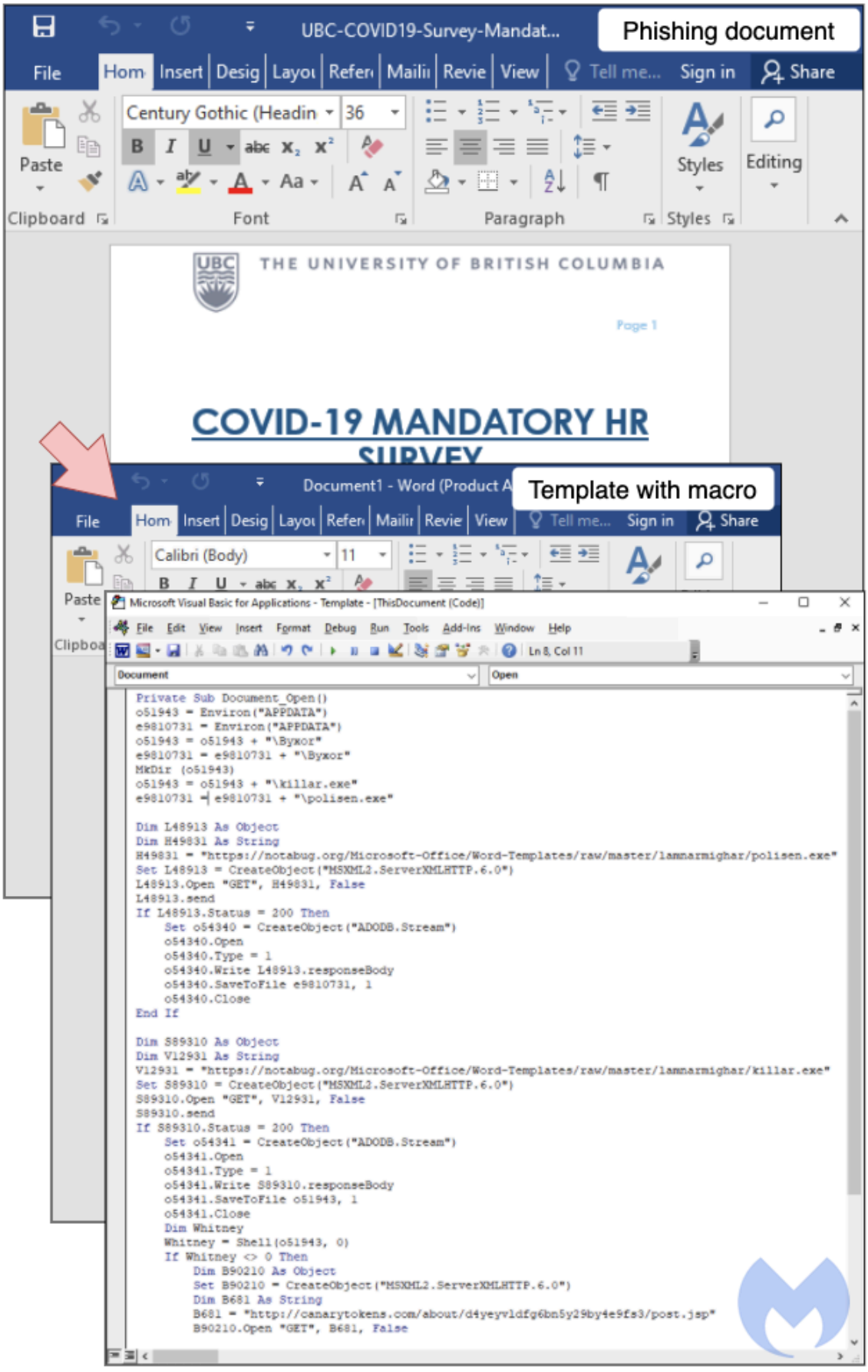
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

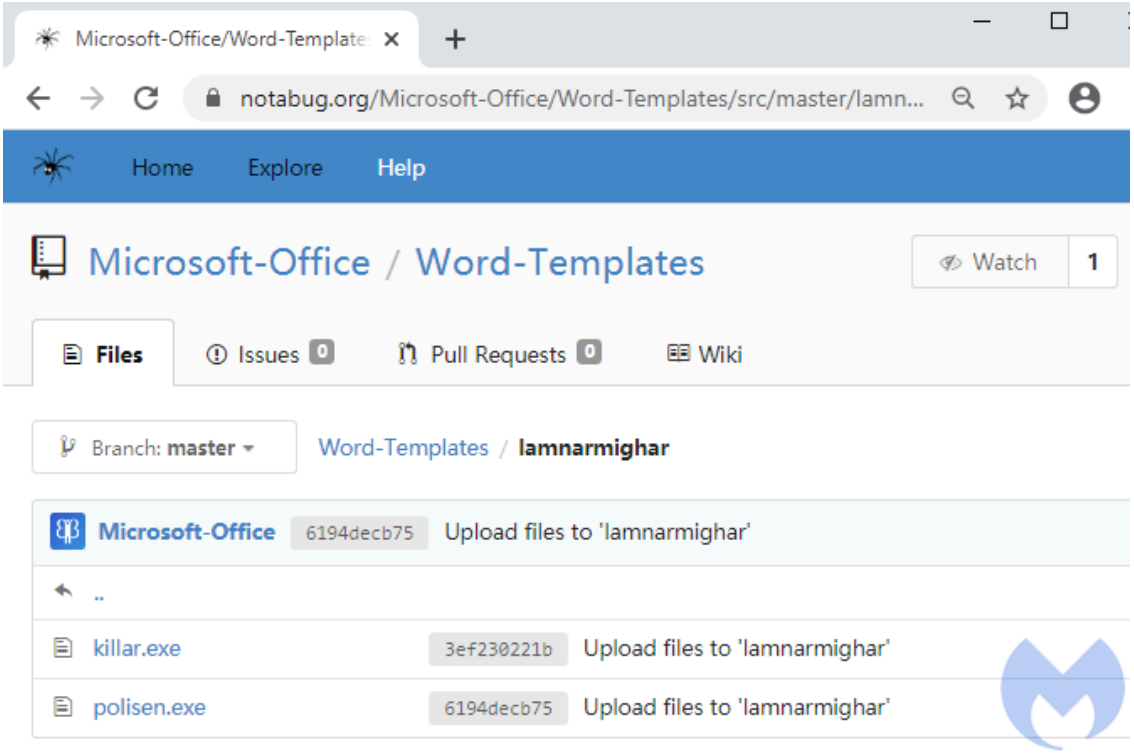
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



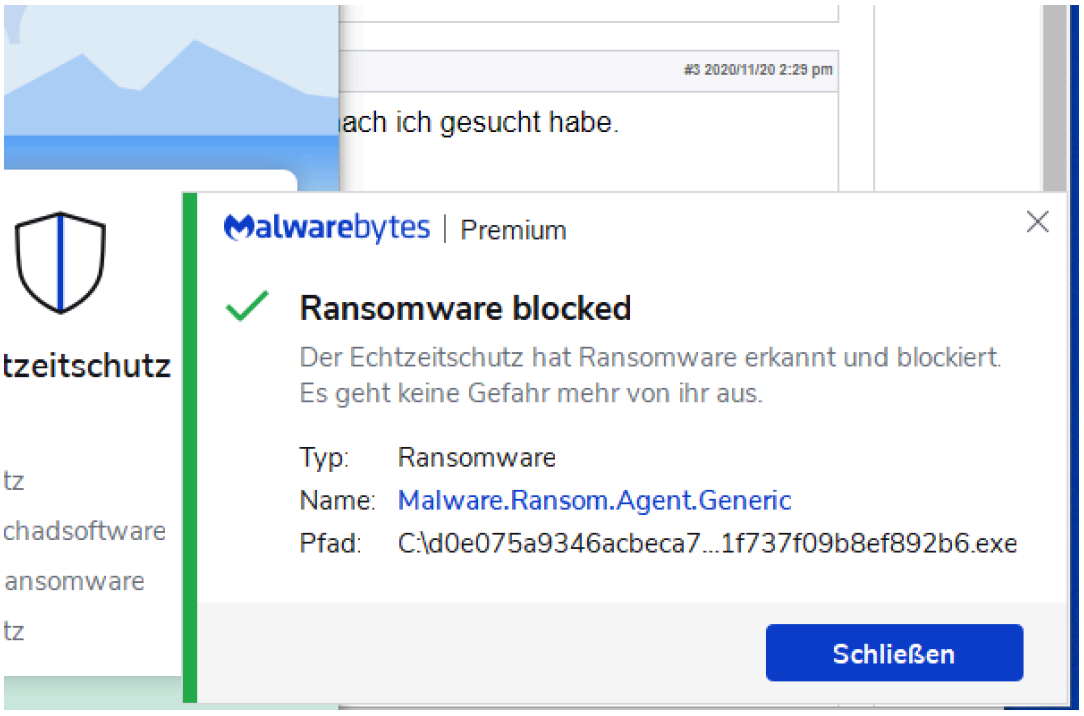
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

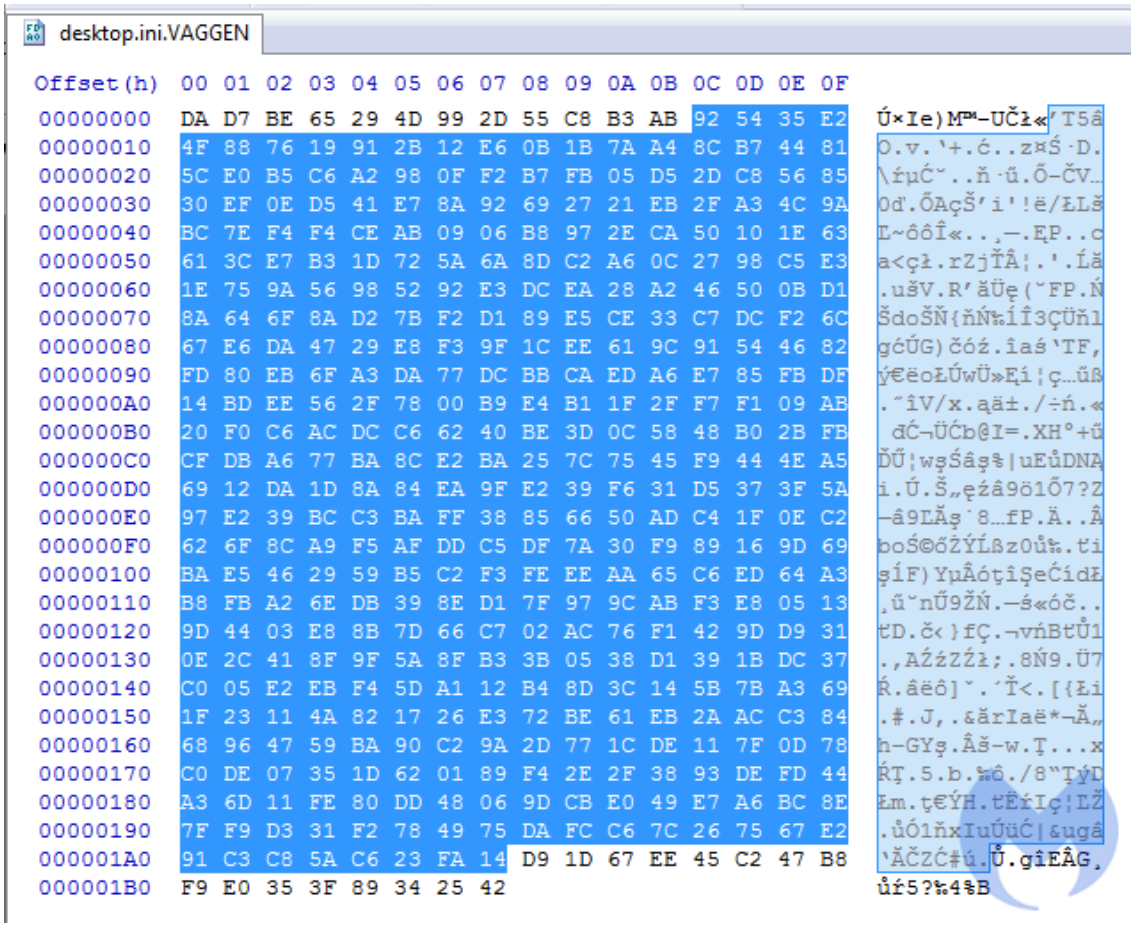
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



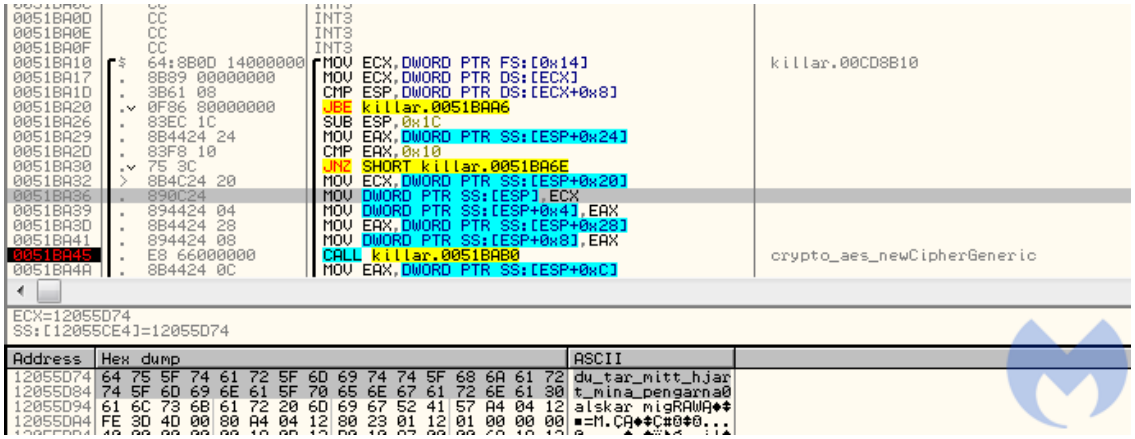
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

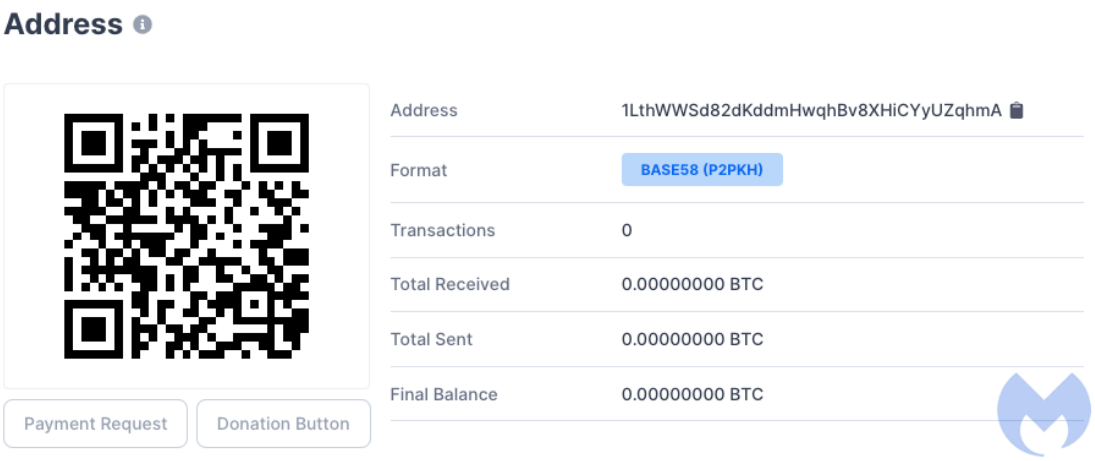
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

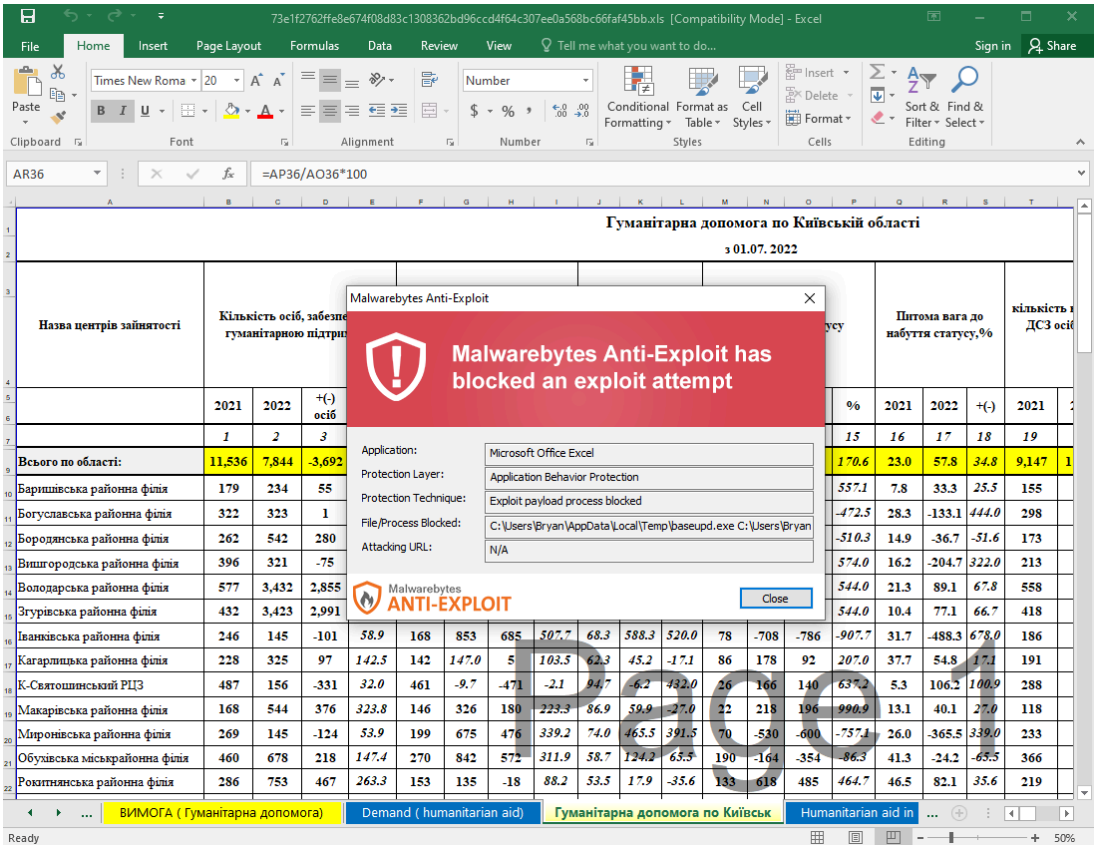
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

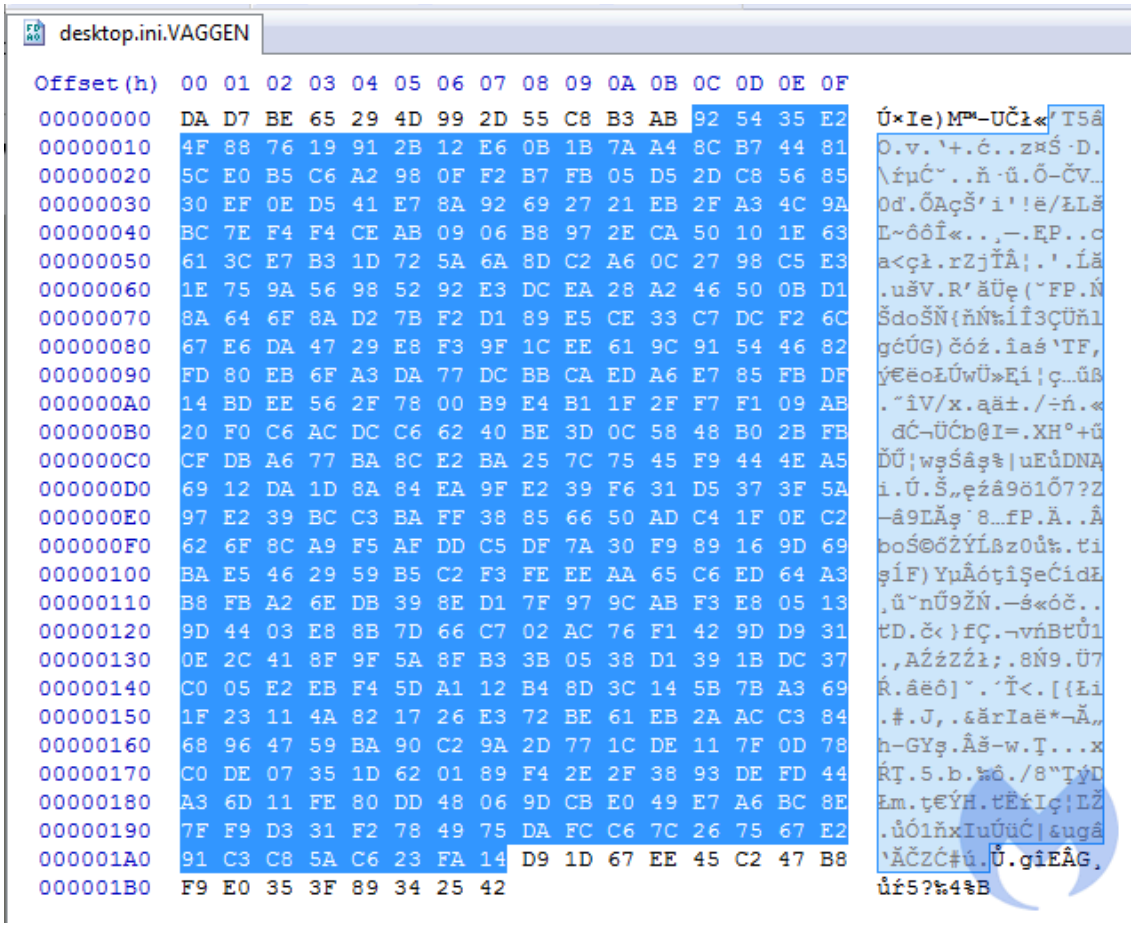
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



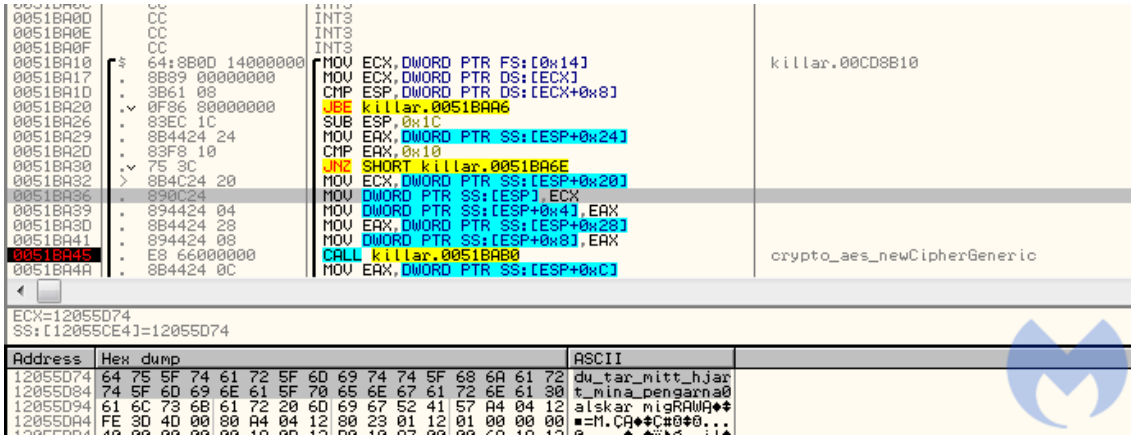
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

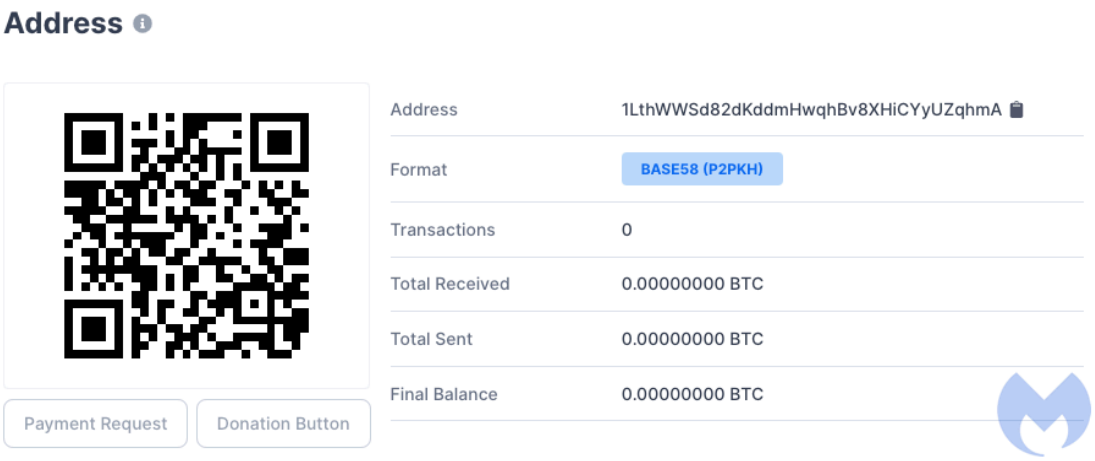
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

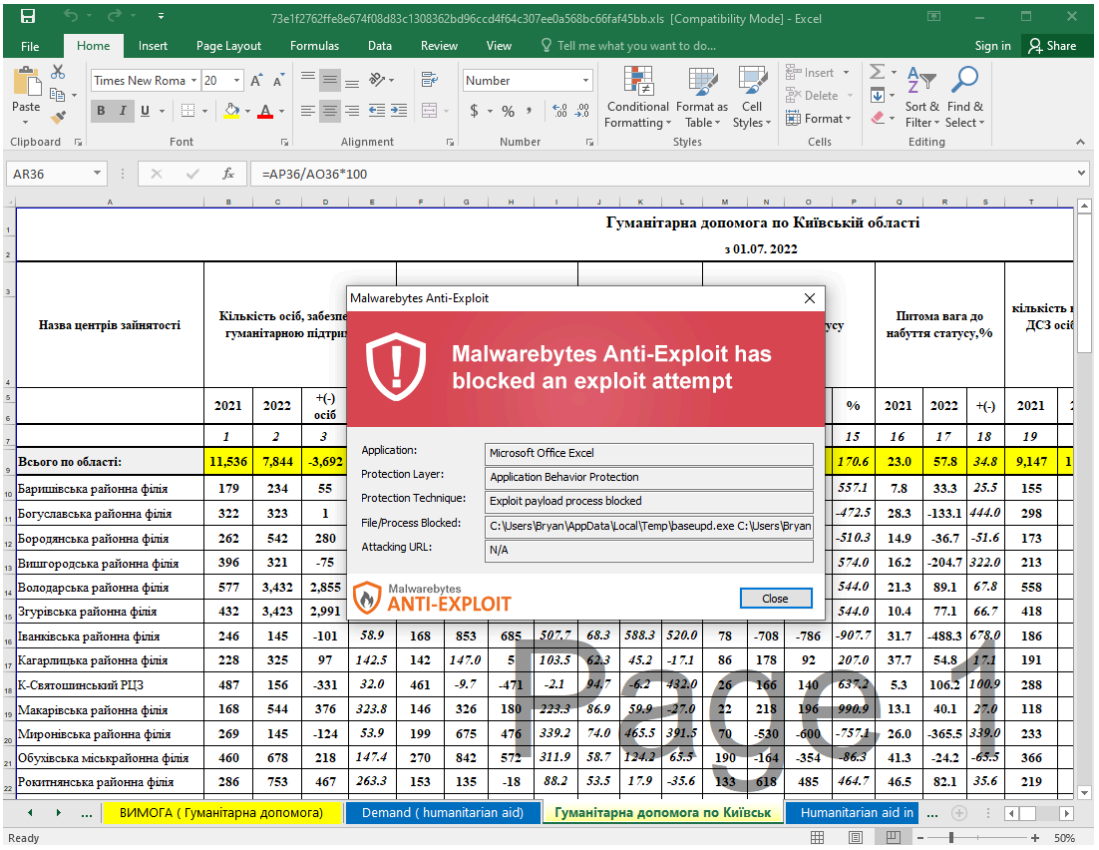
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

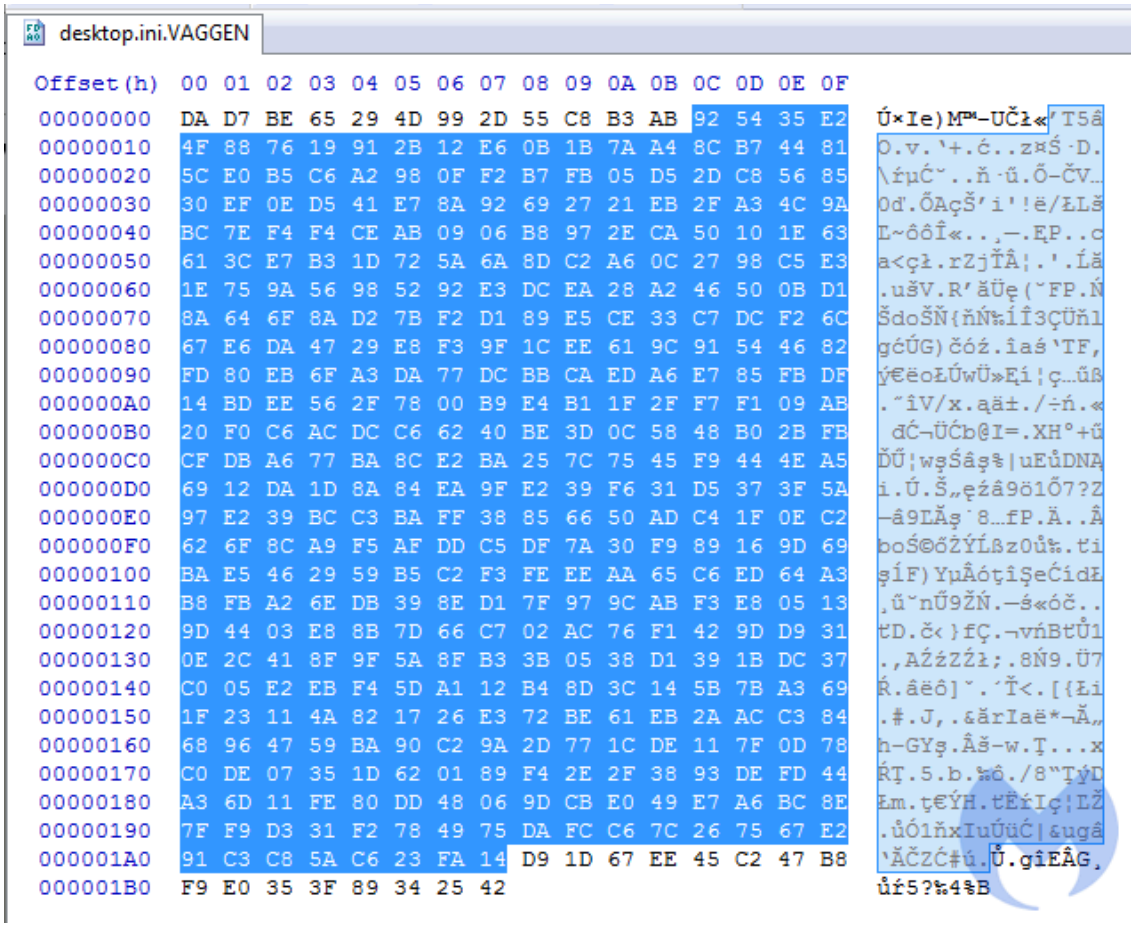
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

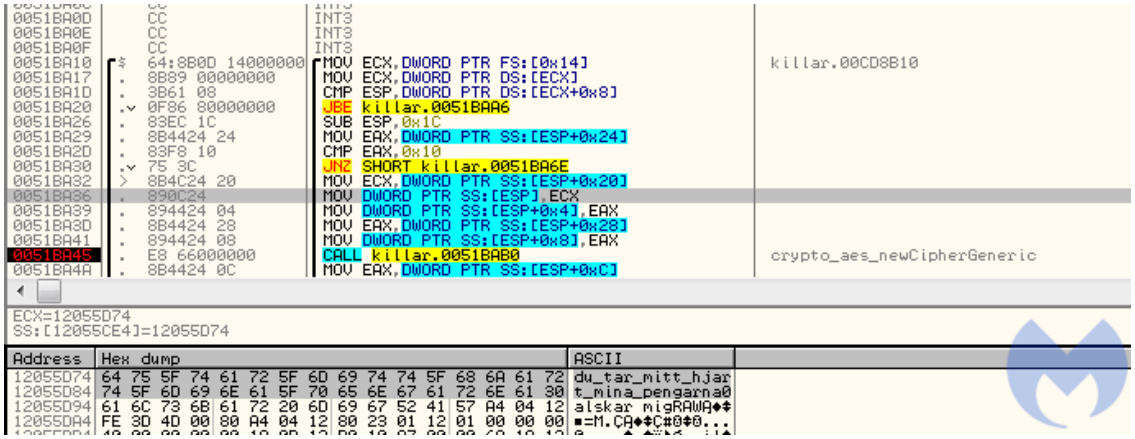


The content of the output file (with .VAGGEN extension) contains:

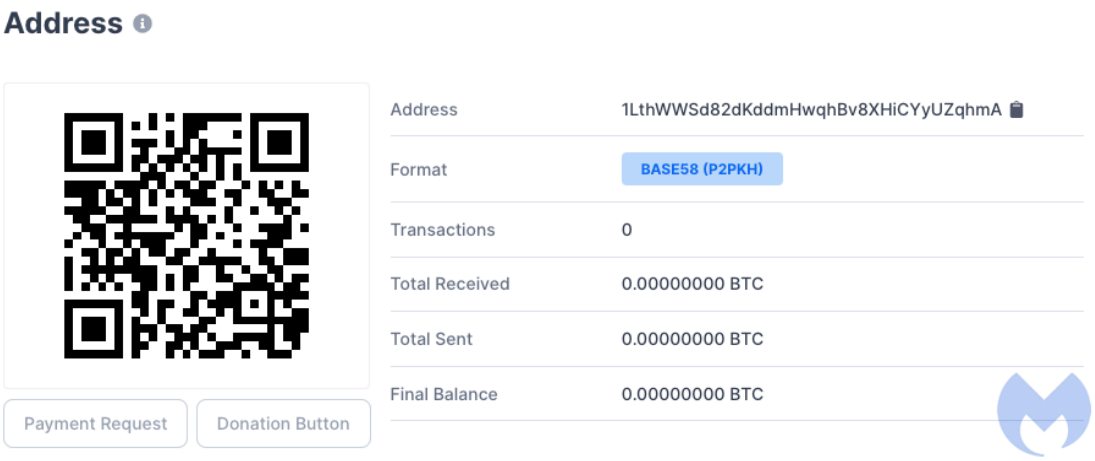
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

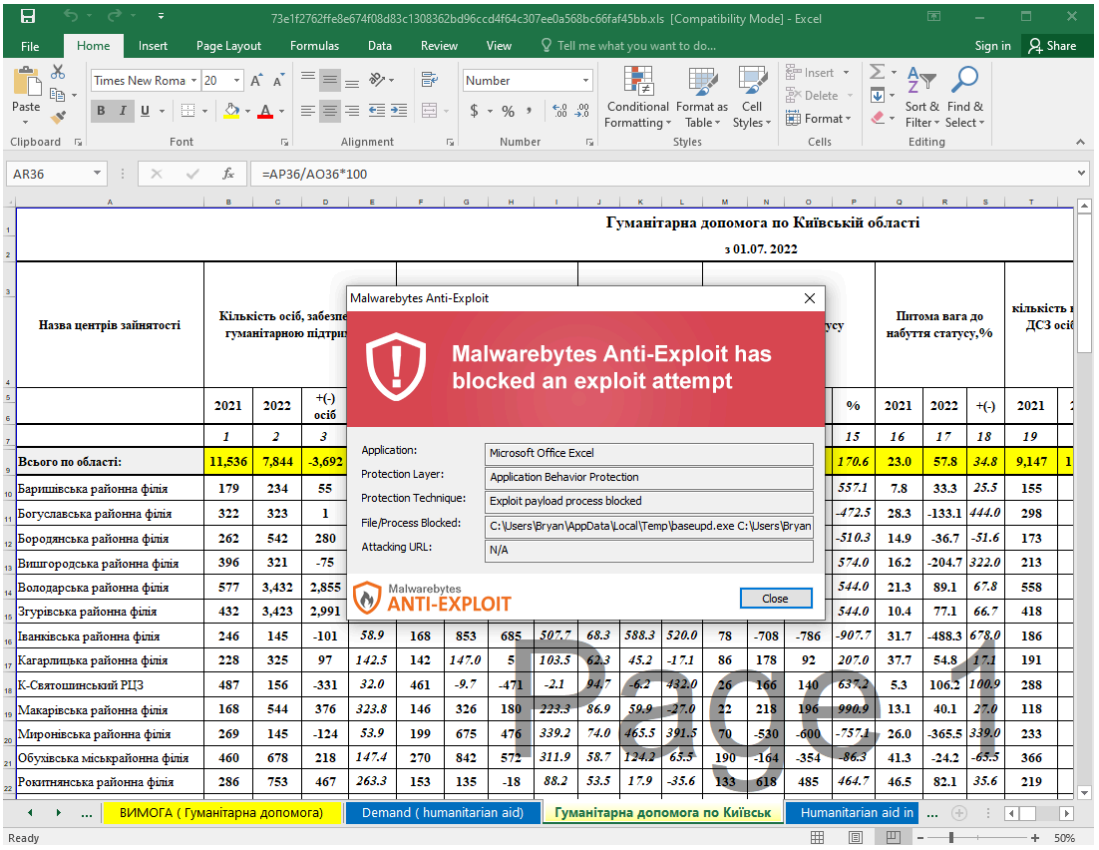
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

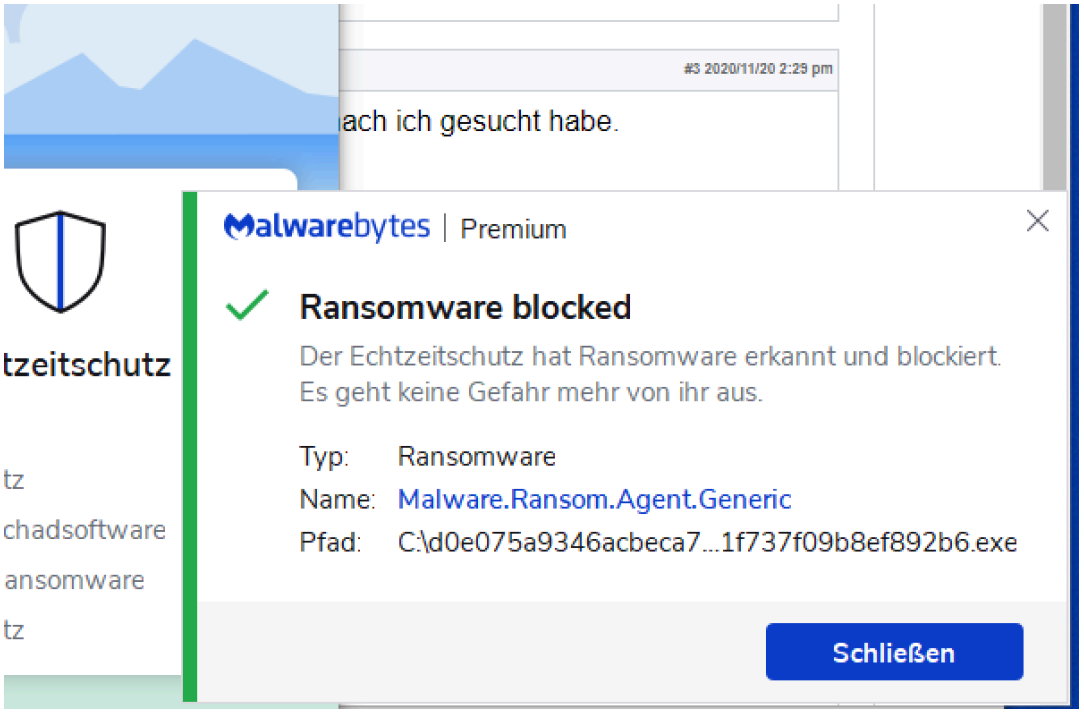
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

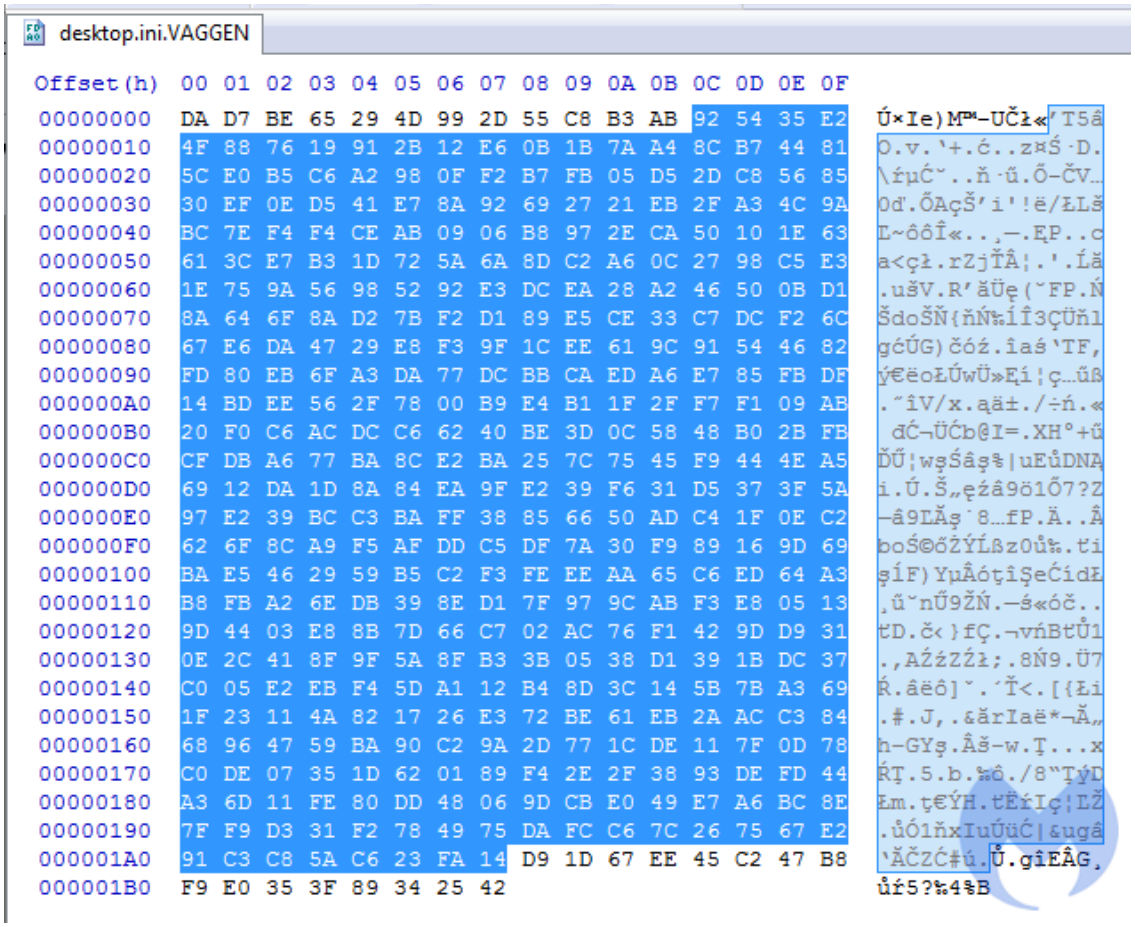
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

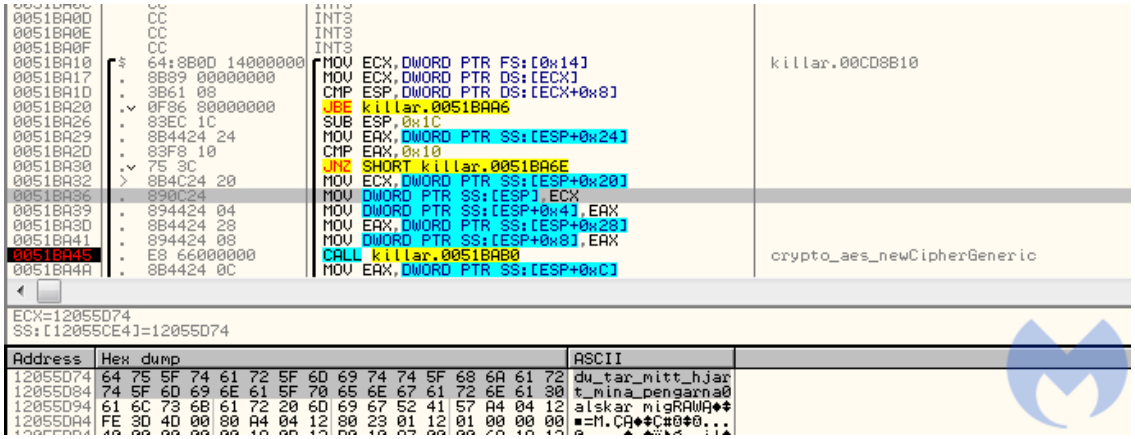


The content of the output file (with .VAGGEN extension) contains:

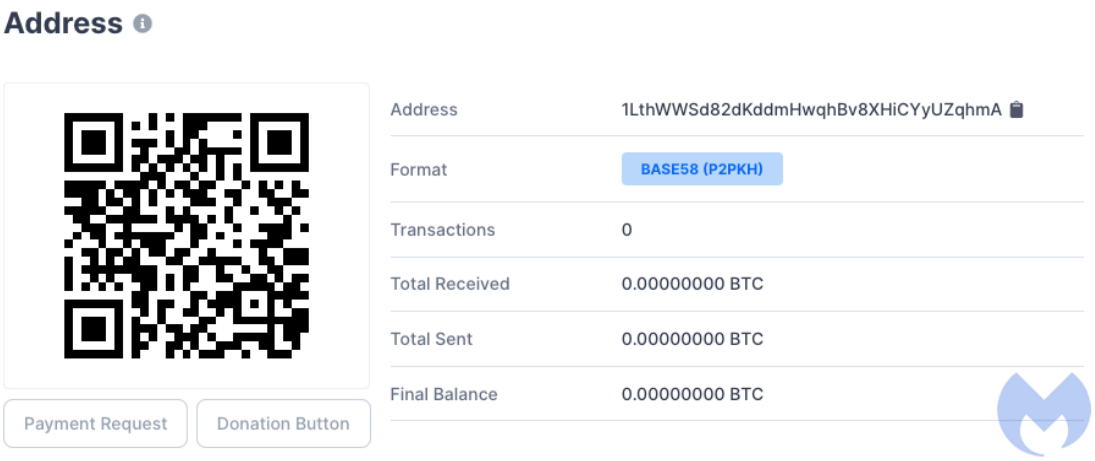
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

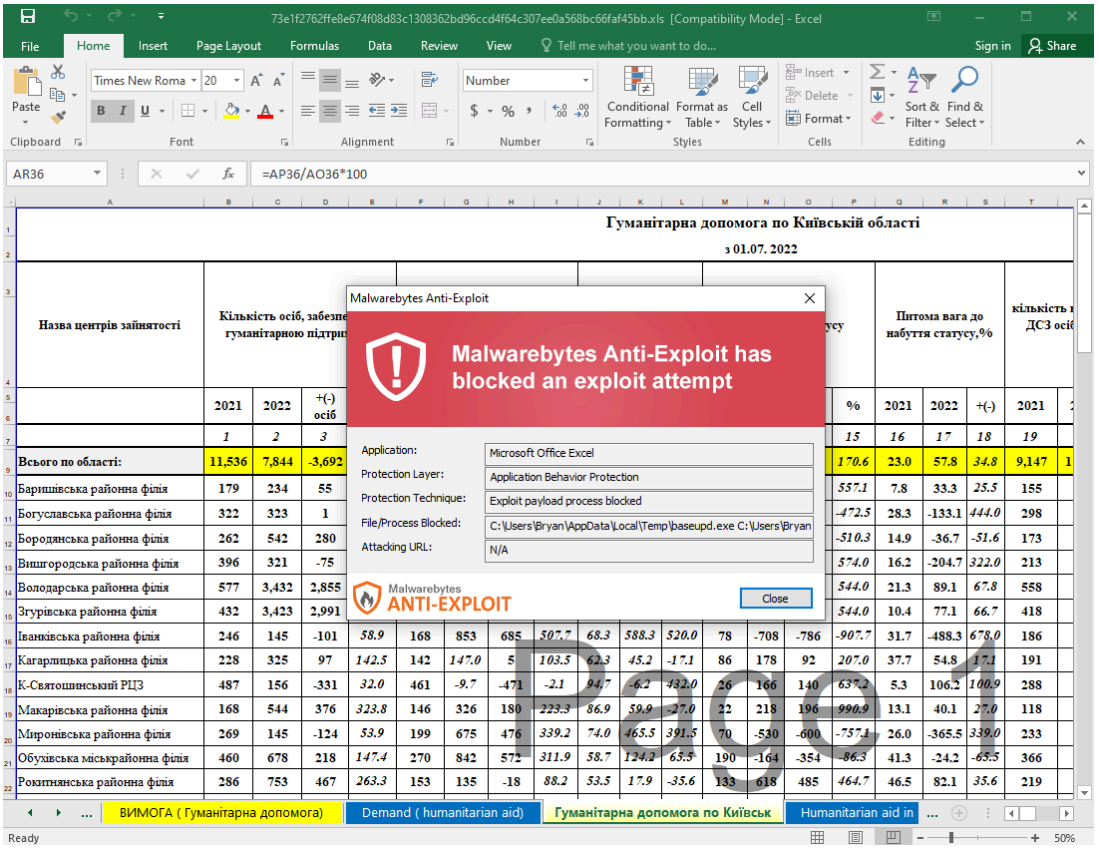
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

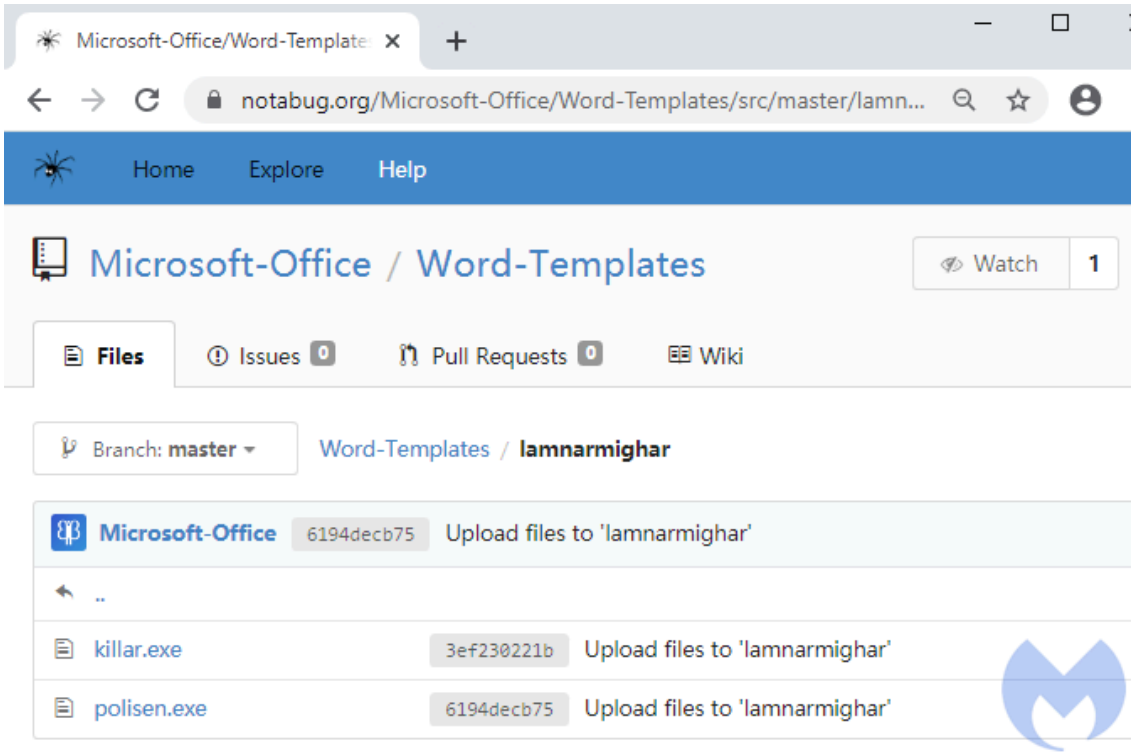
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



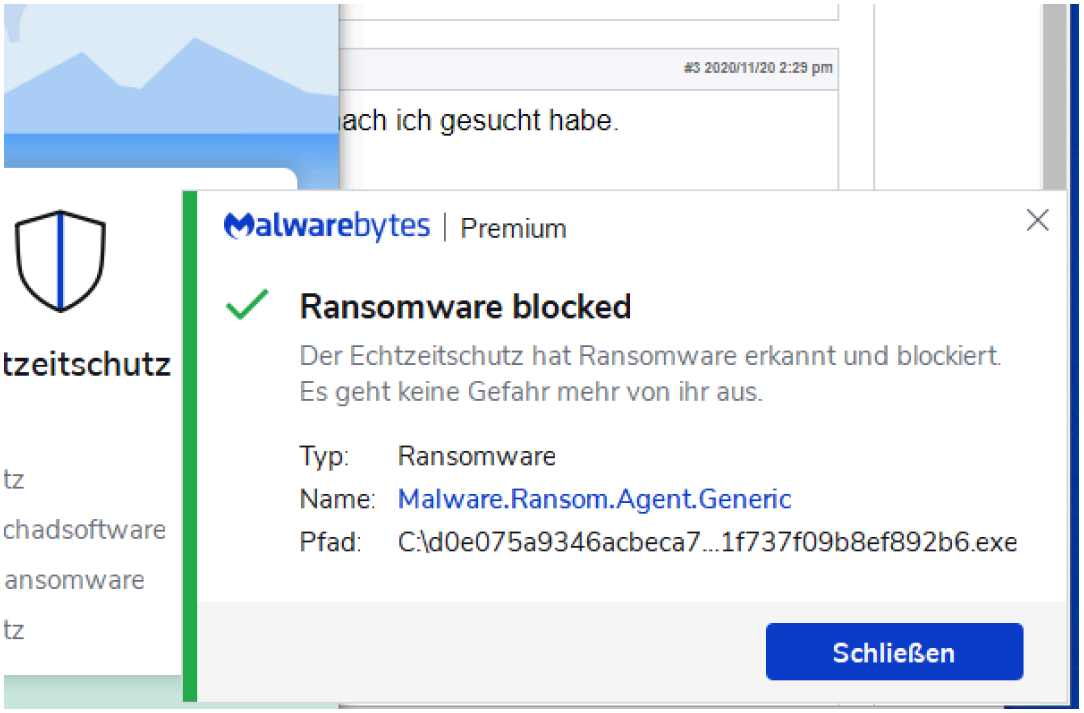
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

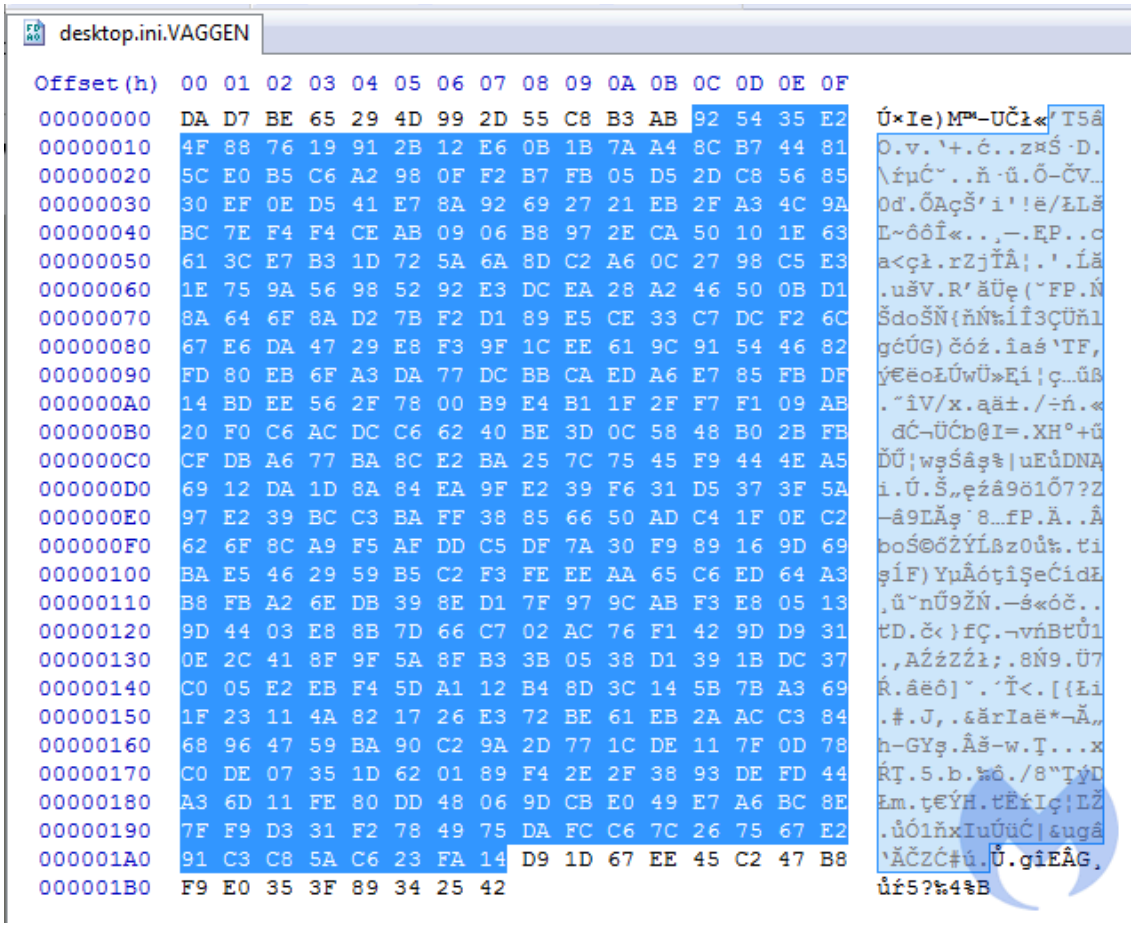
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



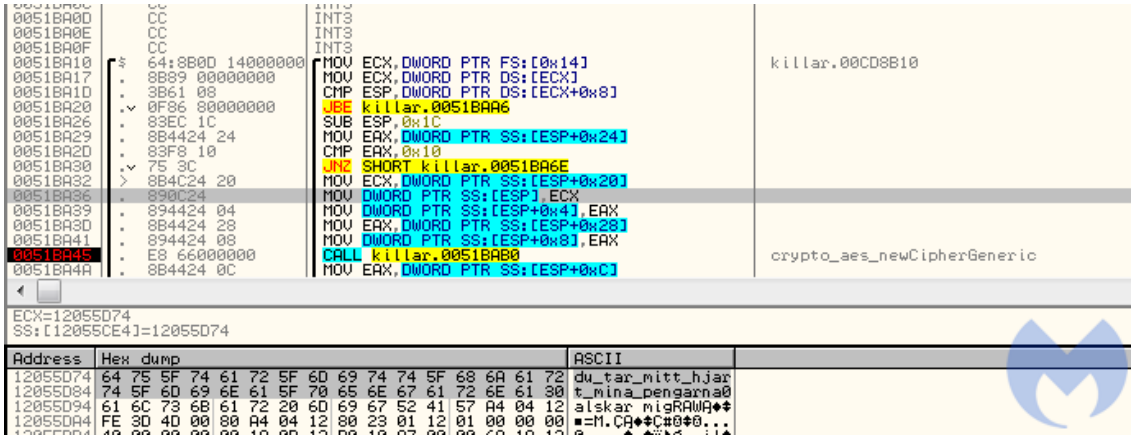
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

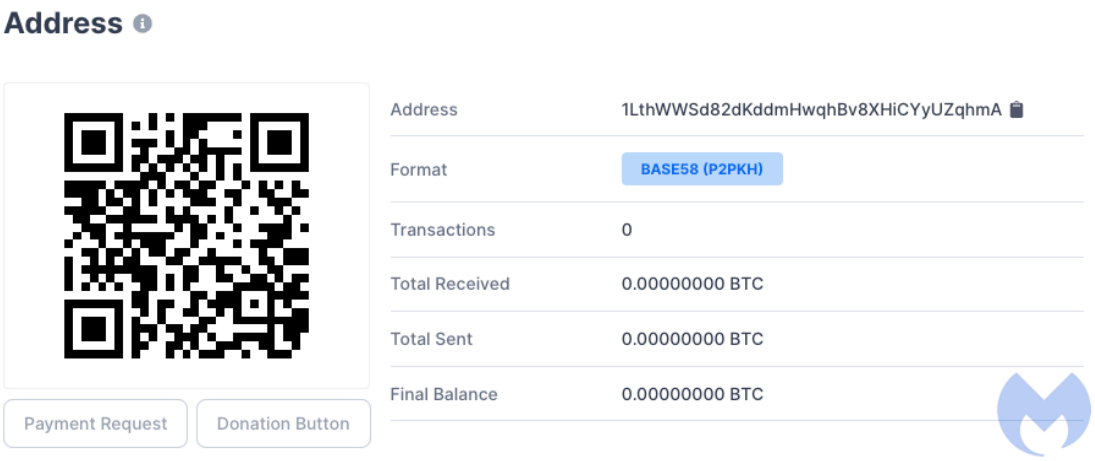
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

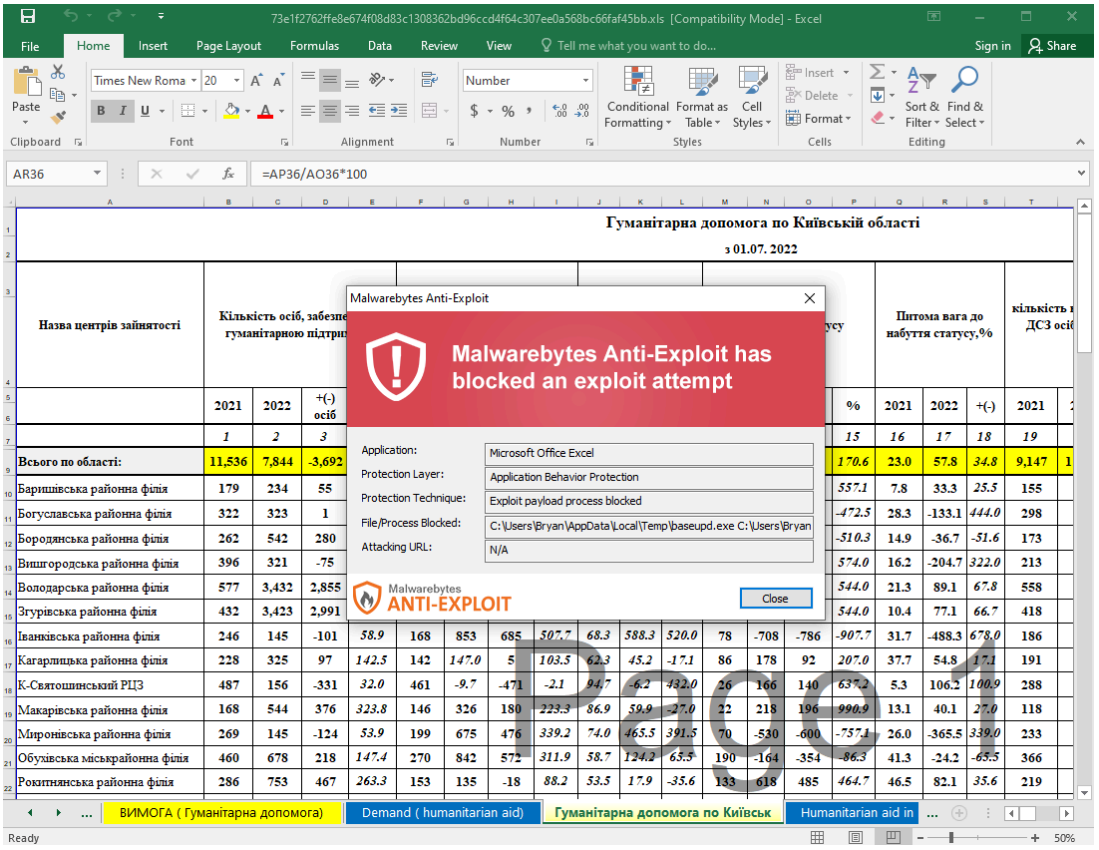
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

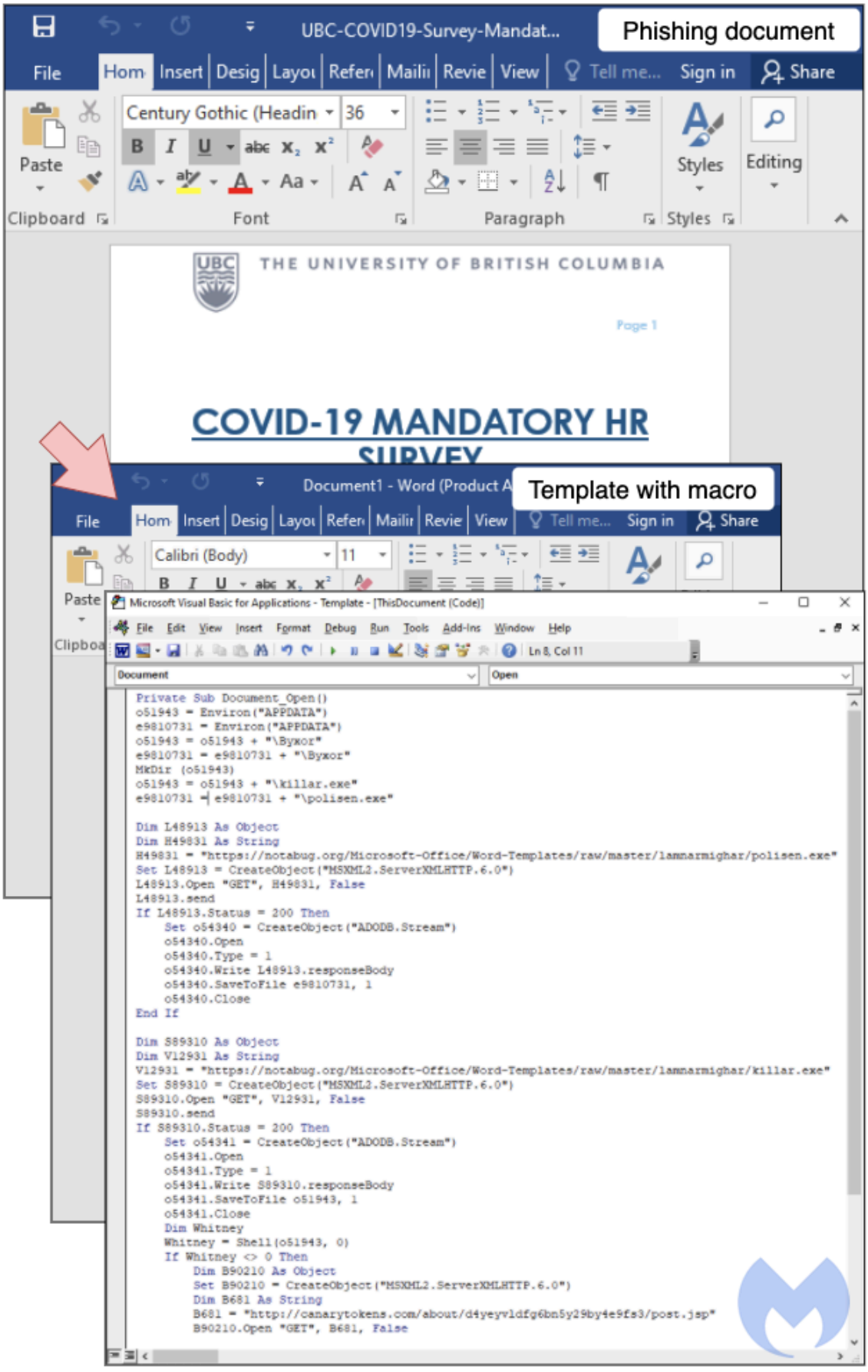
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

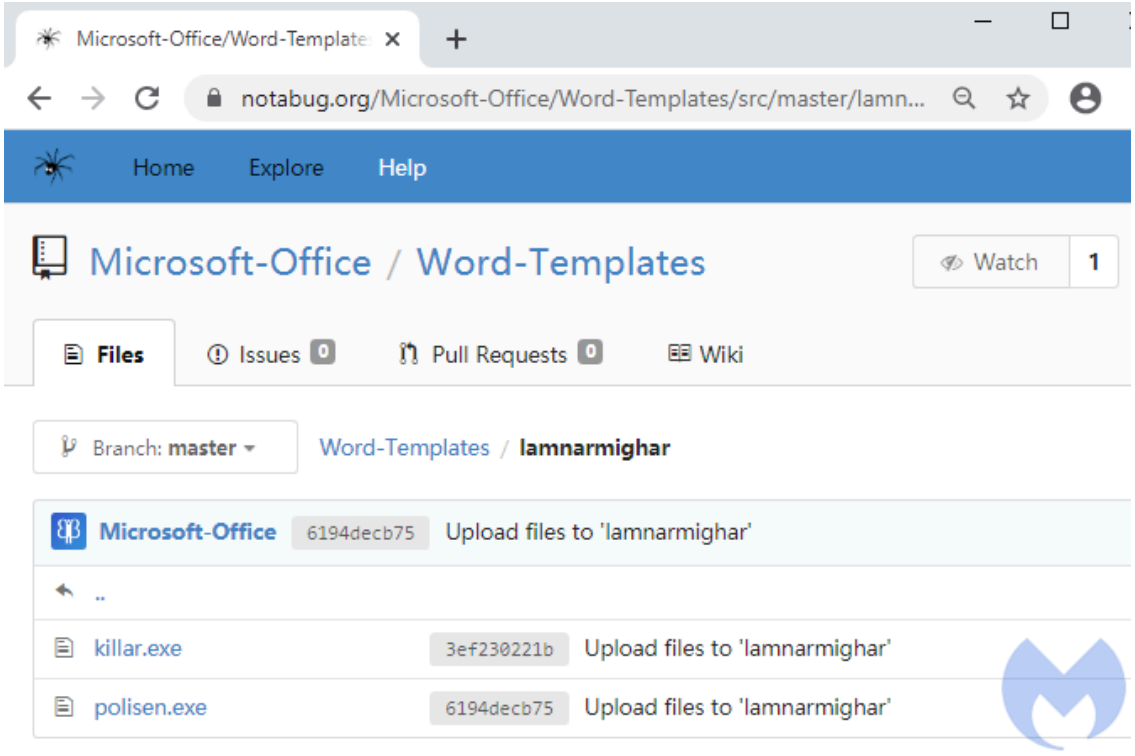
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



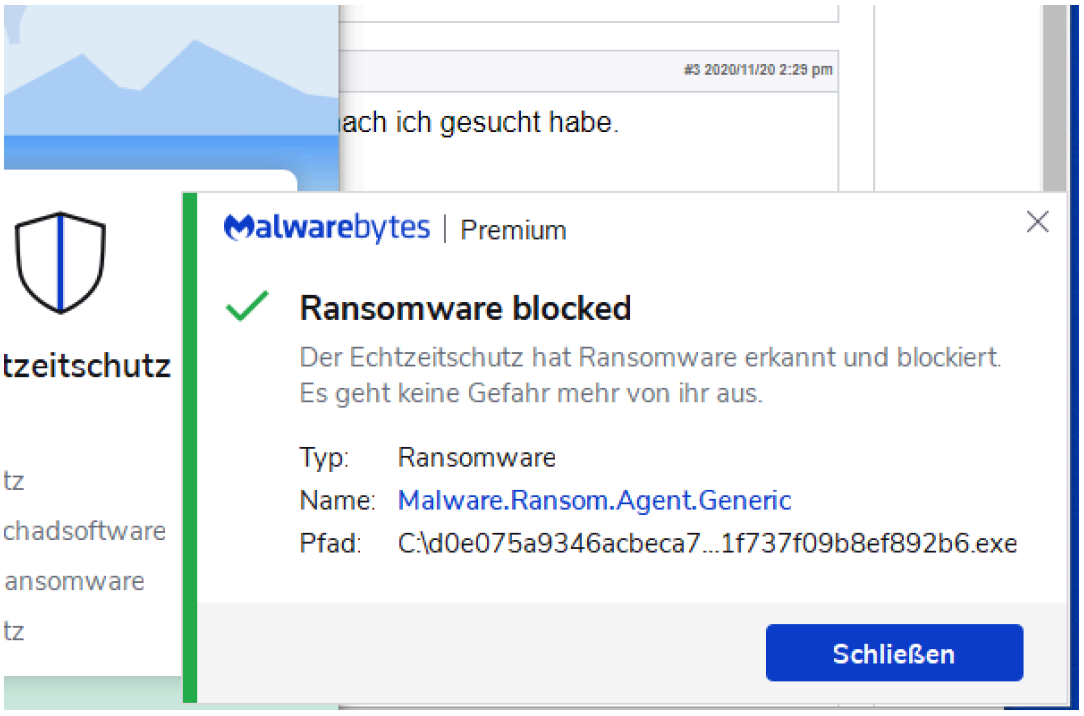
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

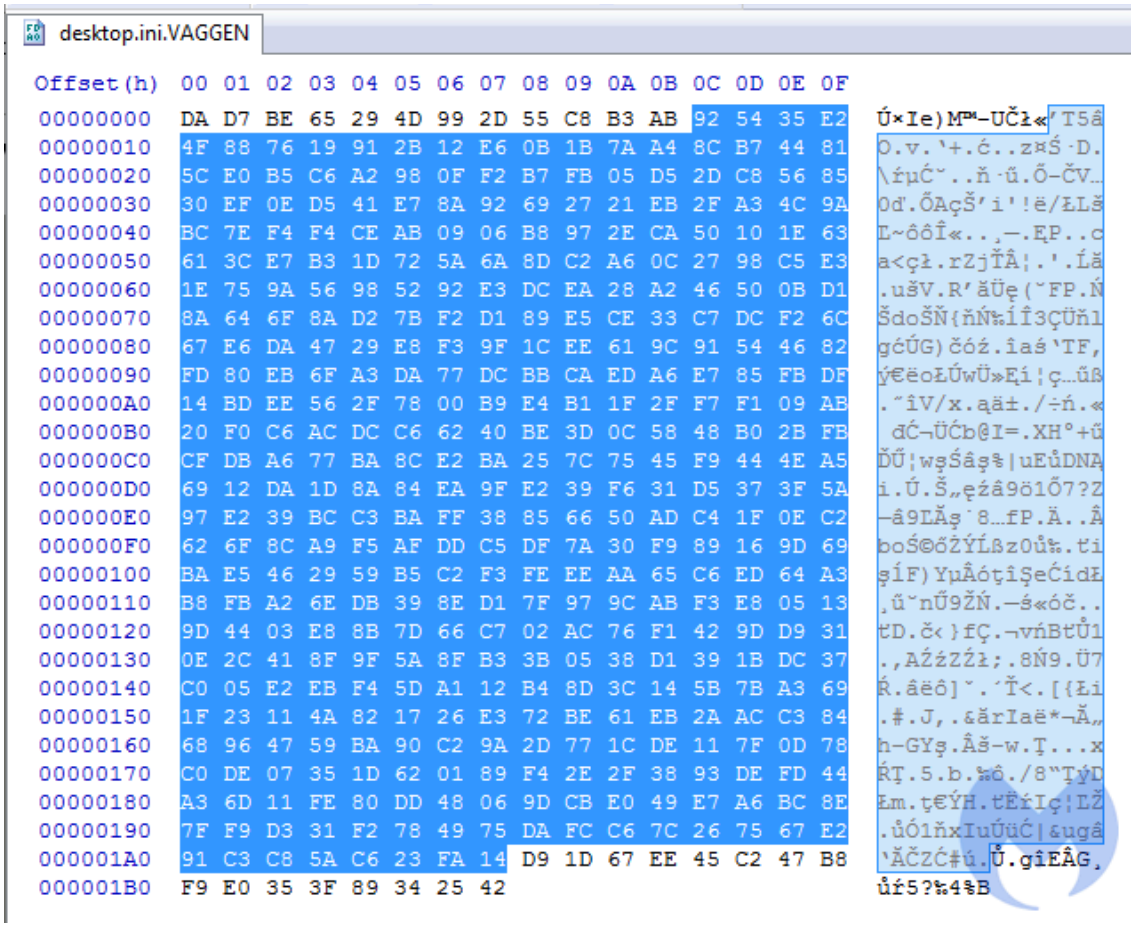
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



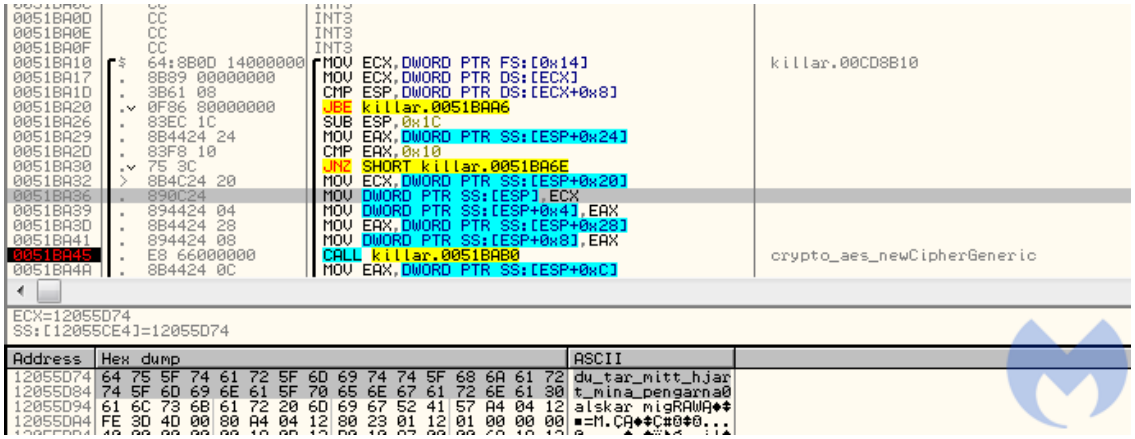
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

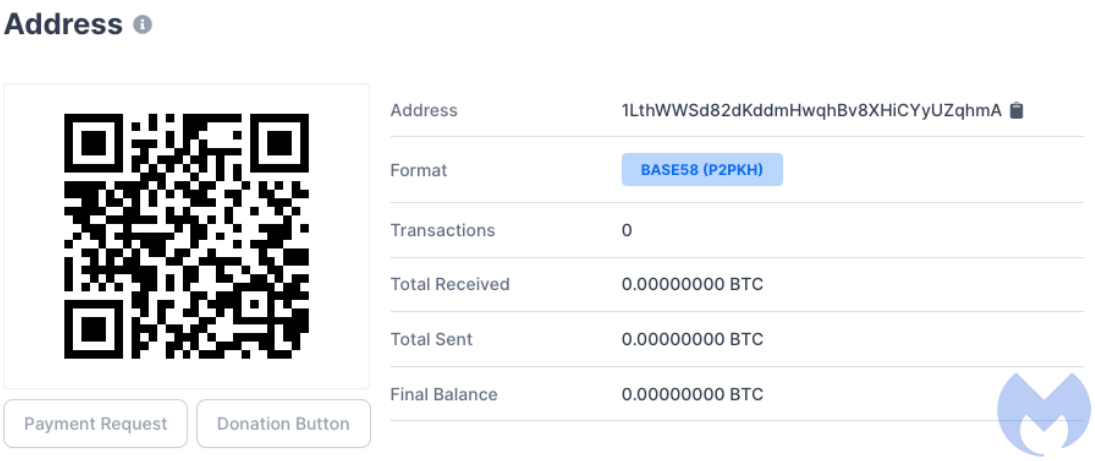
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

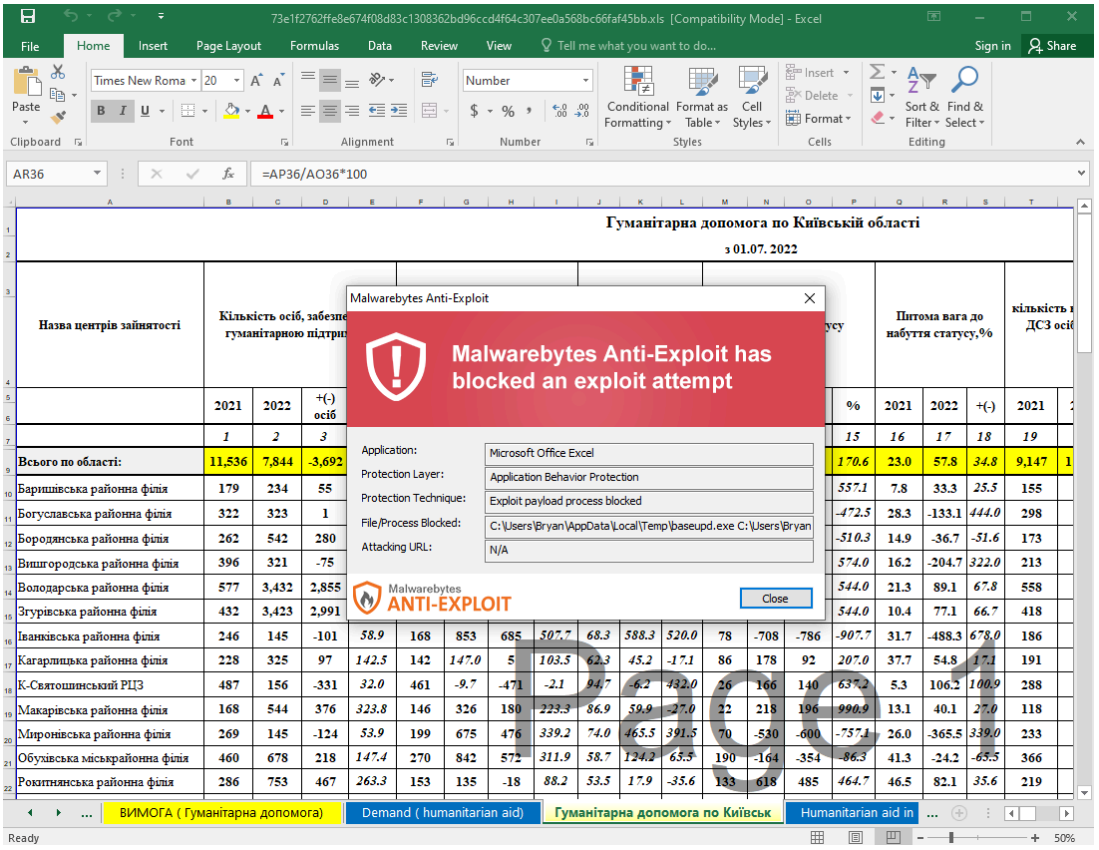
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

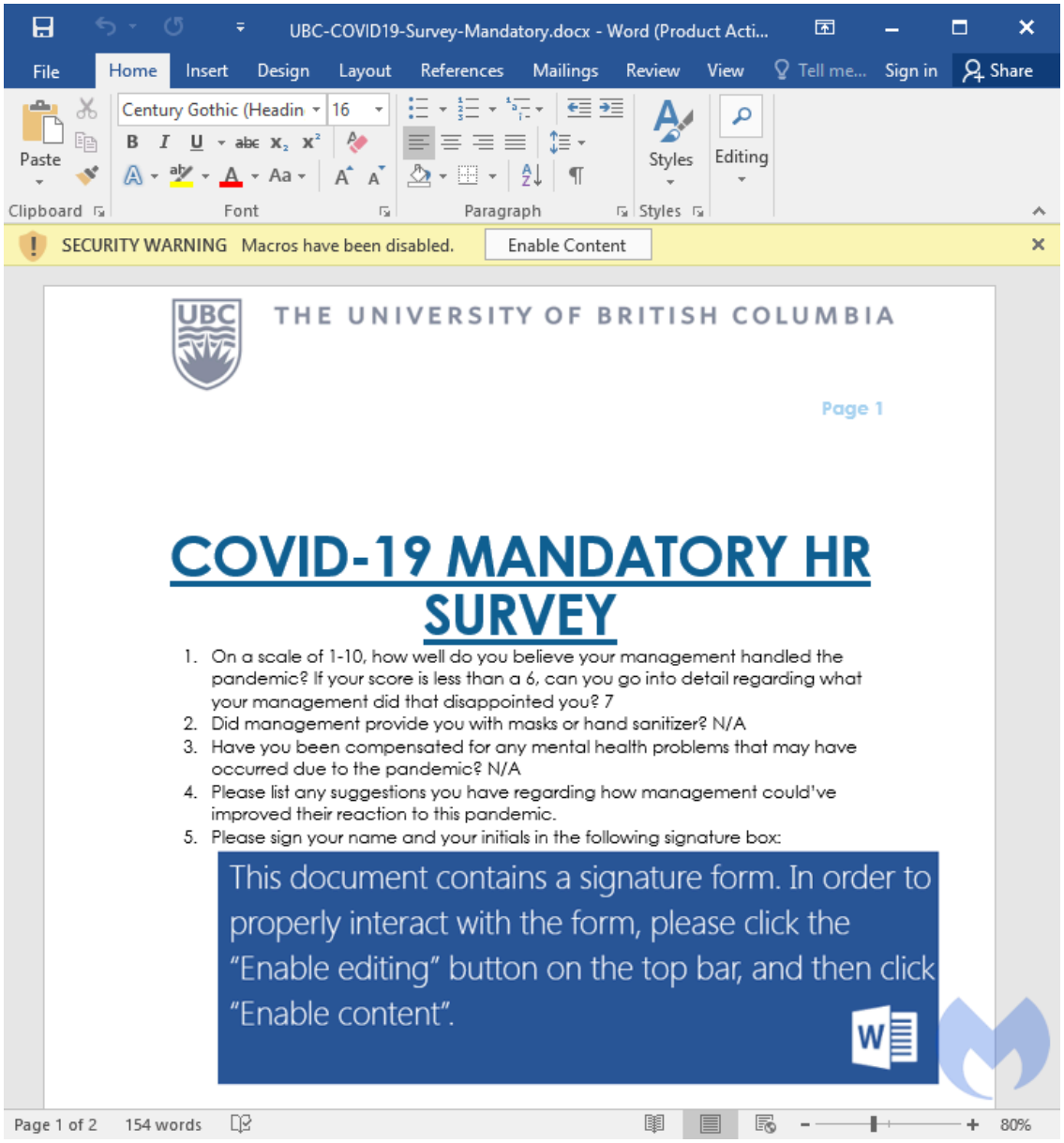
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

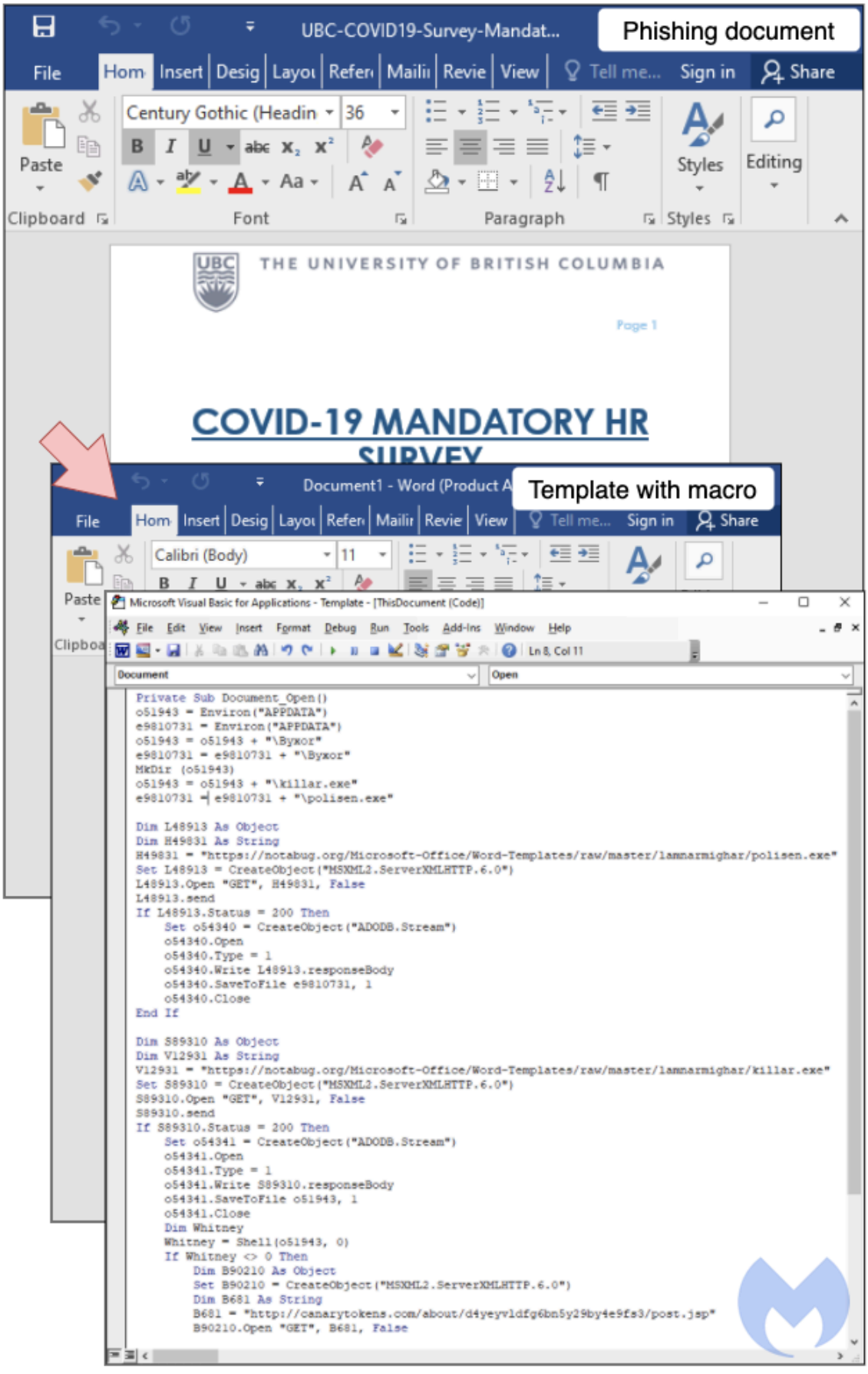
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

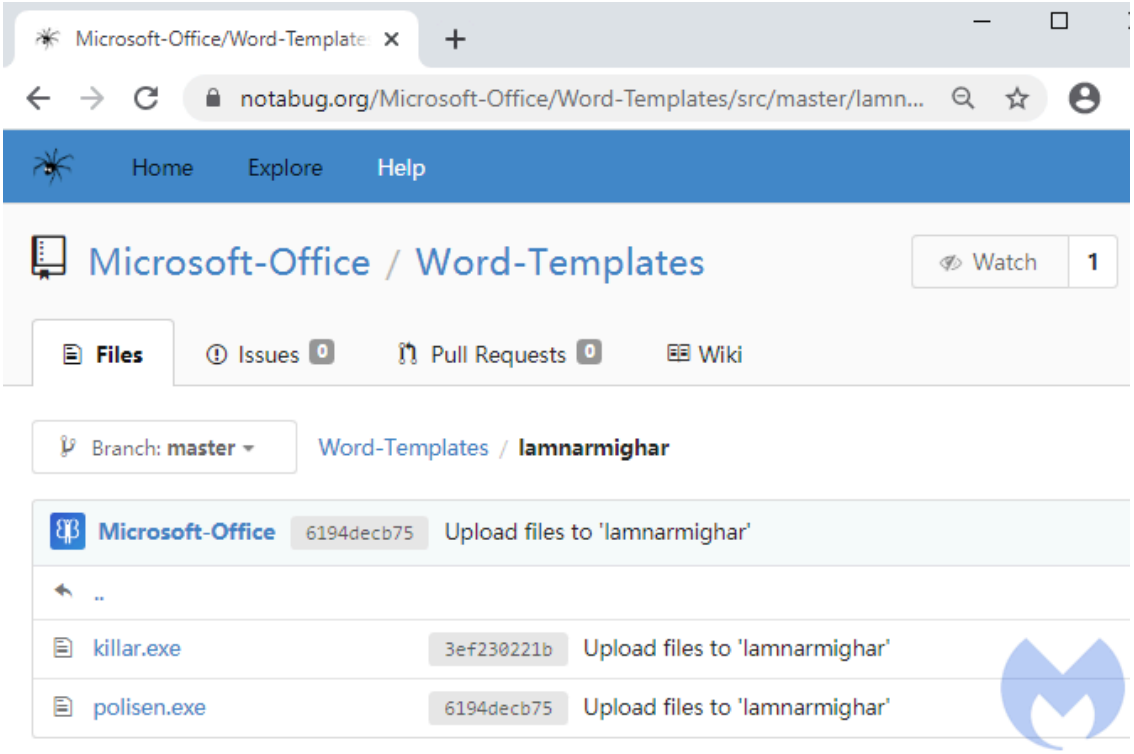
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



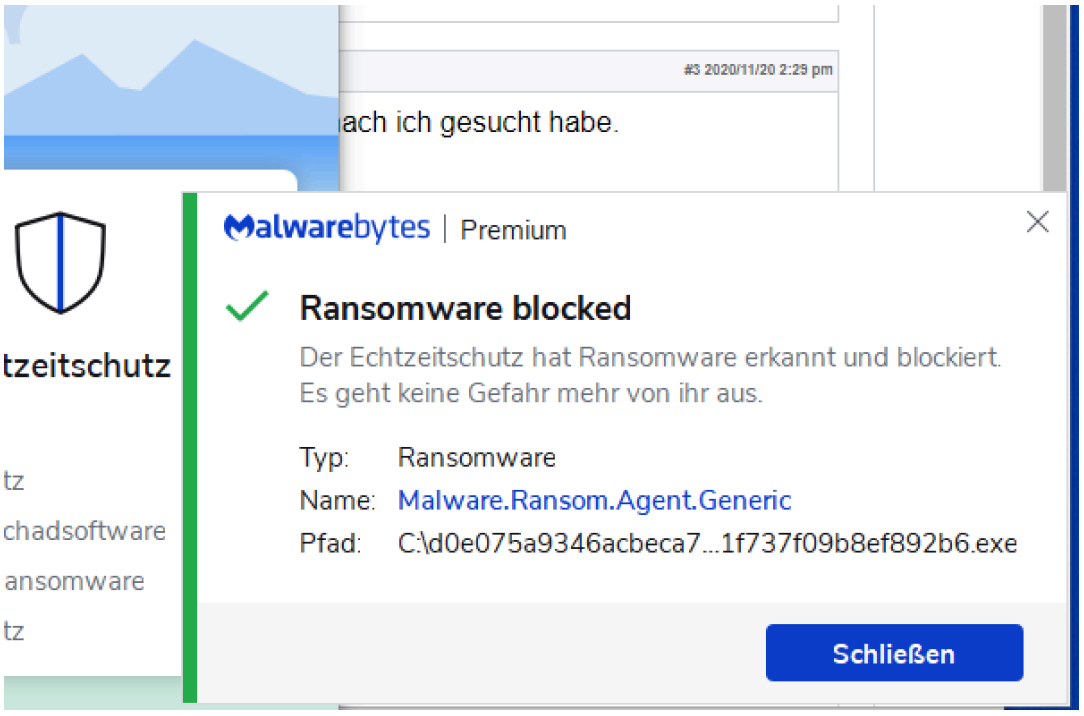
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

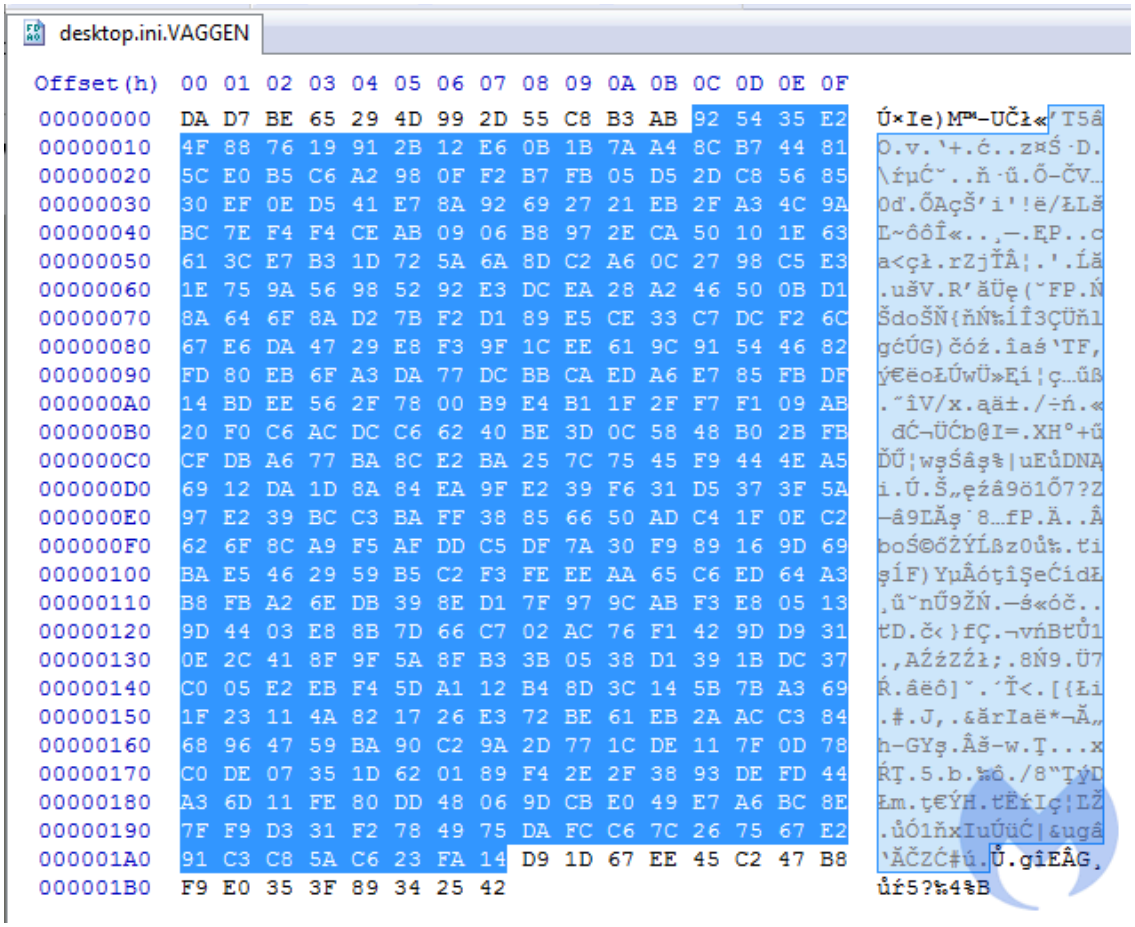
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

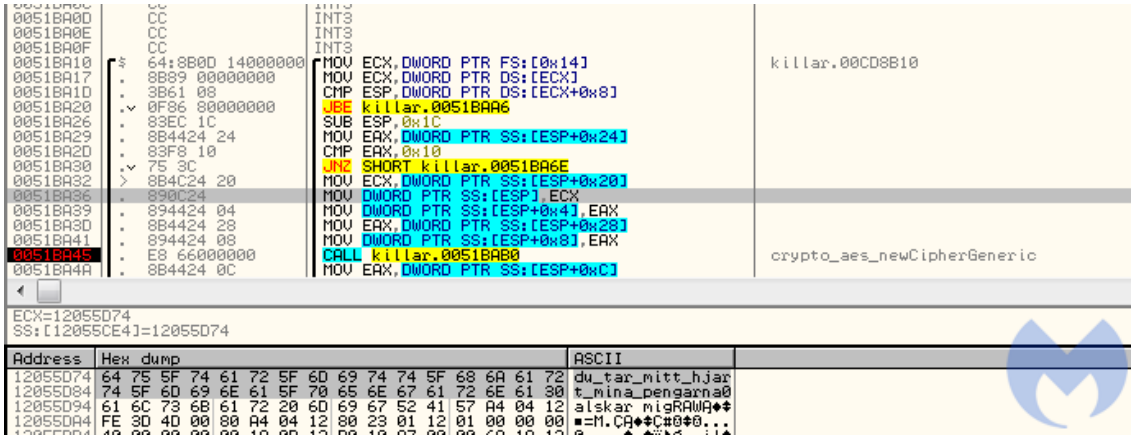


The content of the output file (with .VAGGEN extension) contains:

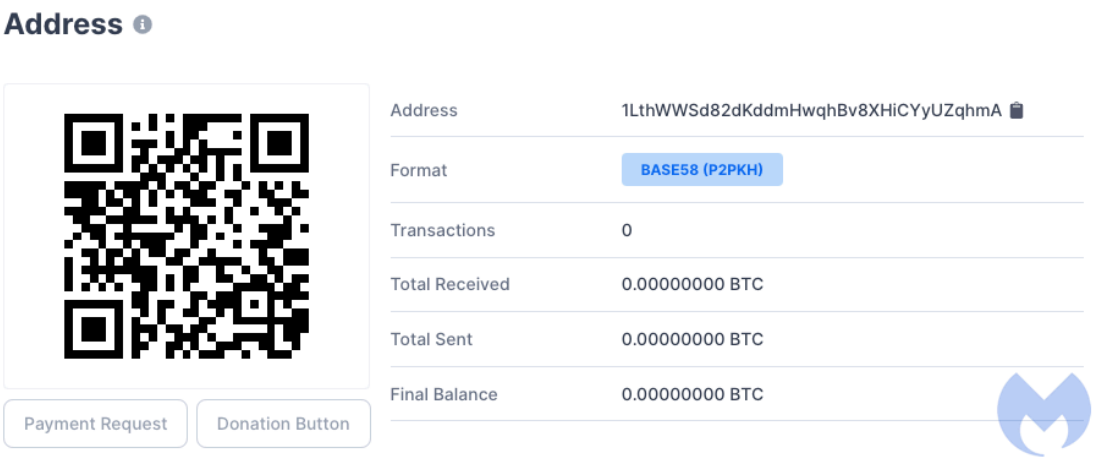
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

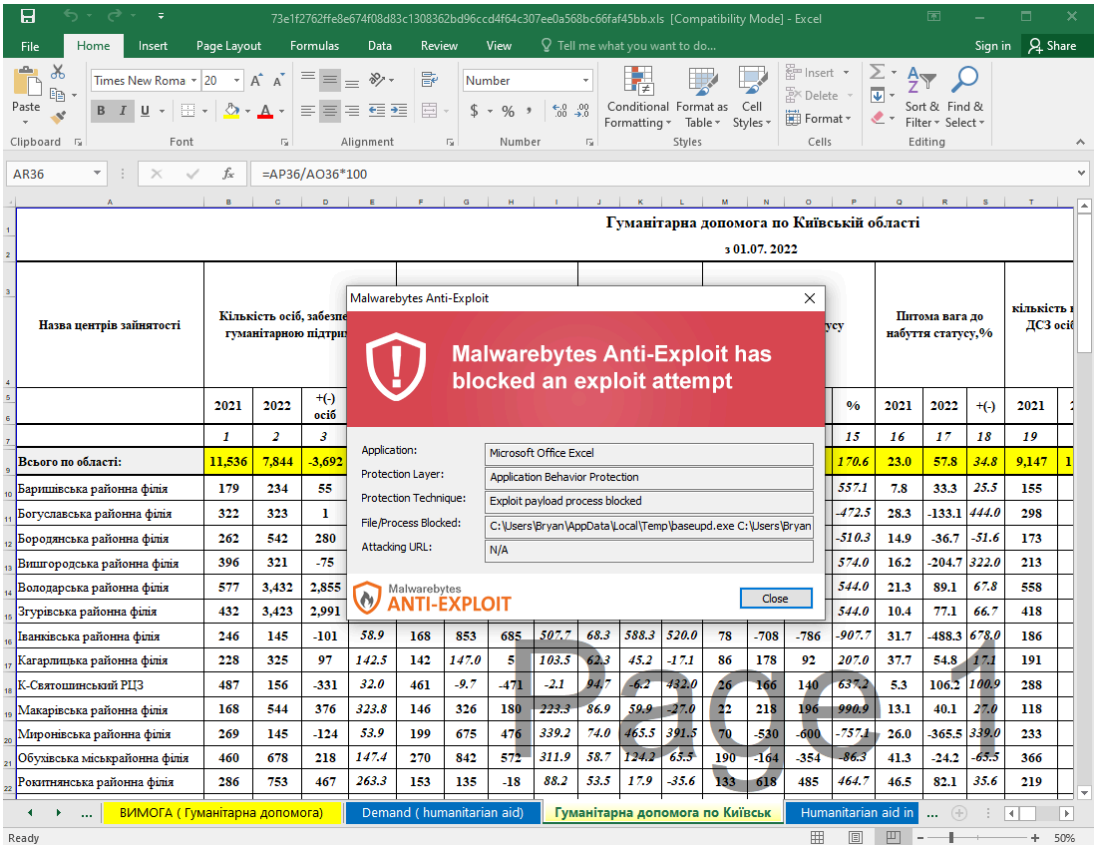
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

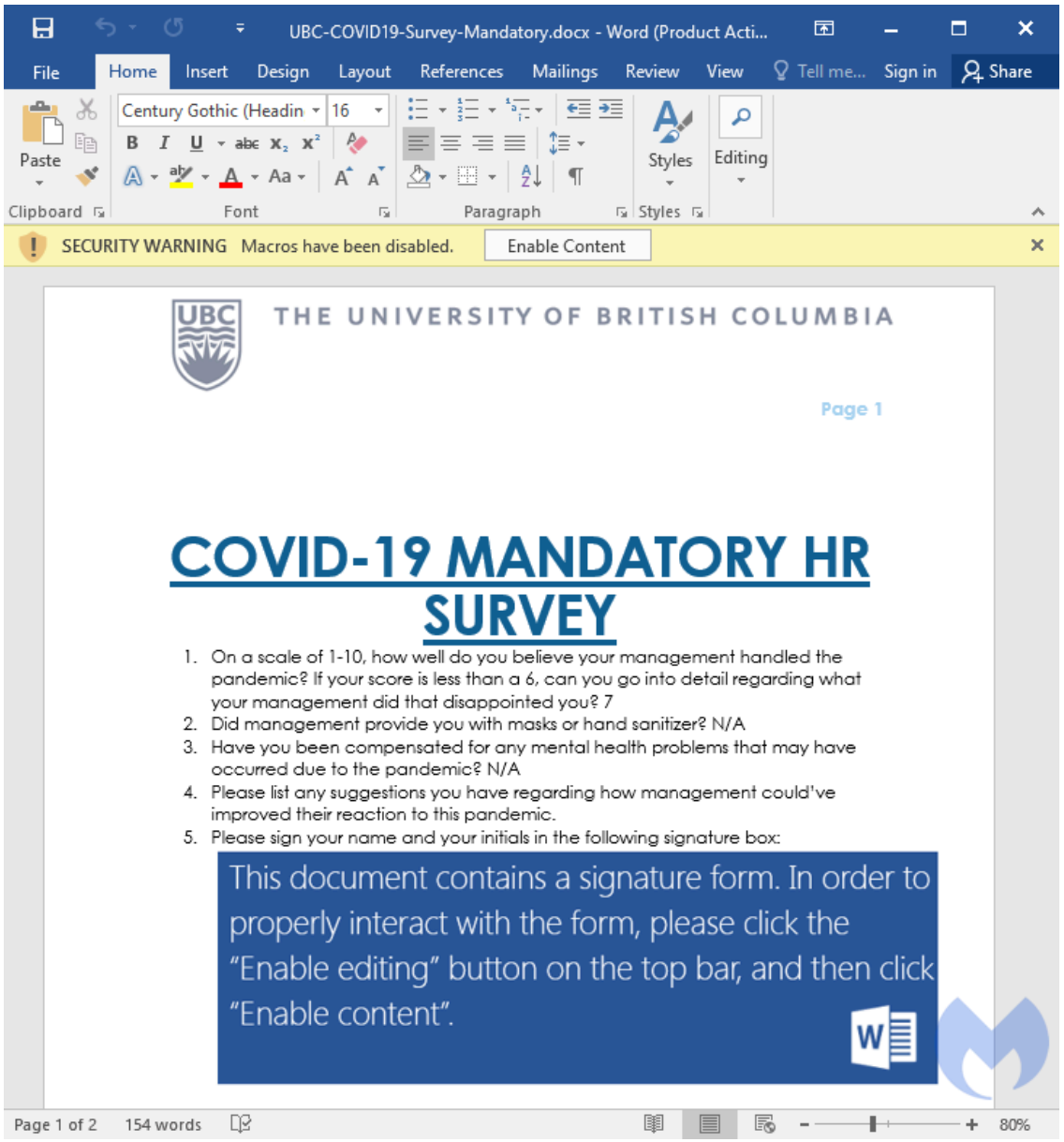
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

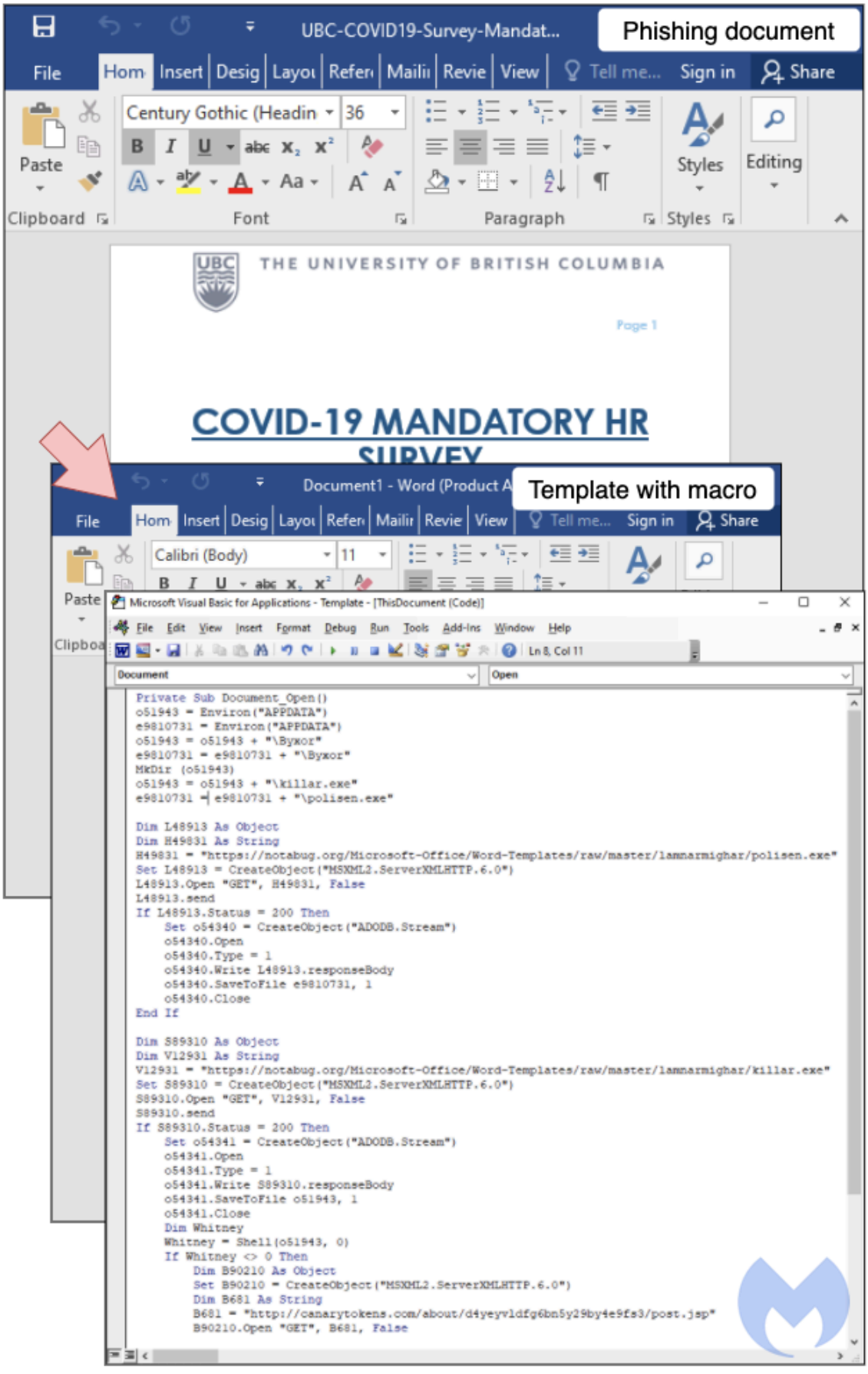
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

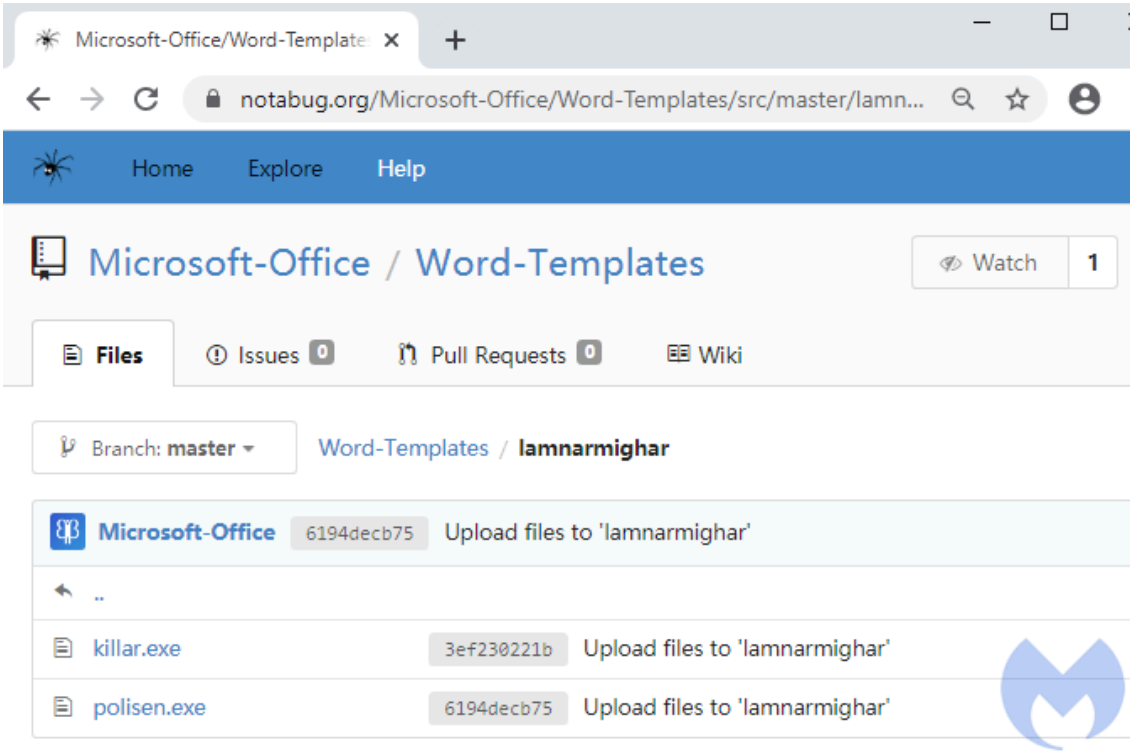
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



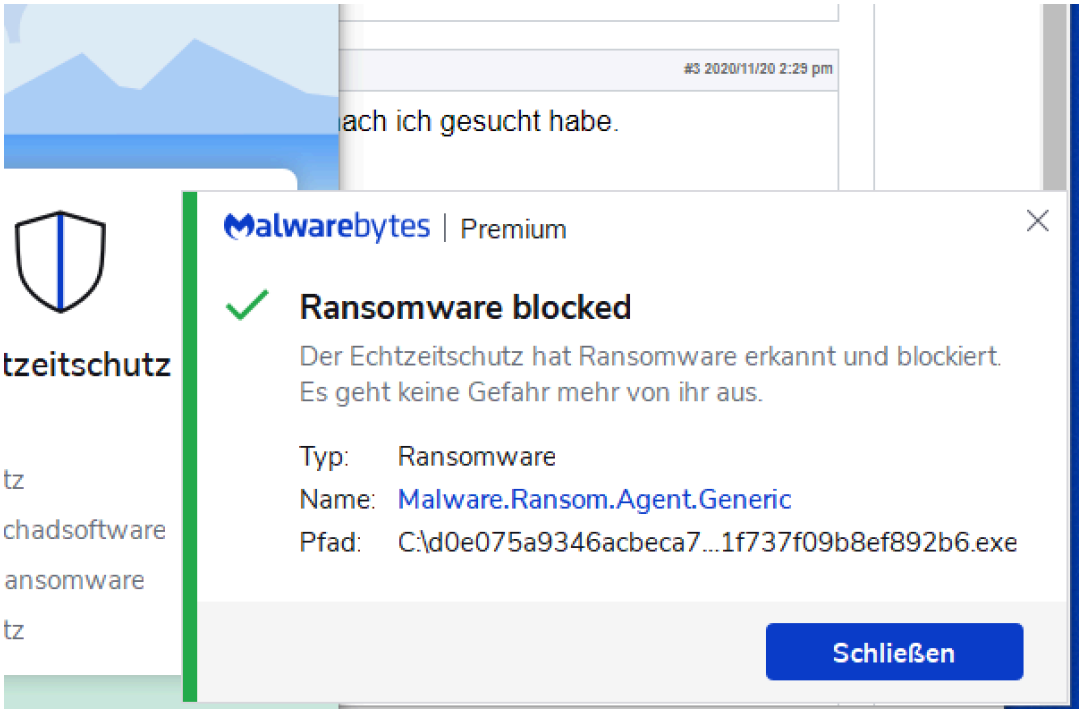
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

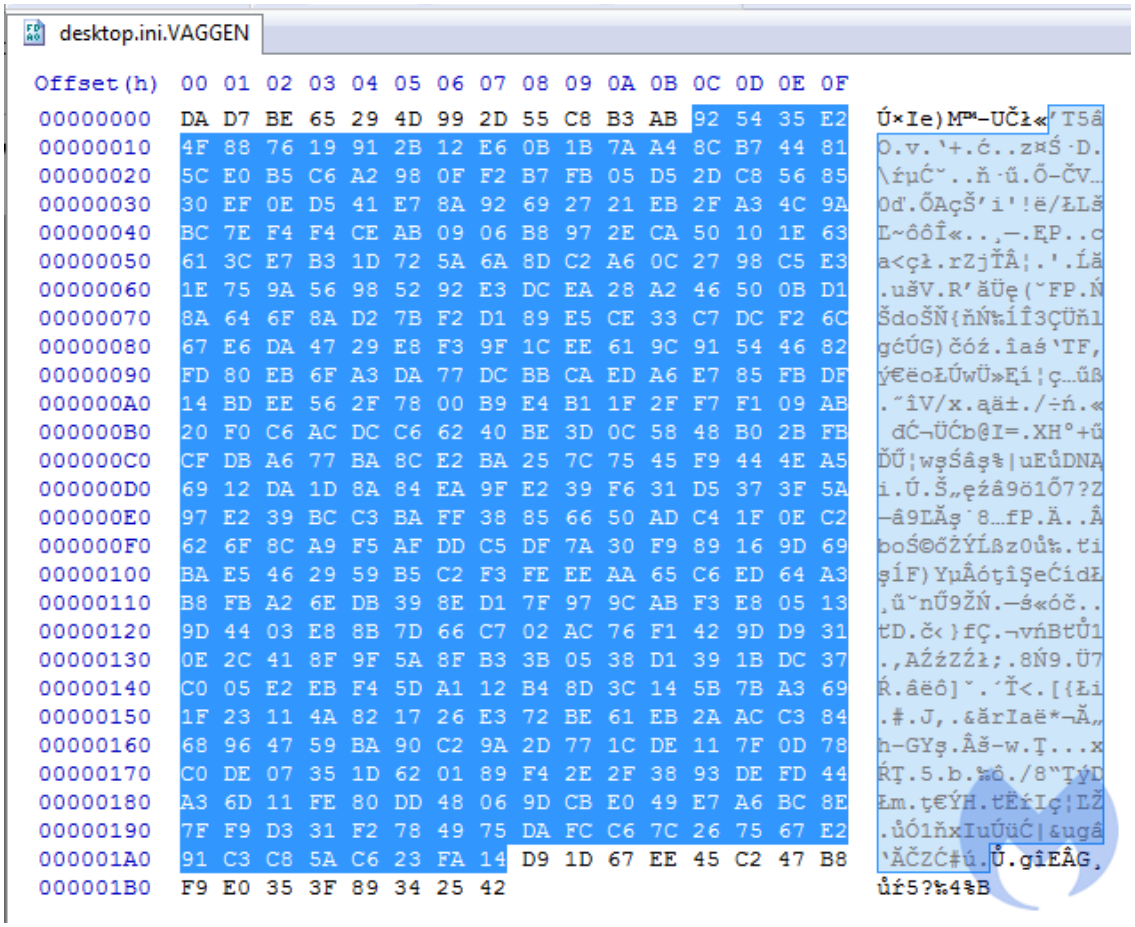
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

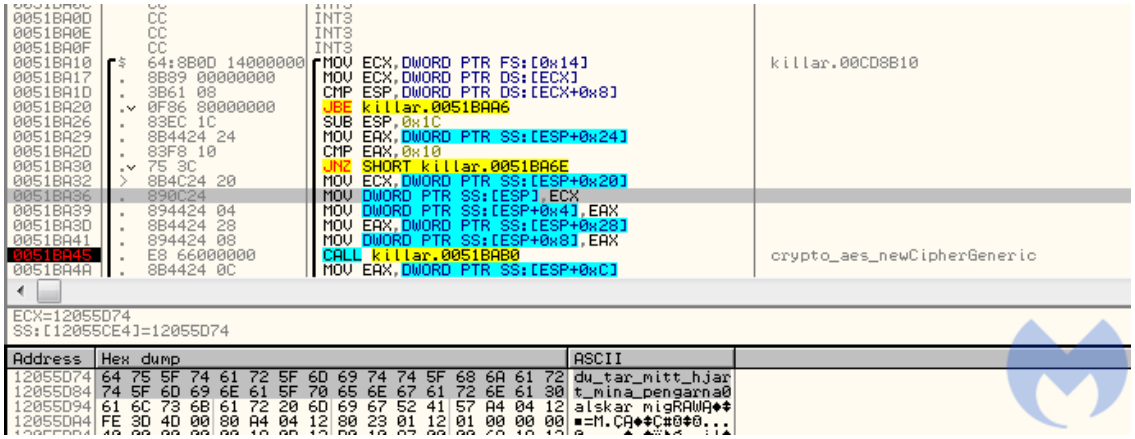


The content of the output file (with .VAGGEN extension) contains:

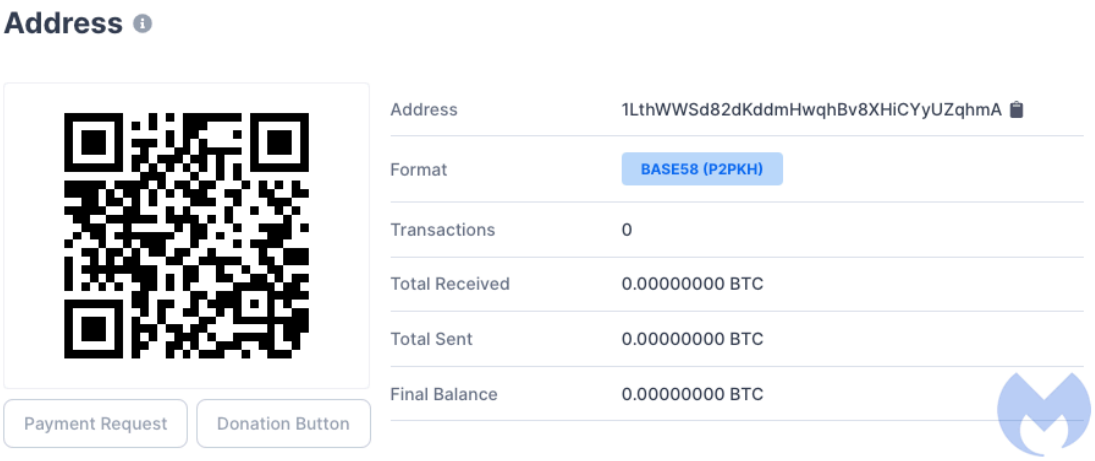
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

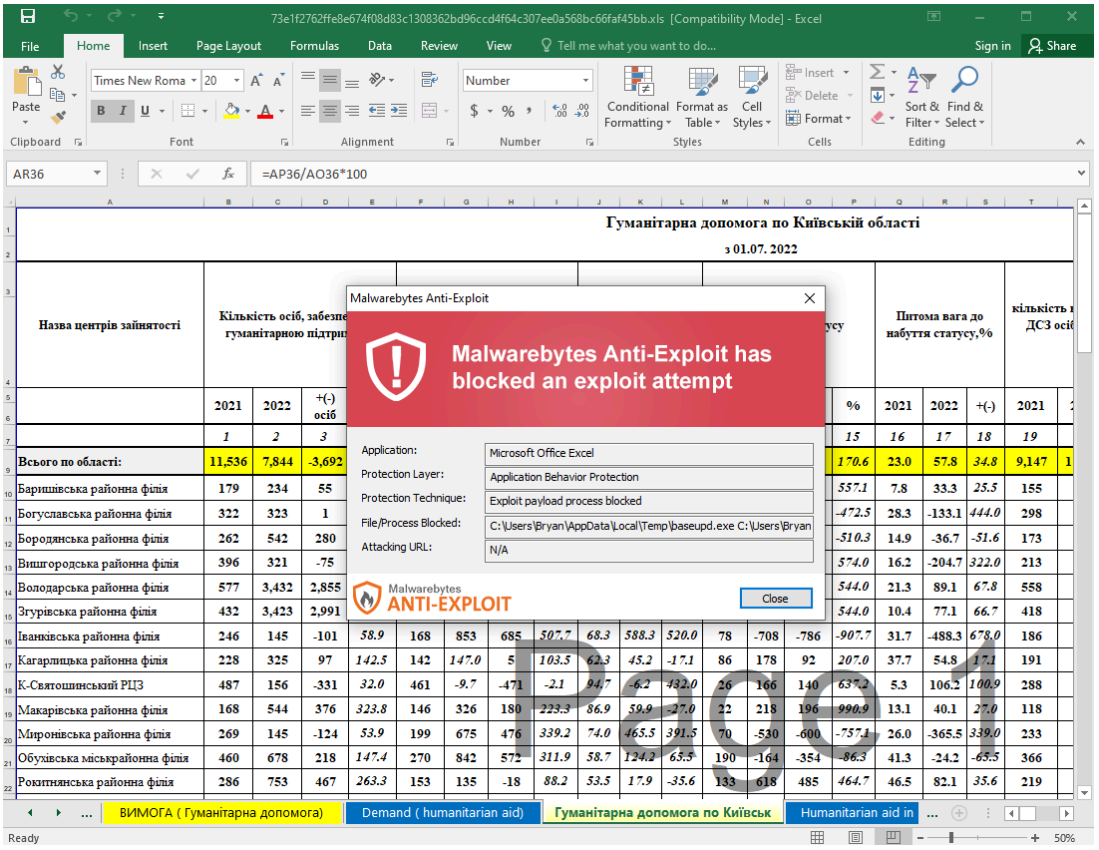
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

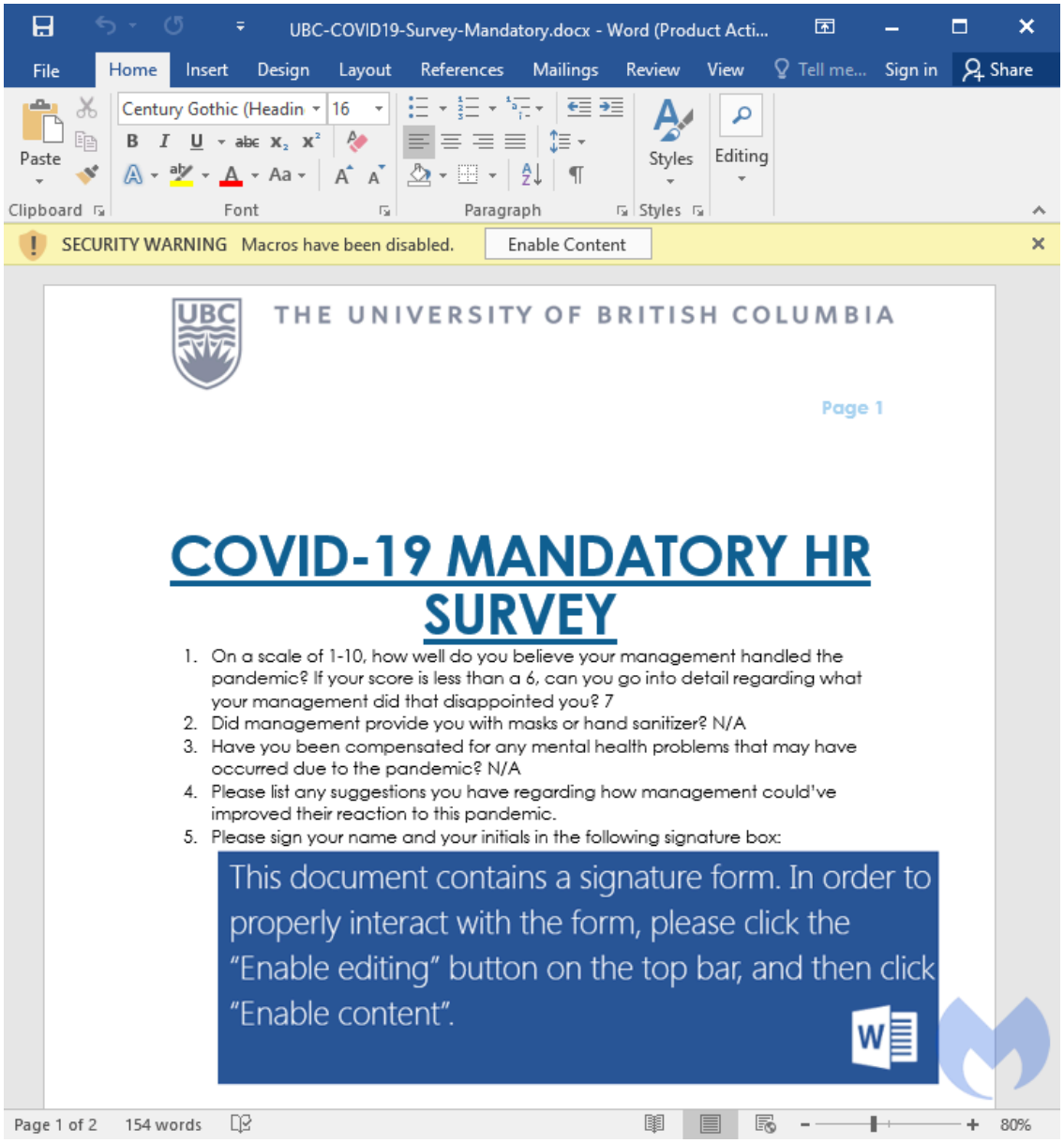
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

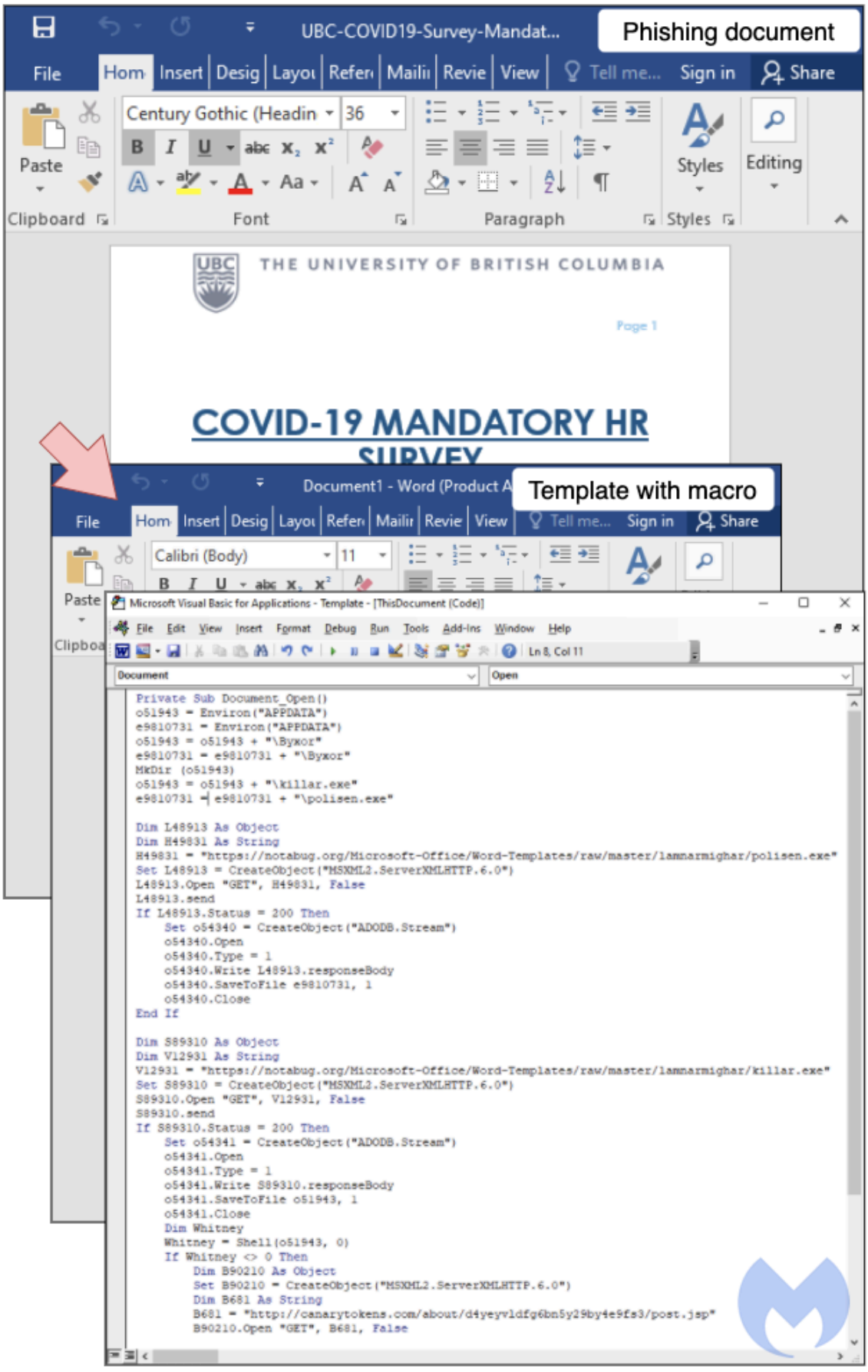
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

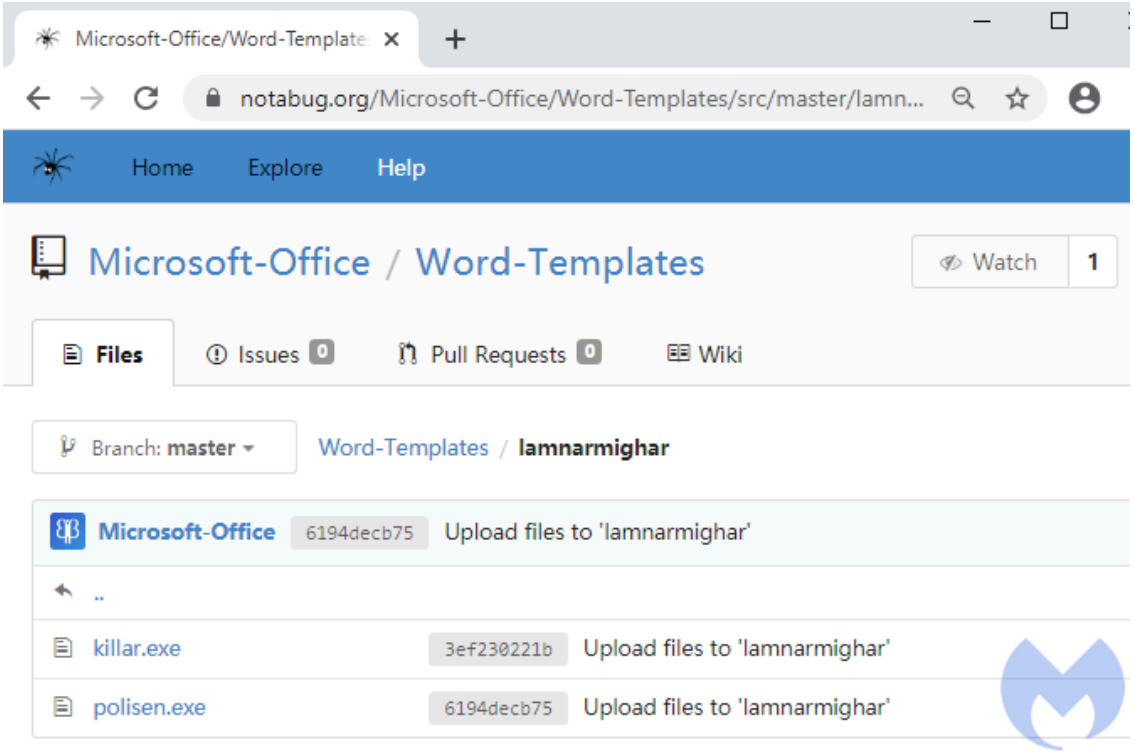
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



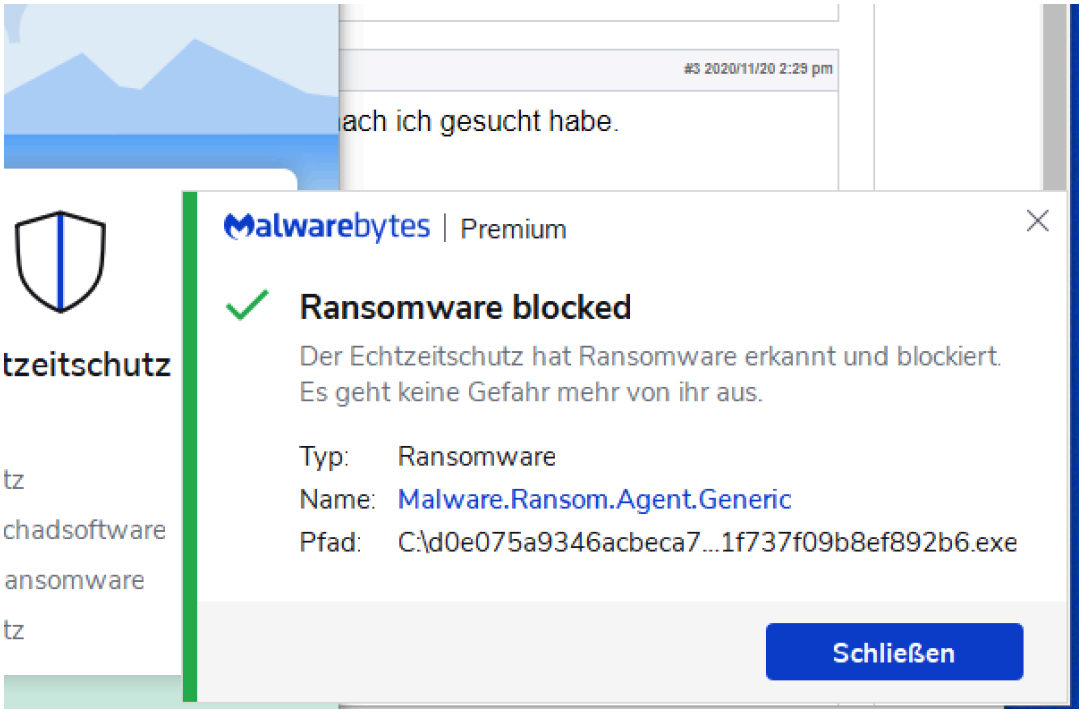
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

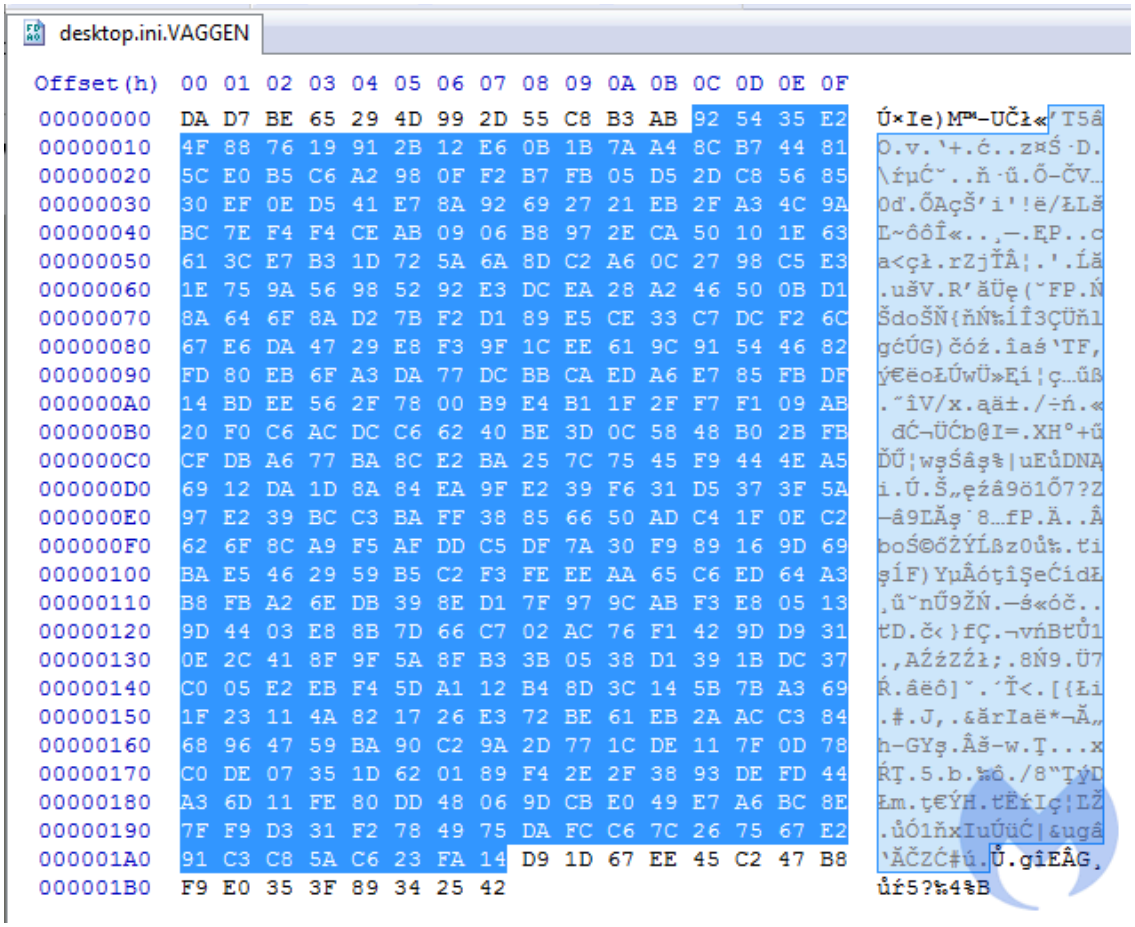
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

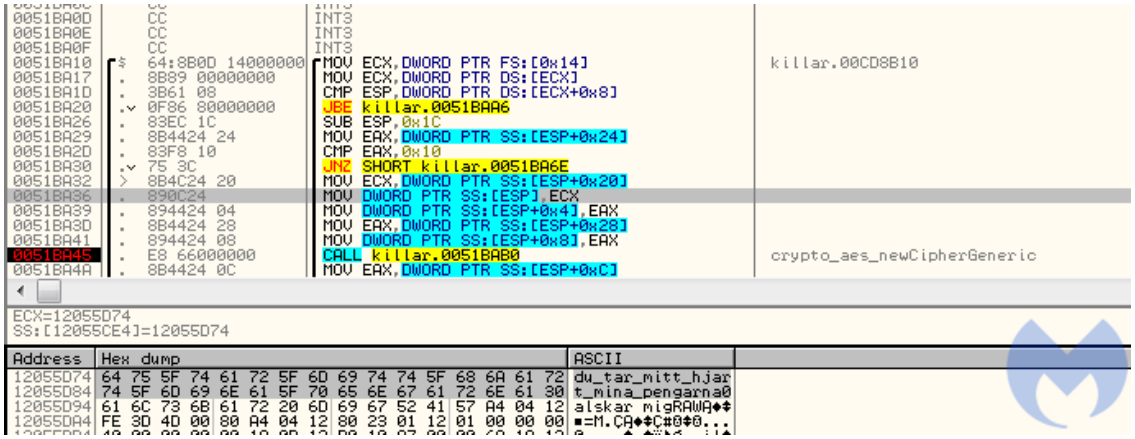


The content of the output file (with .VAGGEN extension) contains:

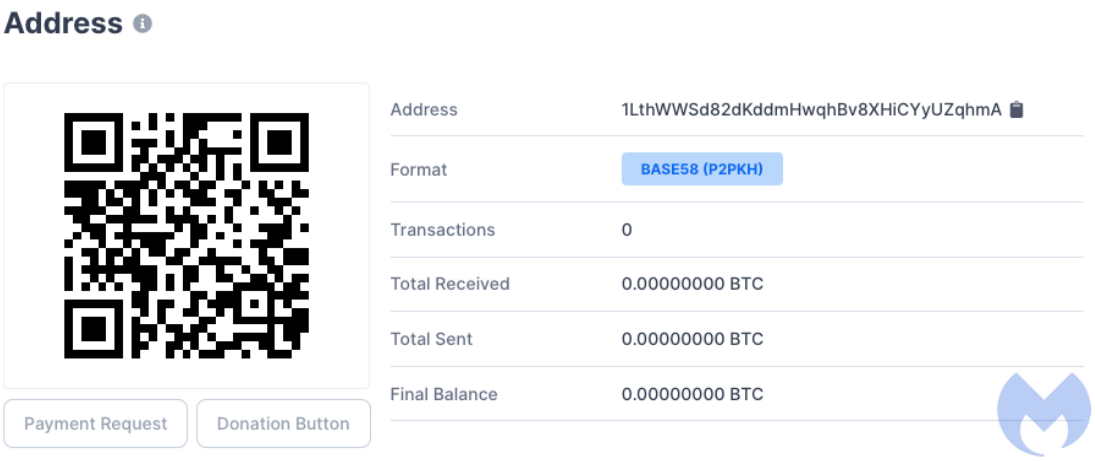
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

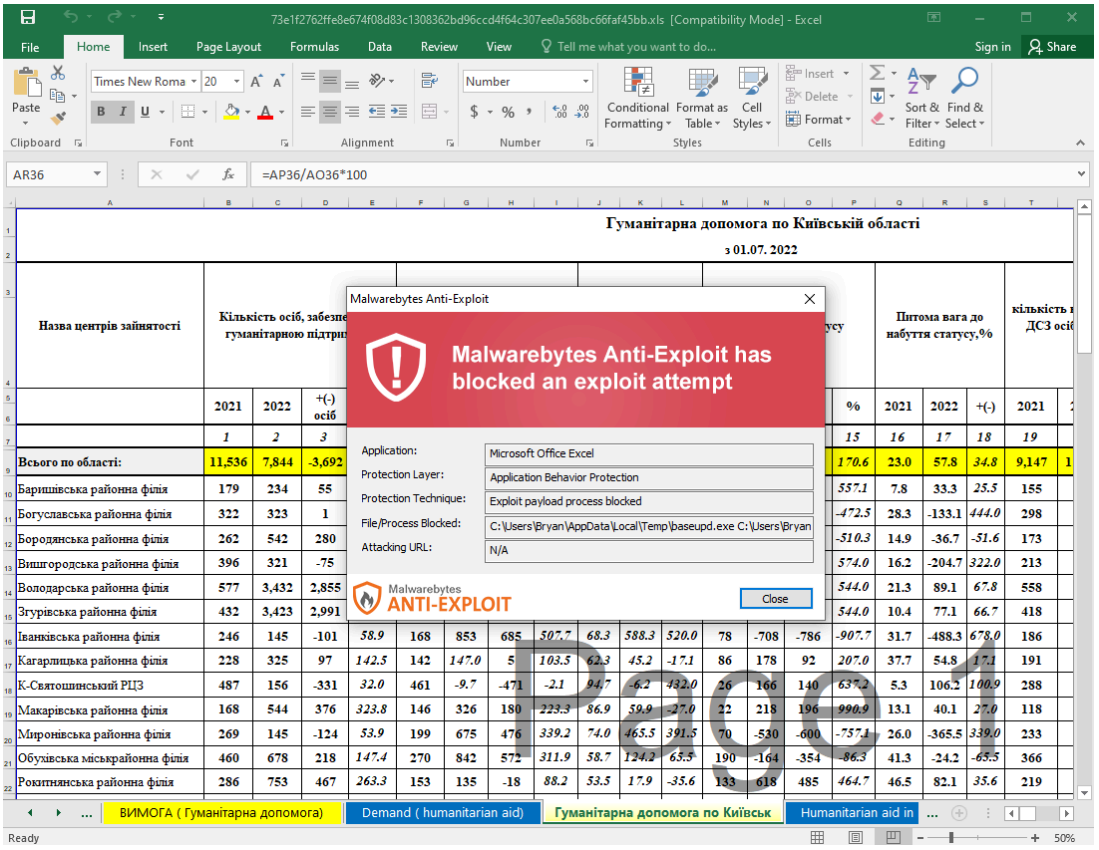
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

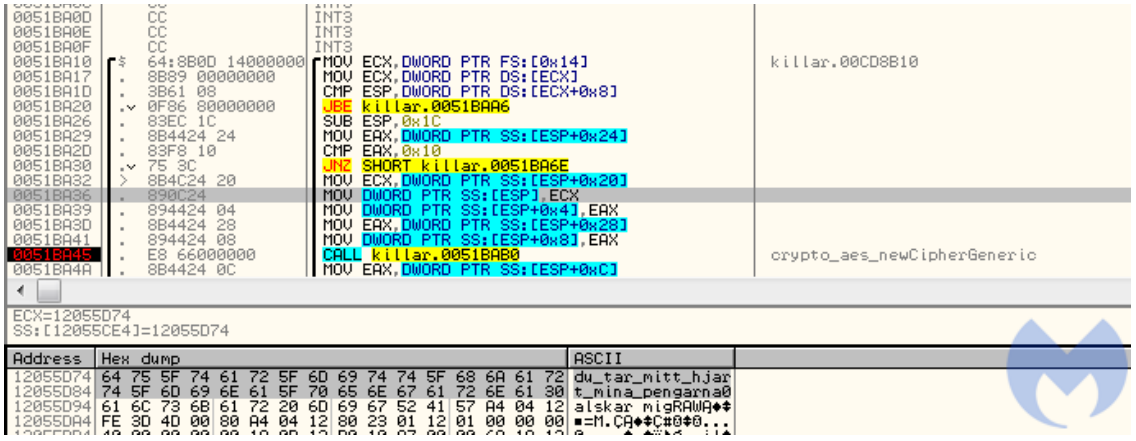
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

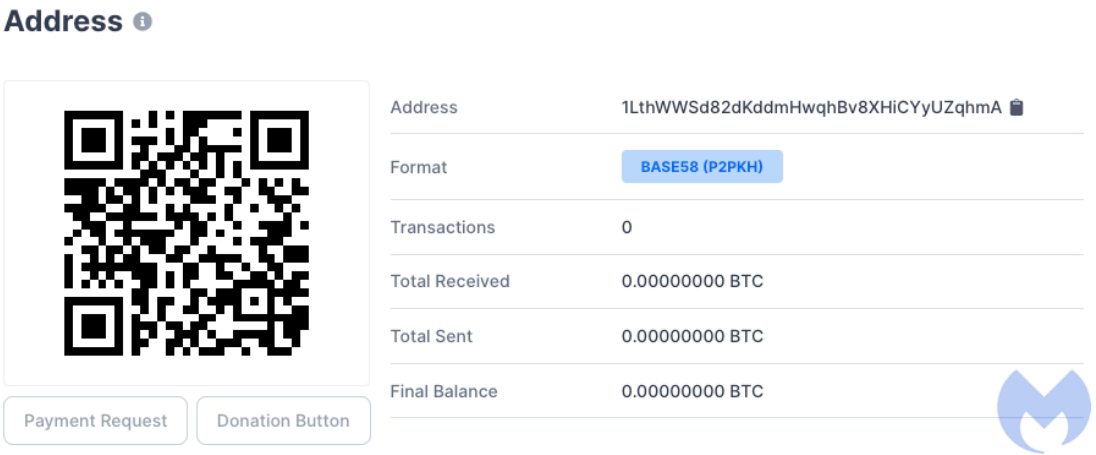
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

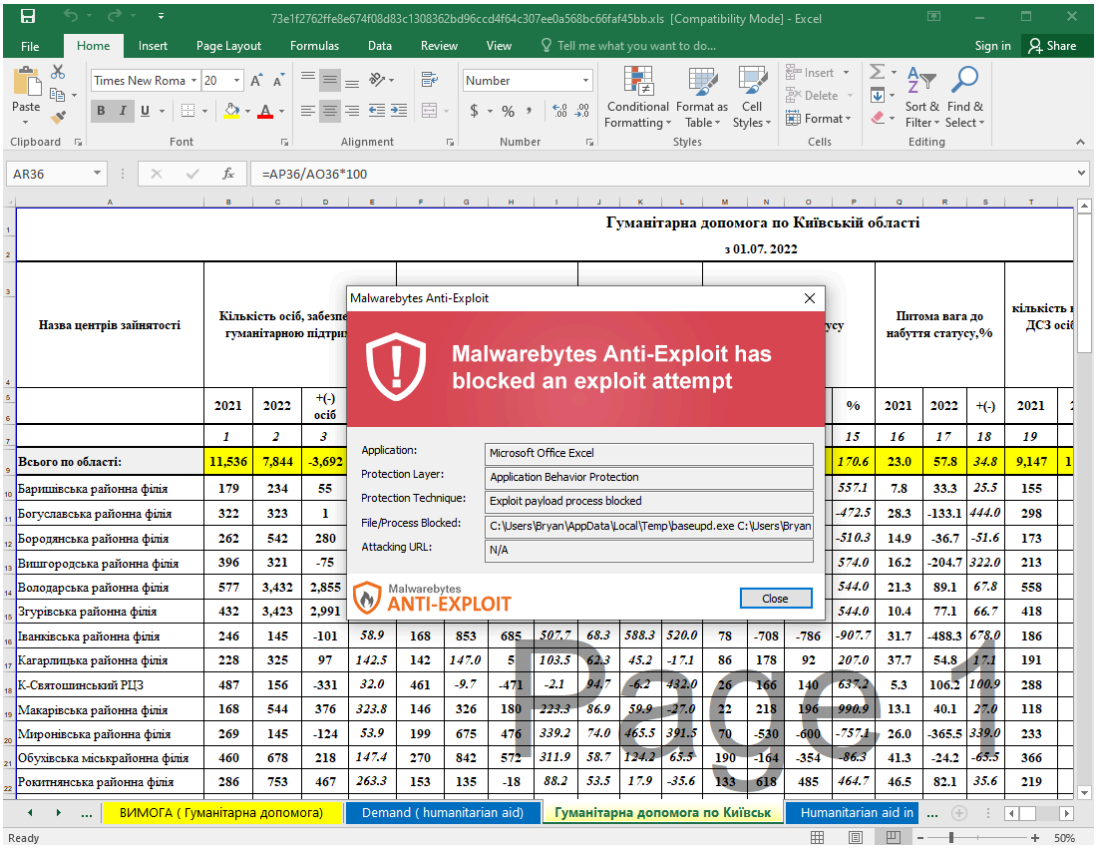
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

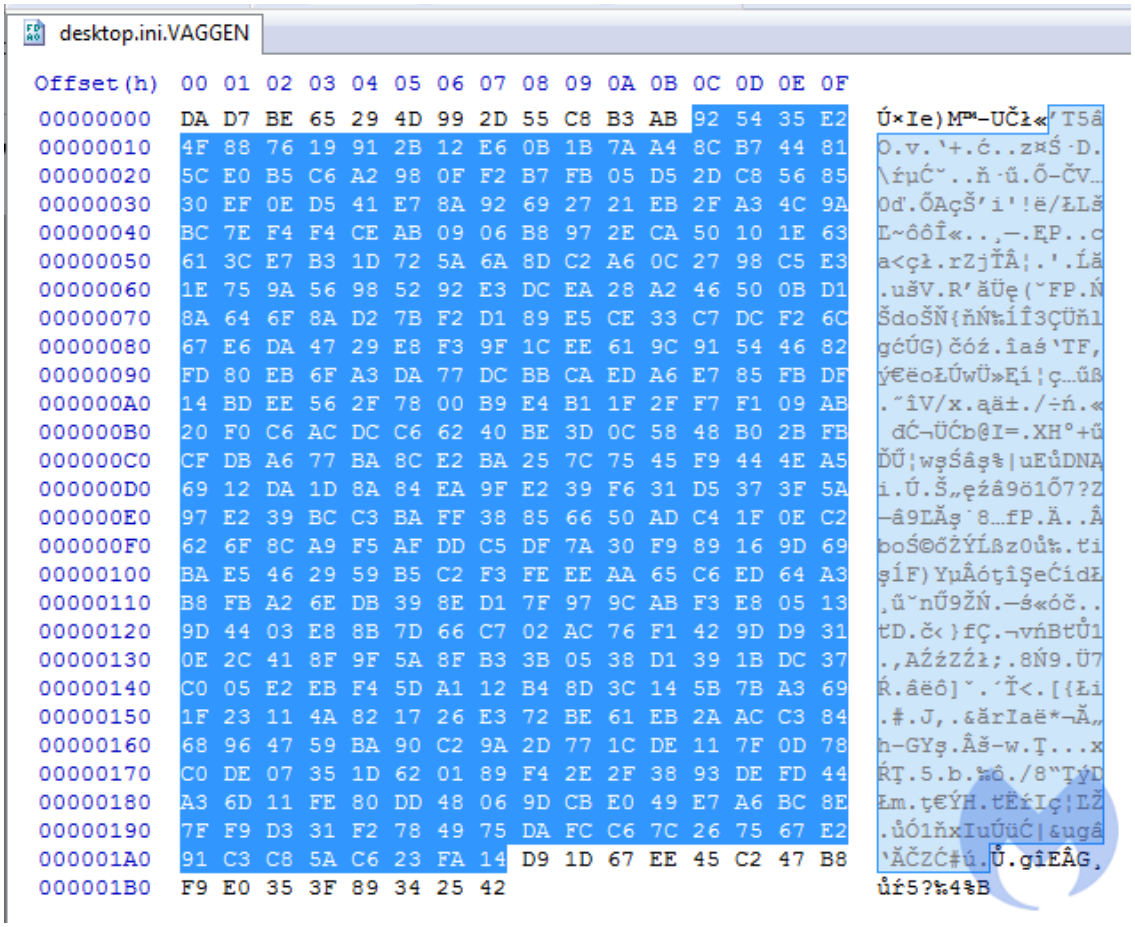
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

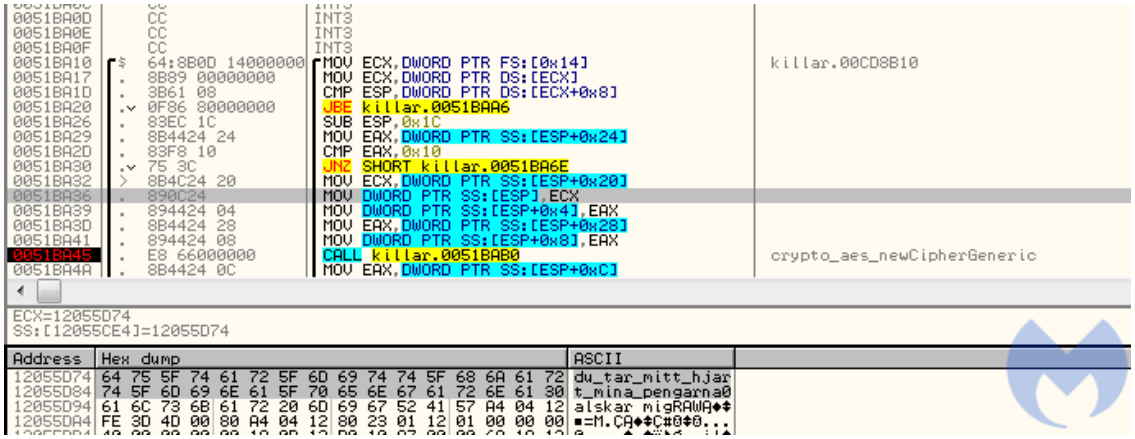


The content of the output file (with .VAGGEN extension) contains:

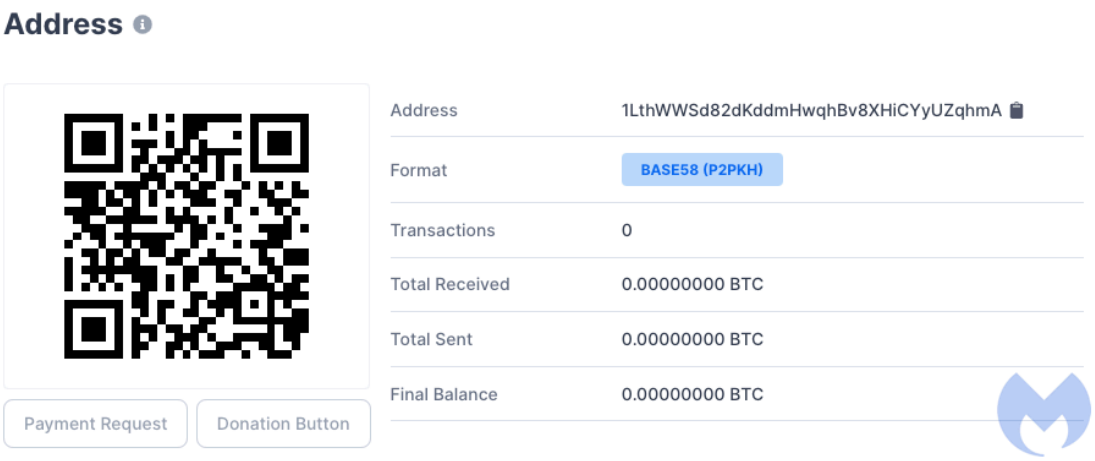
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

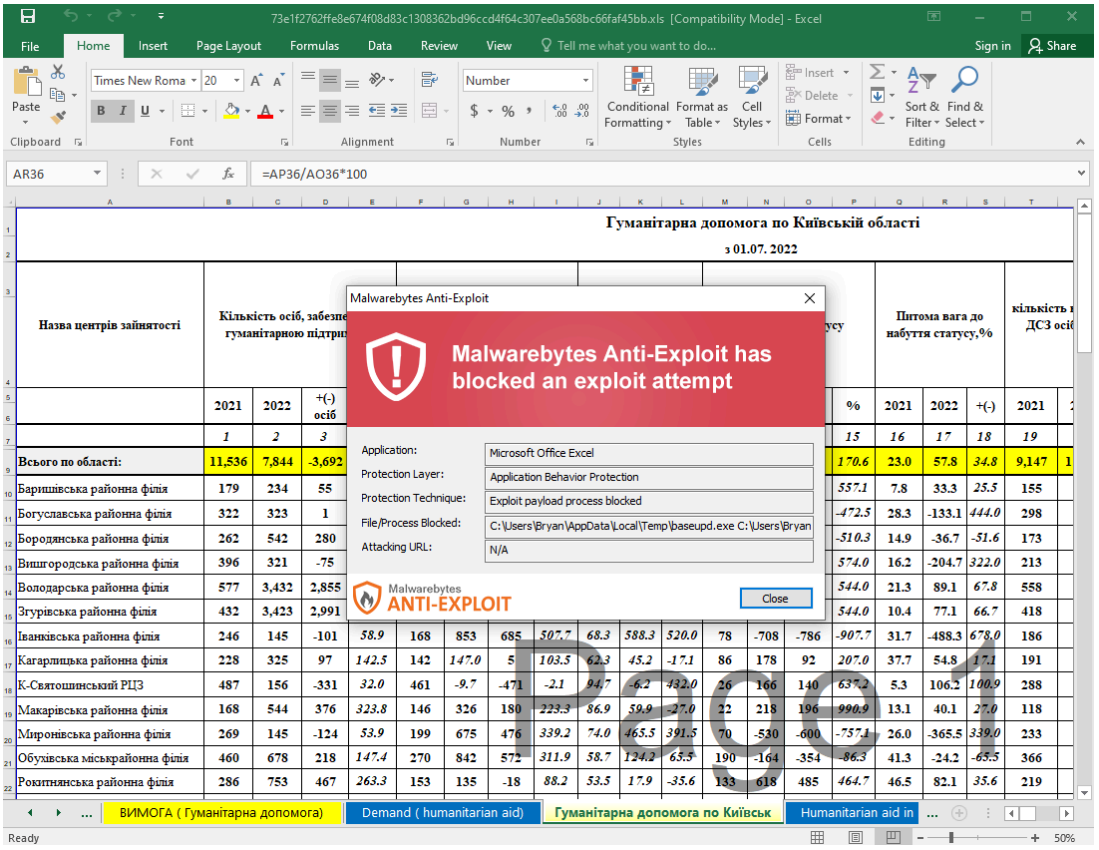
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

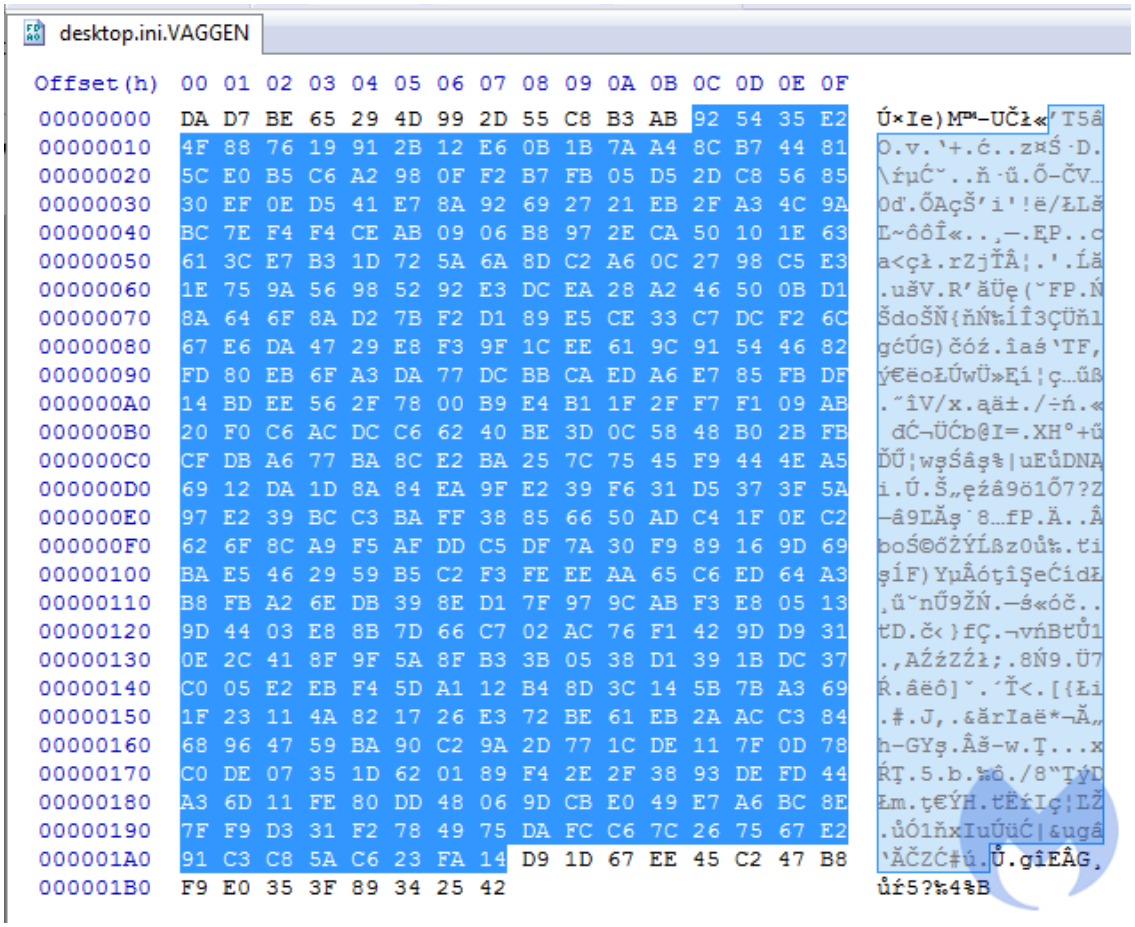
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

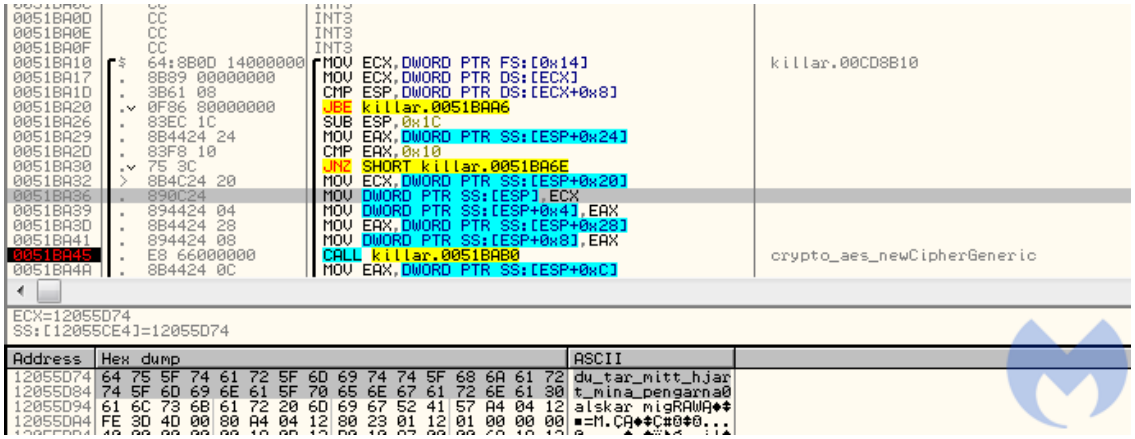


The content of the output file (with .VAGGEN extension) contains:

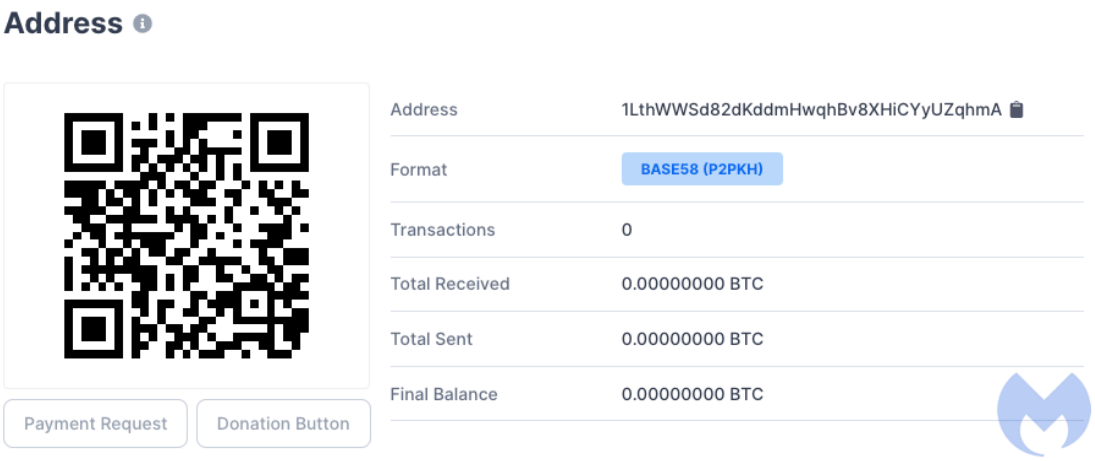
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

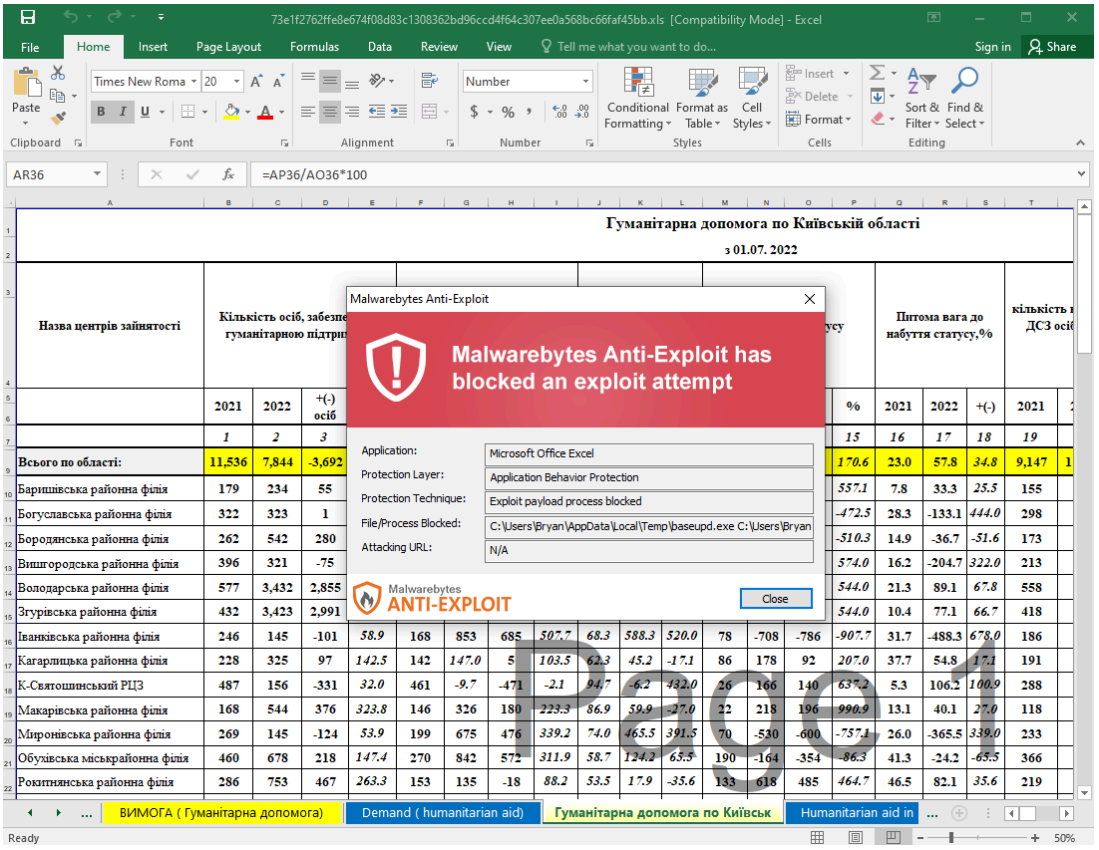
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

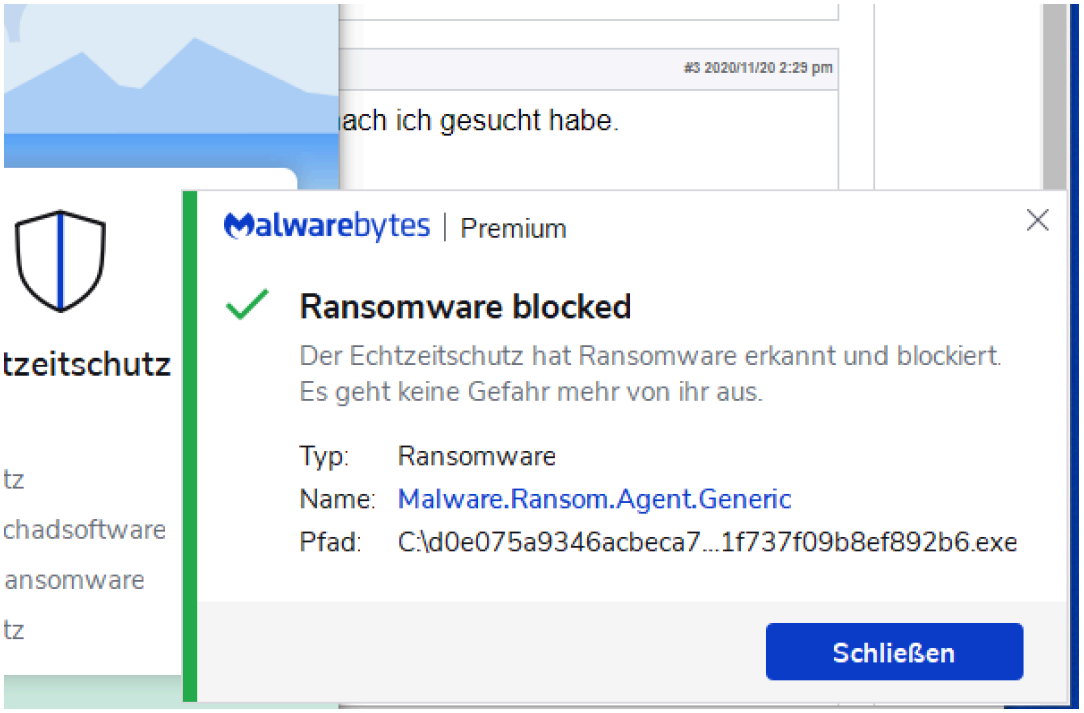
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

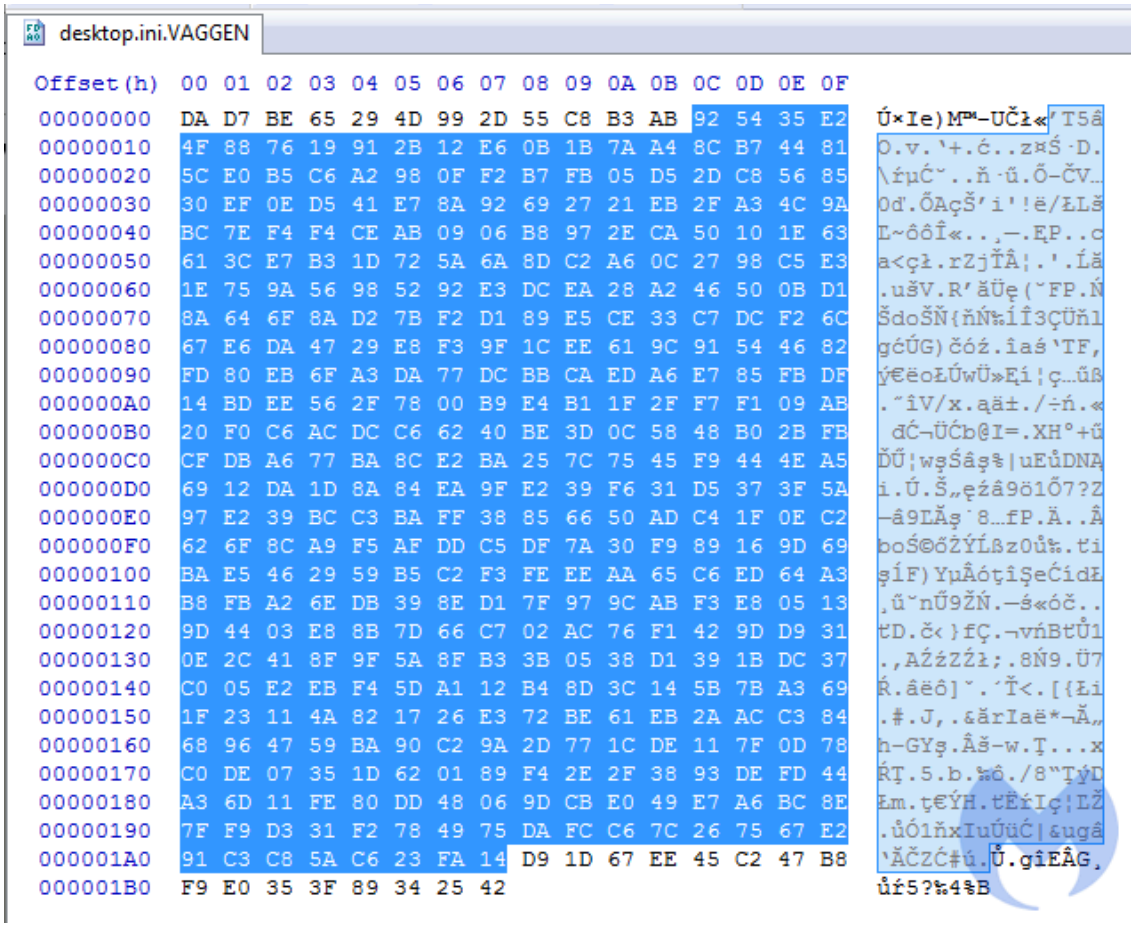
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

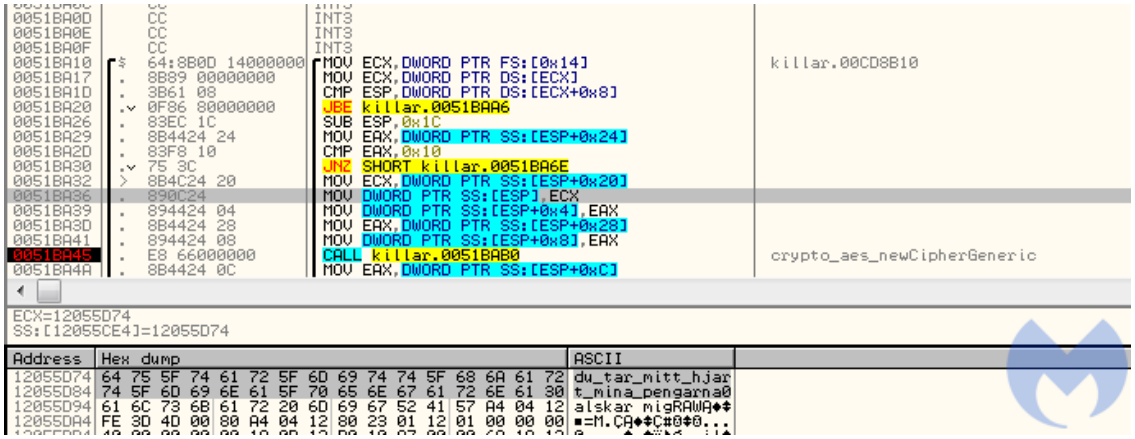


The content of the output file (with .VAGGEN extension) contains:

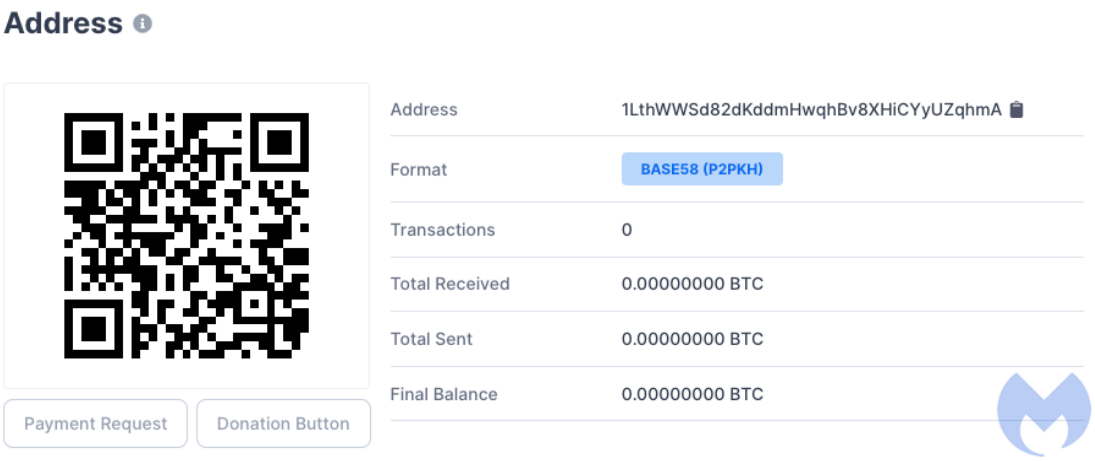
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

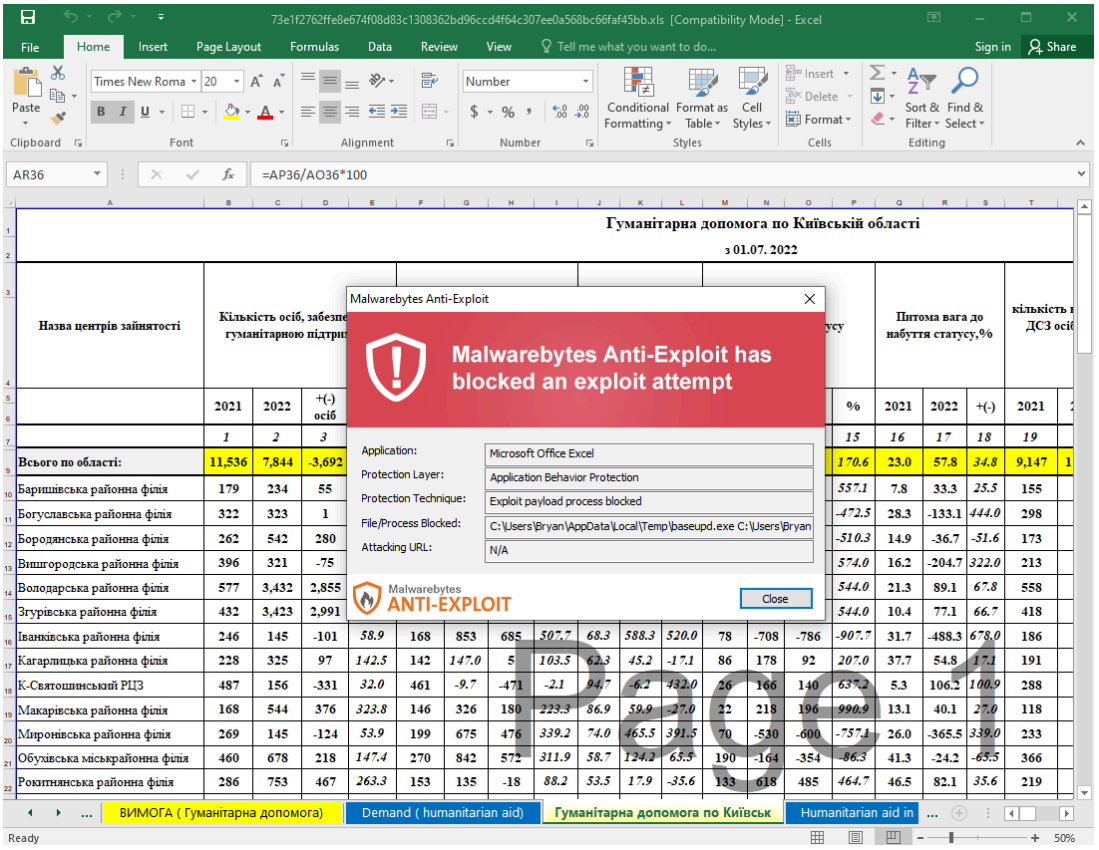
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

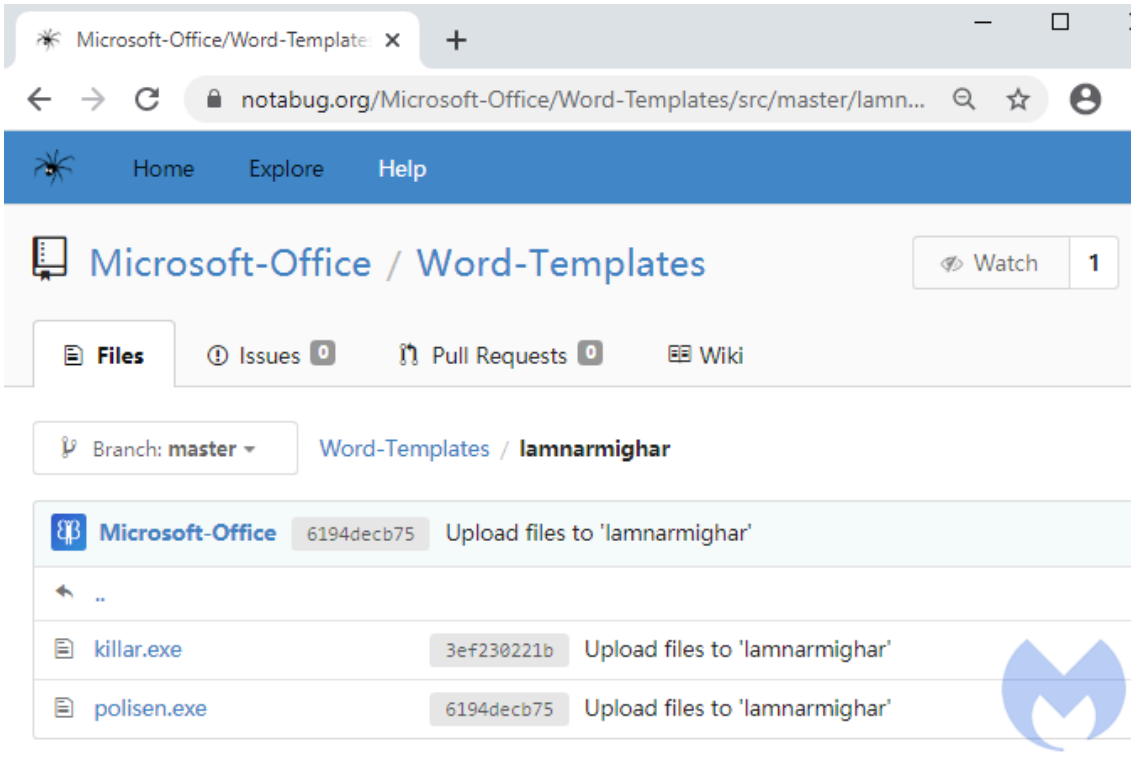
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



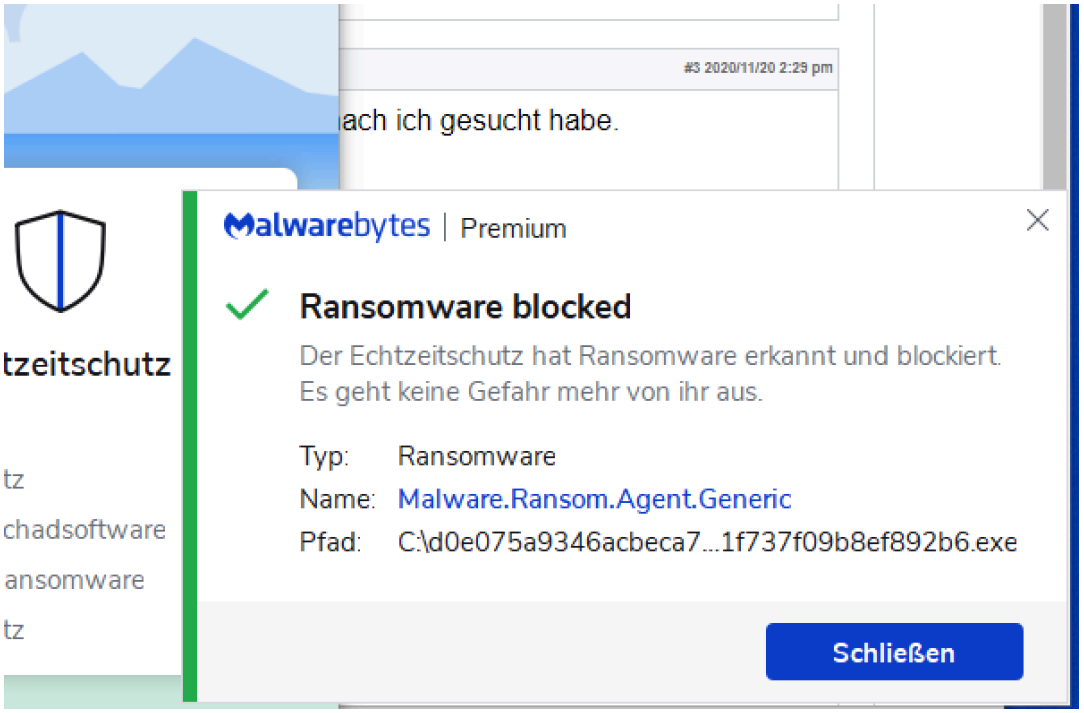
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

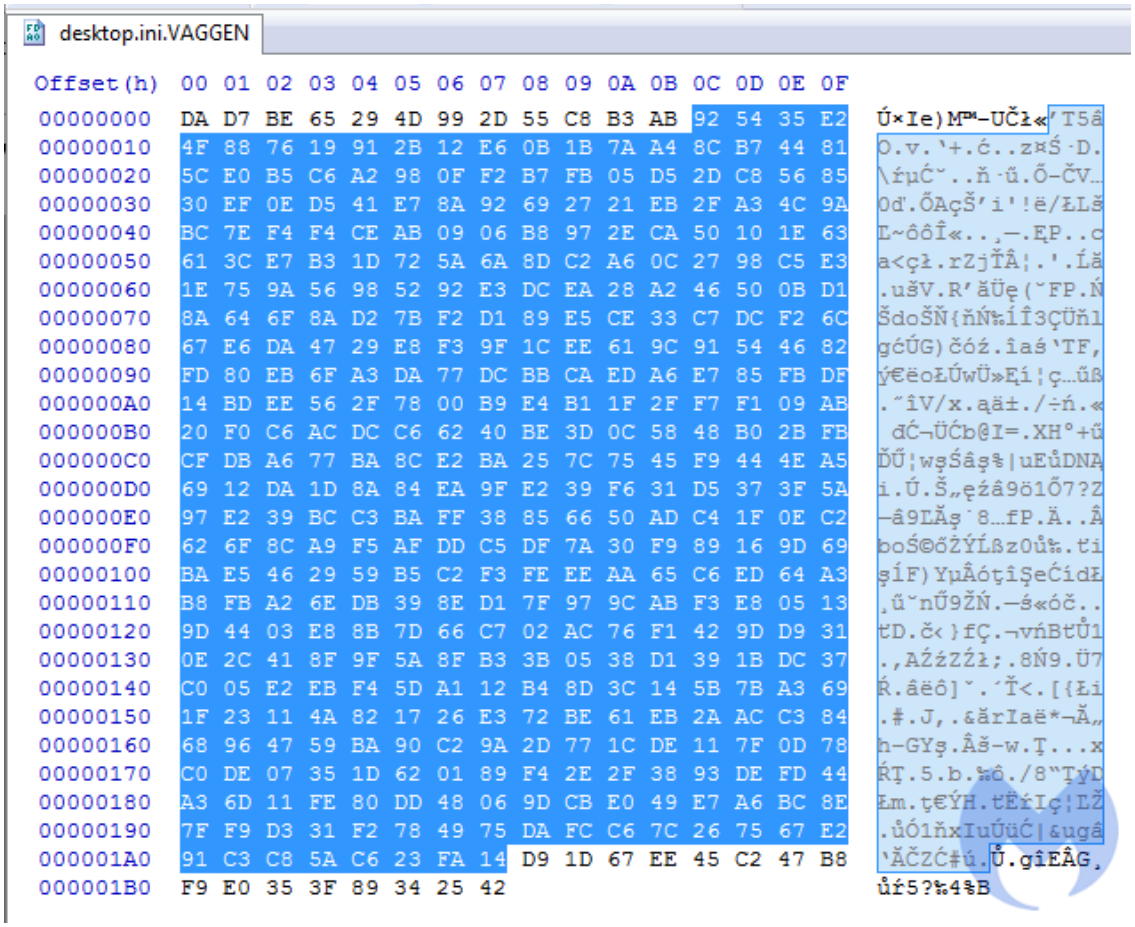
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

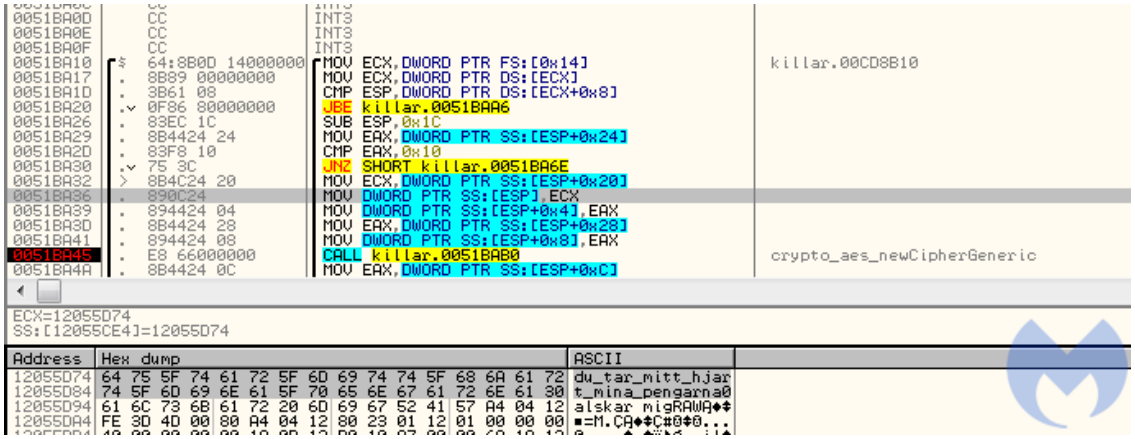


The content of the output file (with .VAGGEN extension) contains:

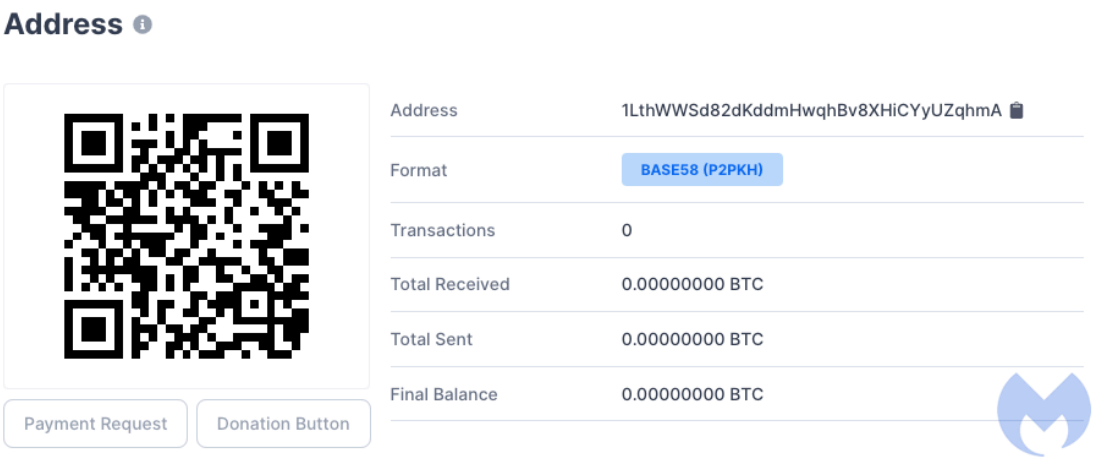
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

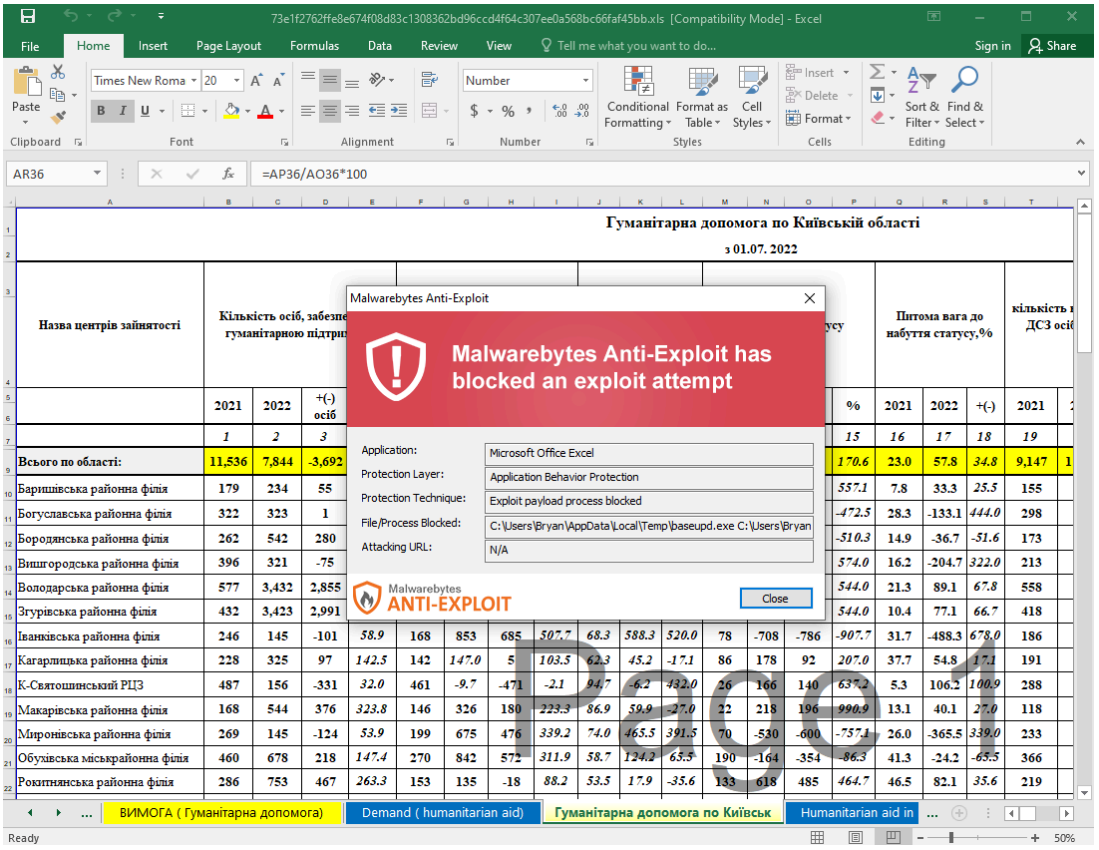
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

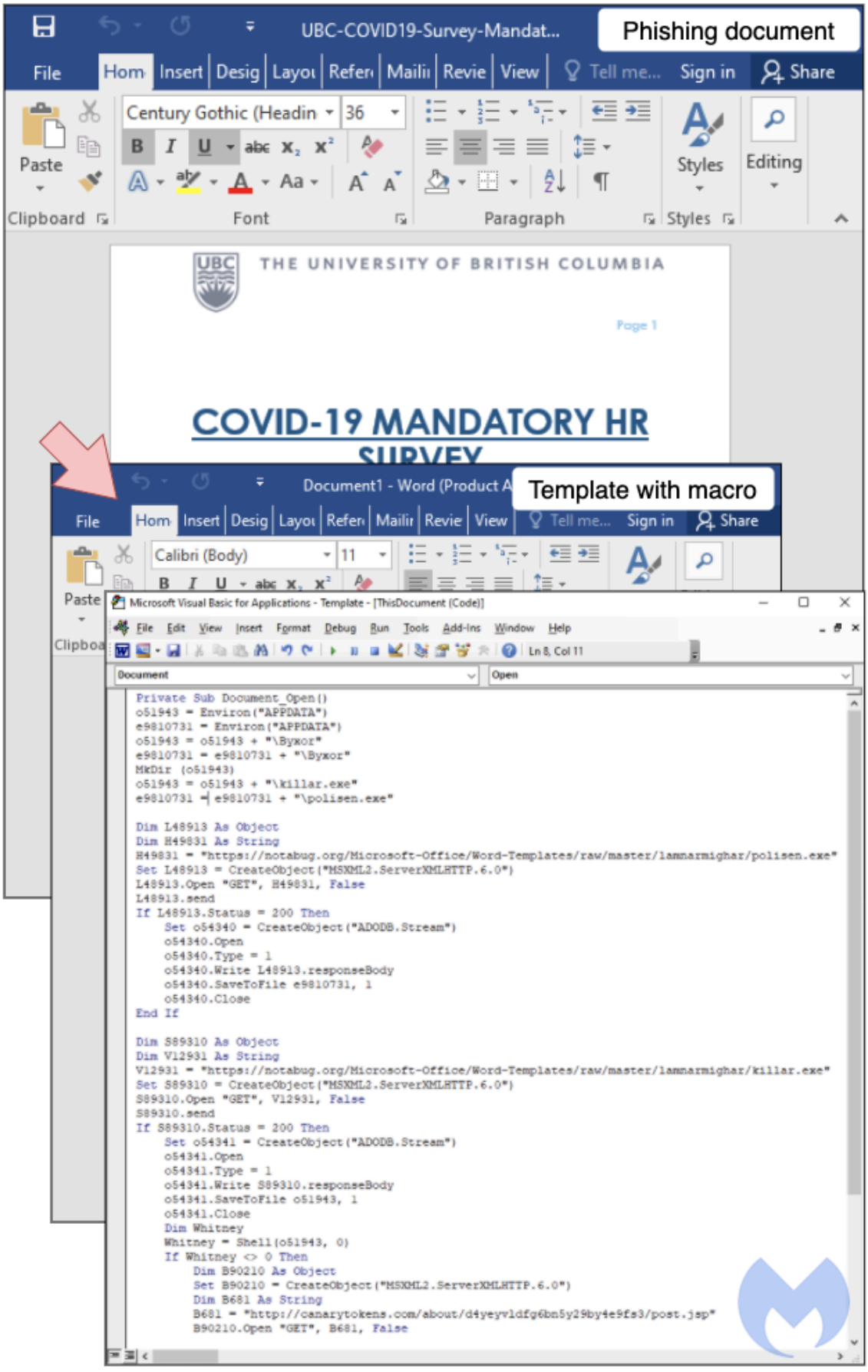
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

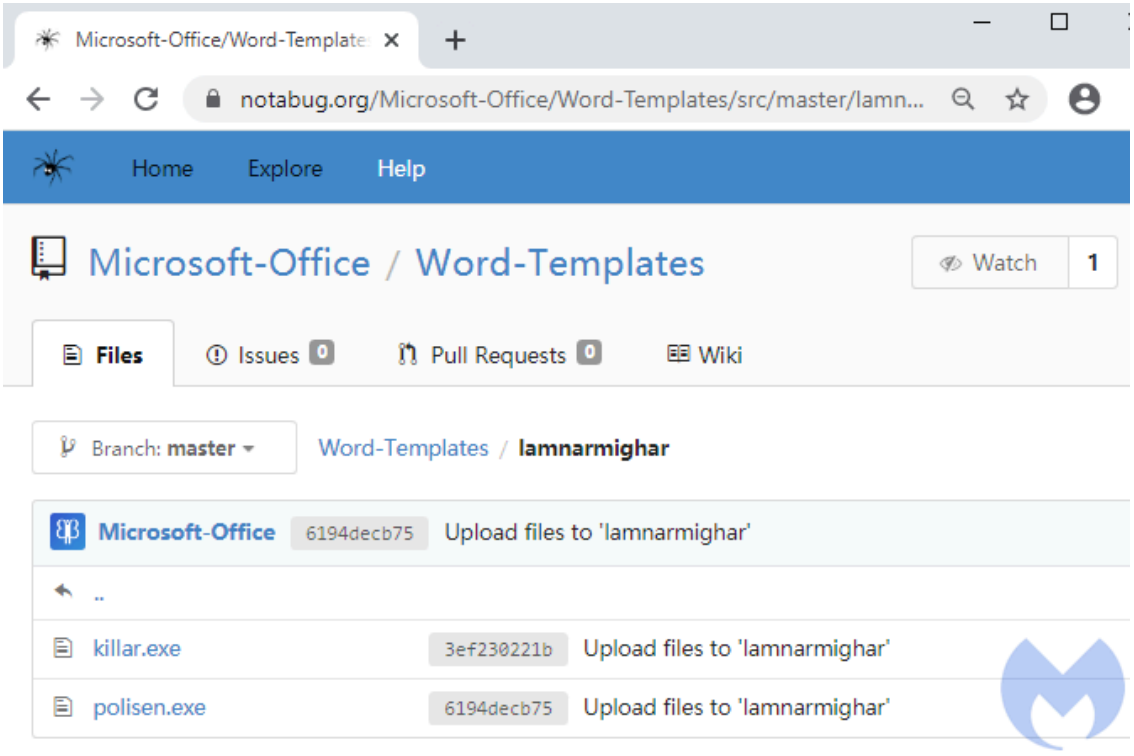
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



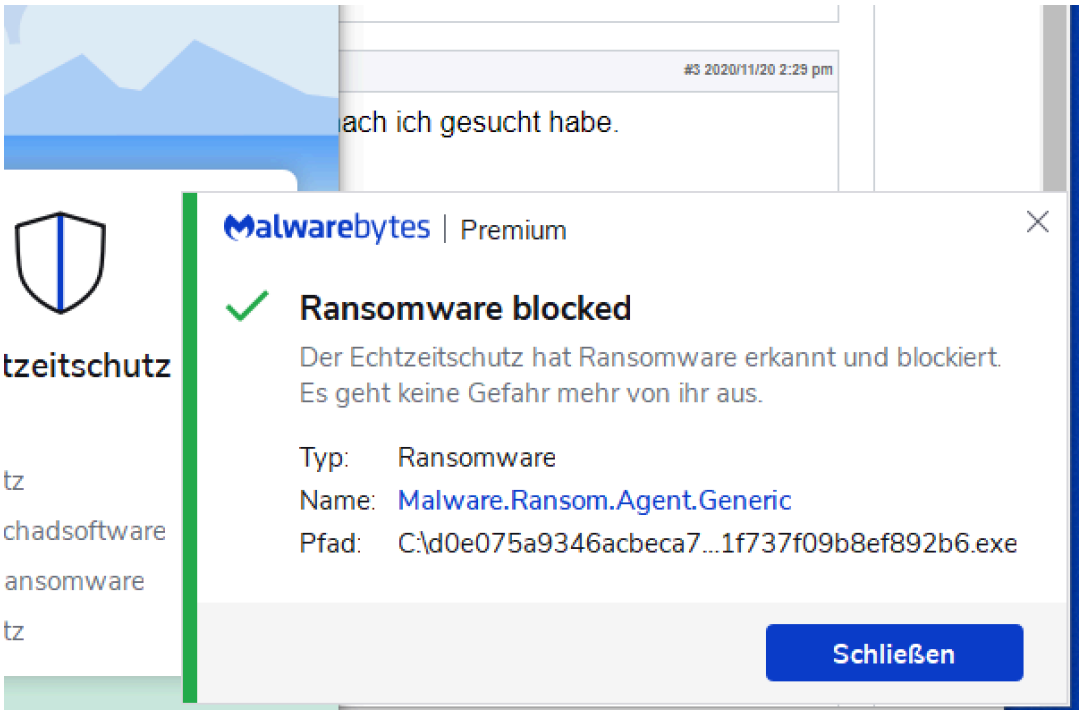
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

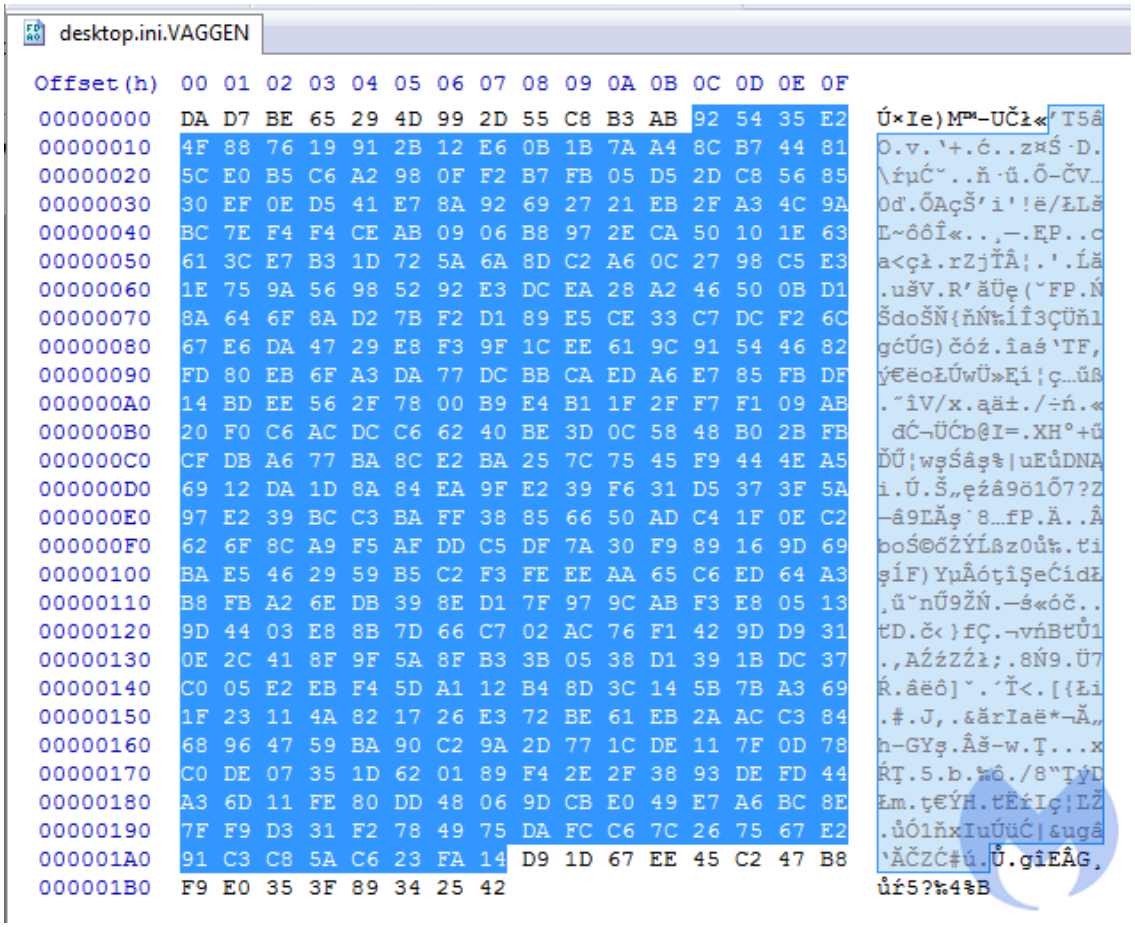
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



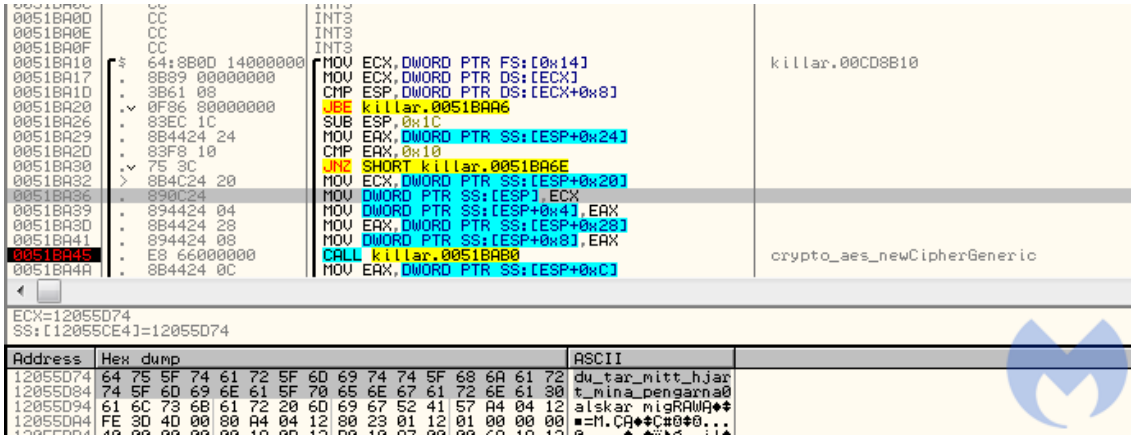
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

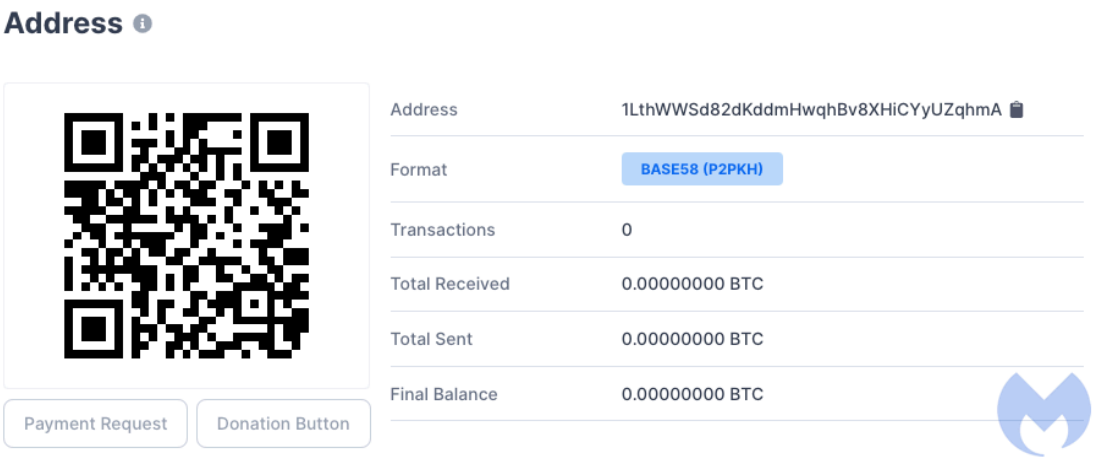
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

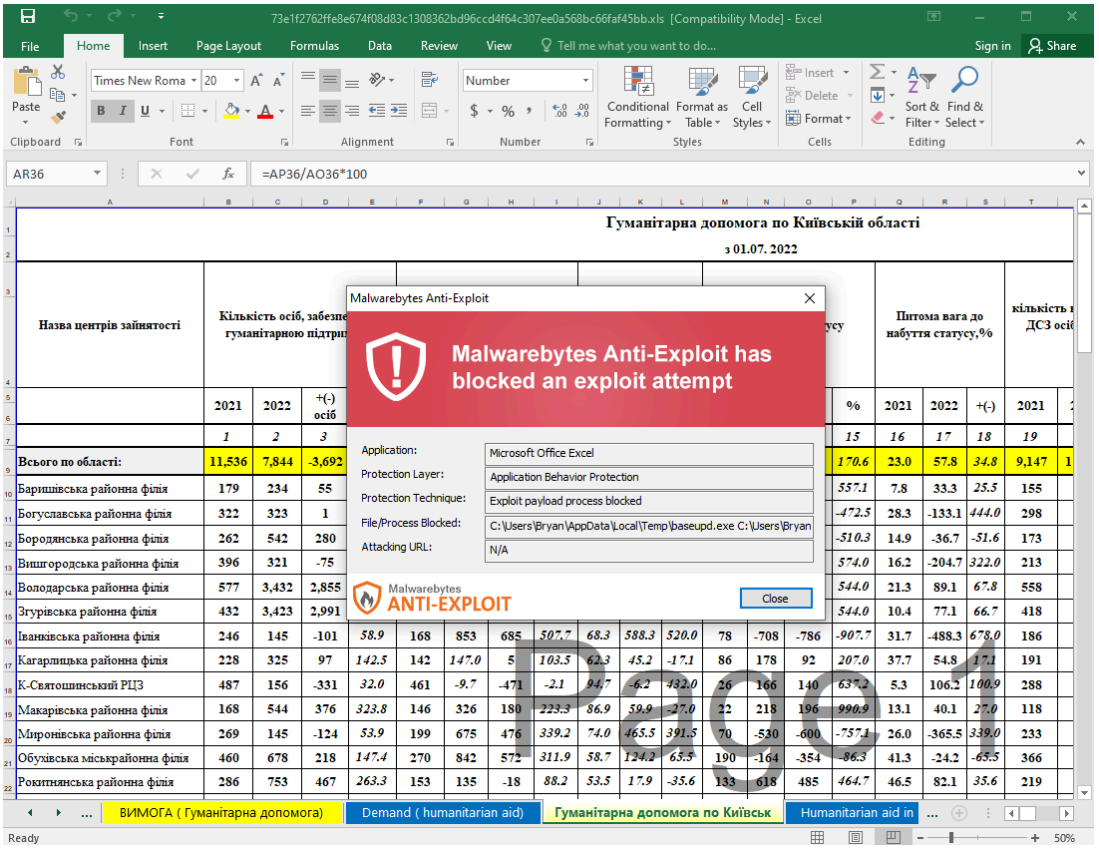
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

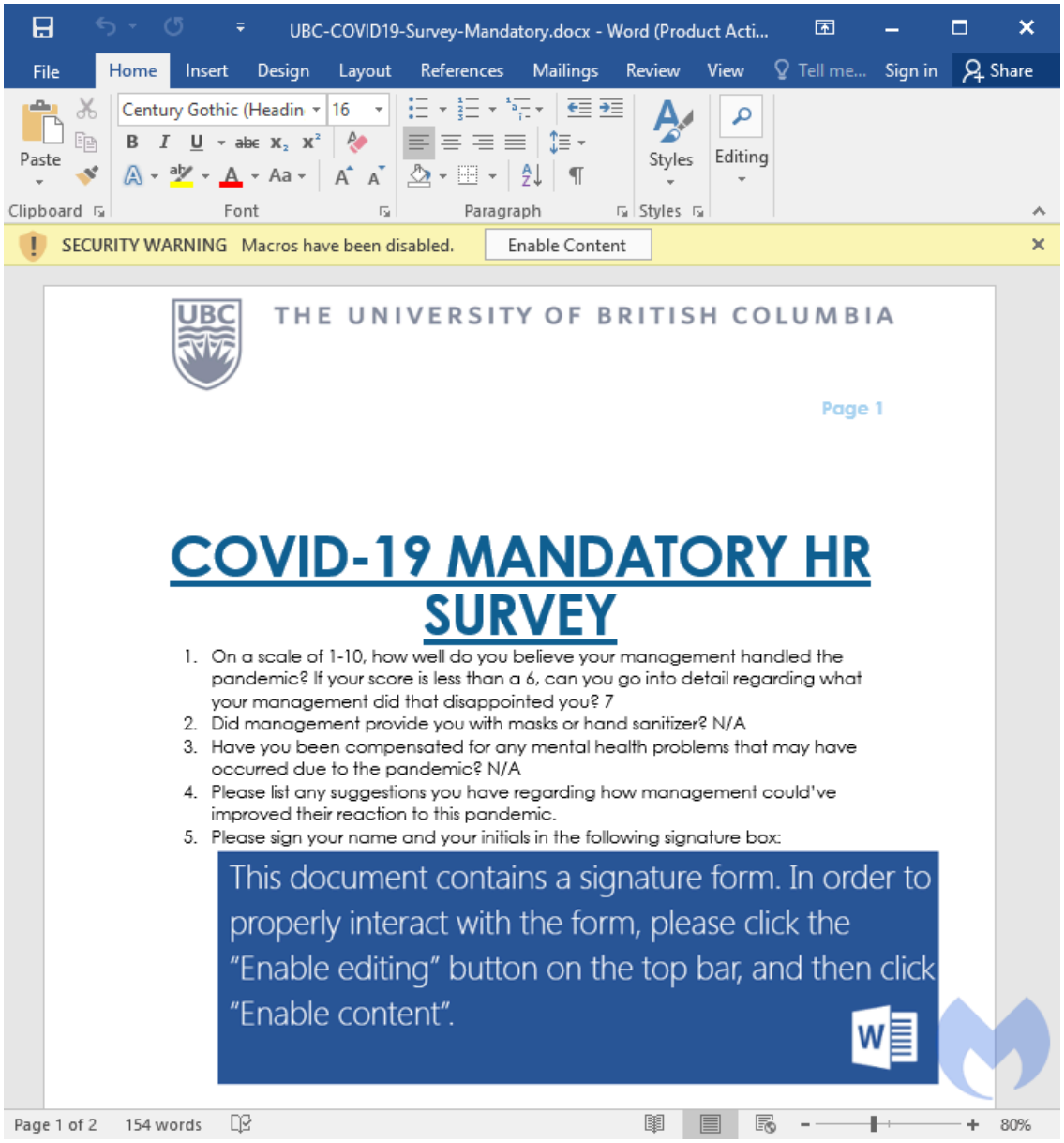
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

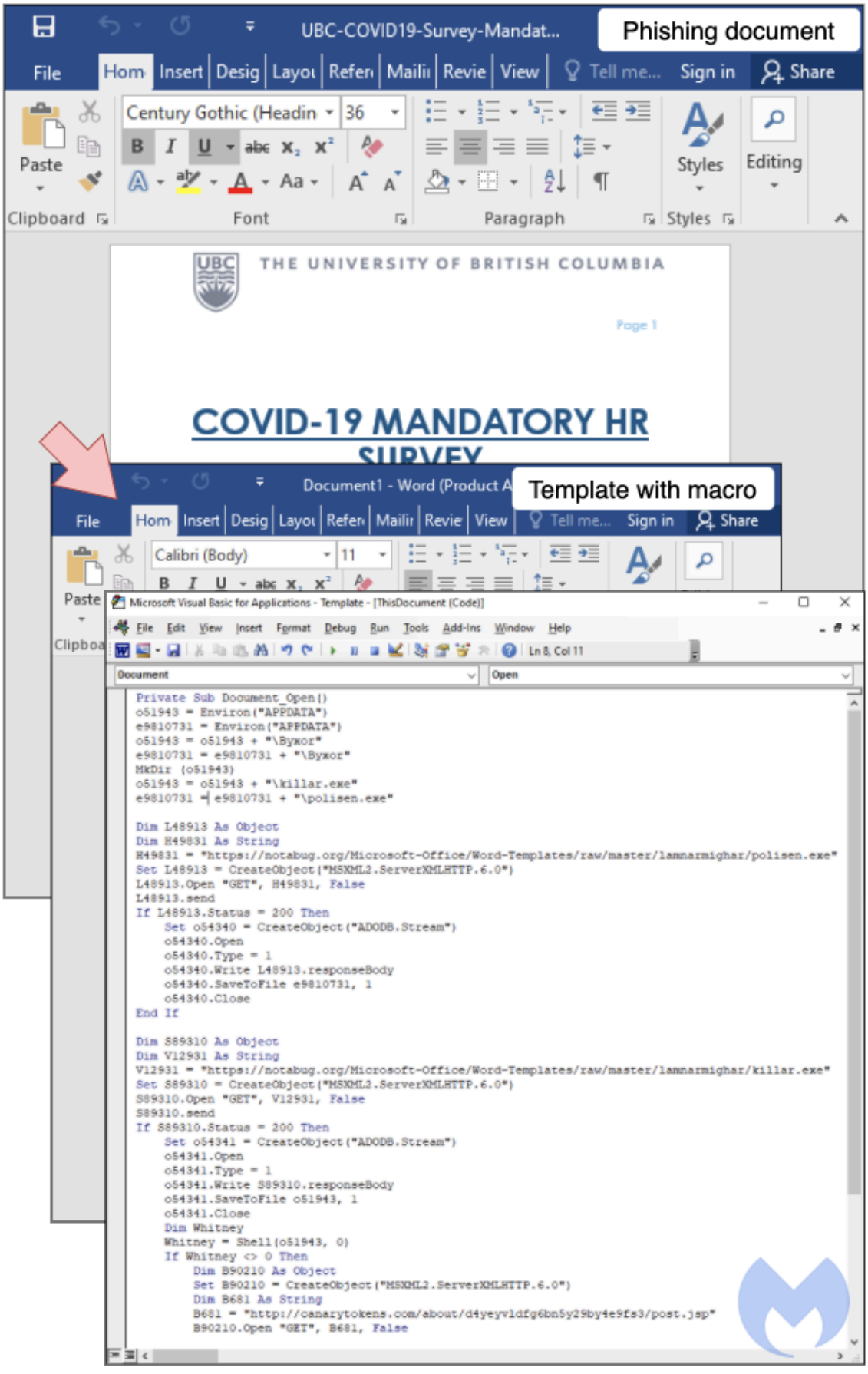
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

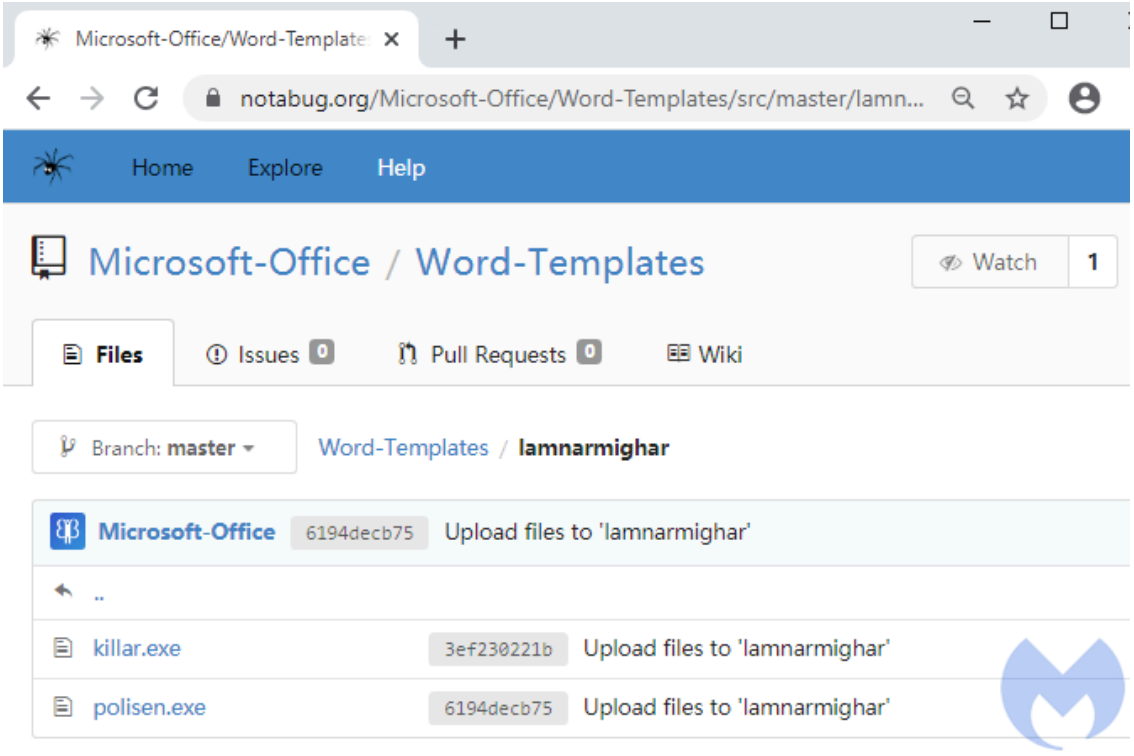
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



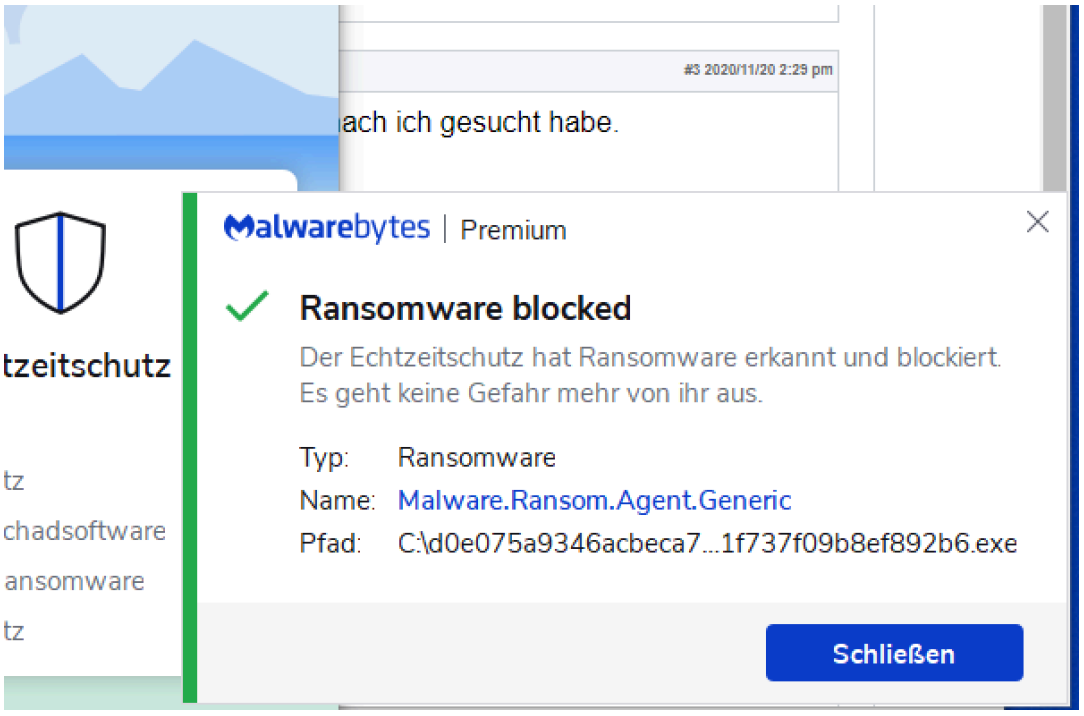
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

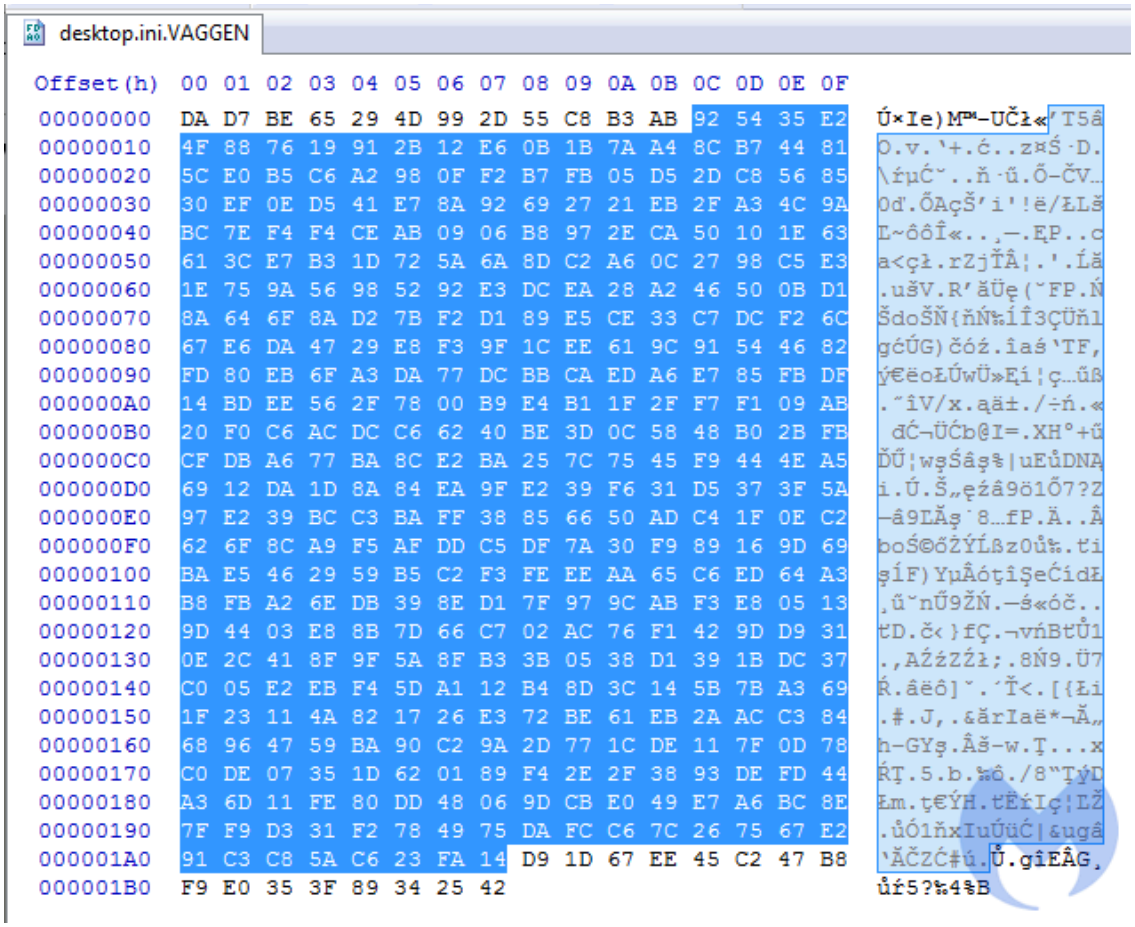
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

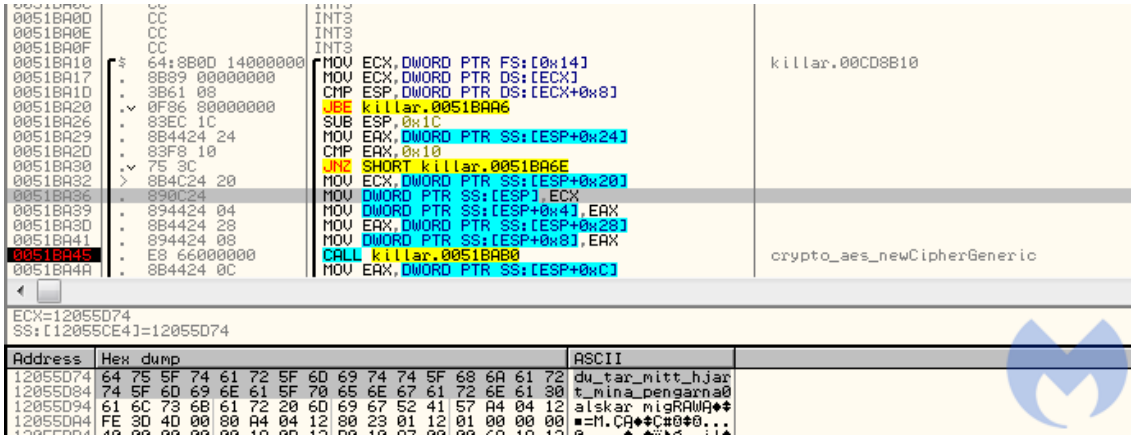


The content of the output file (with .VAGGEN extension) contains:

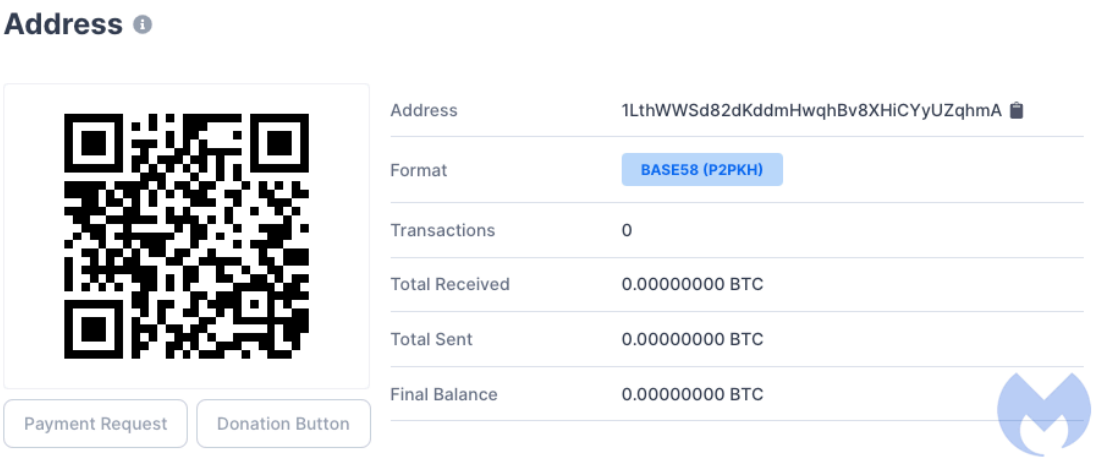
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

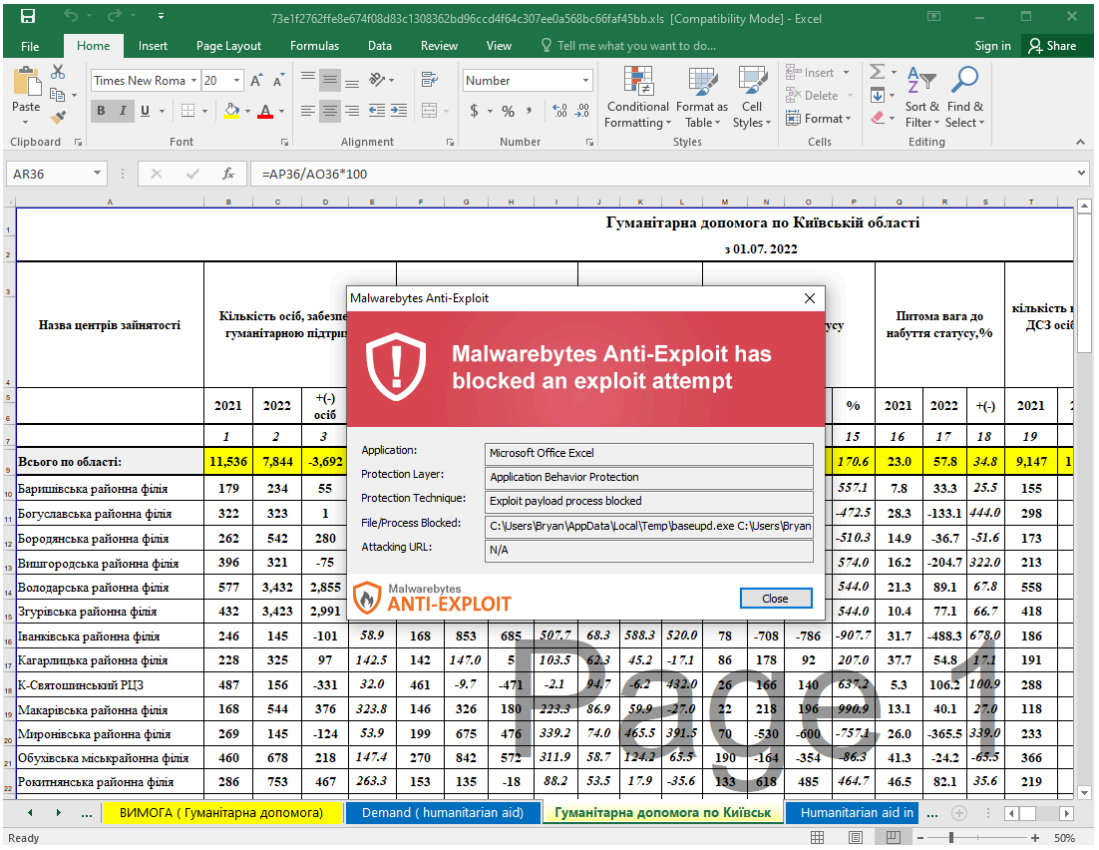
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

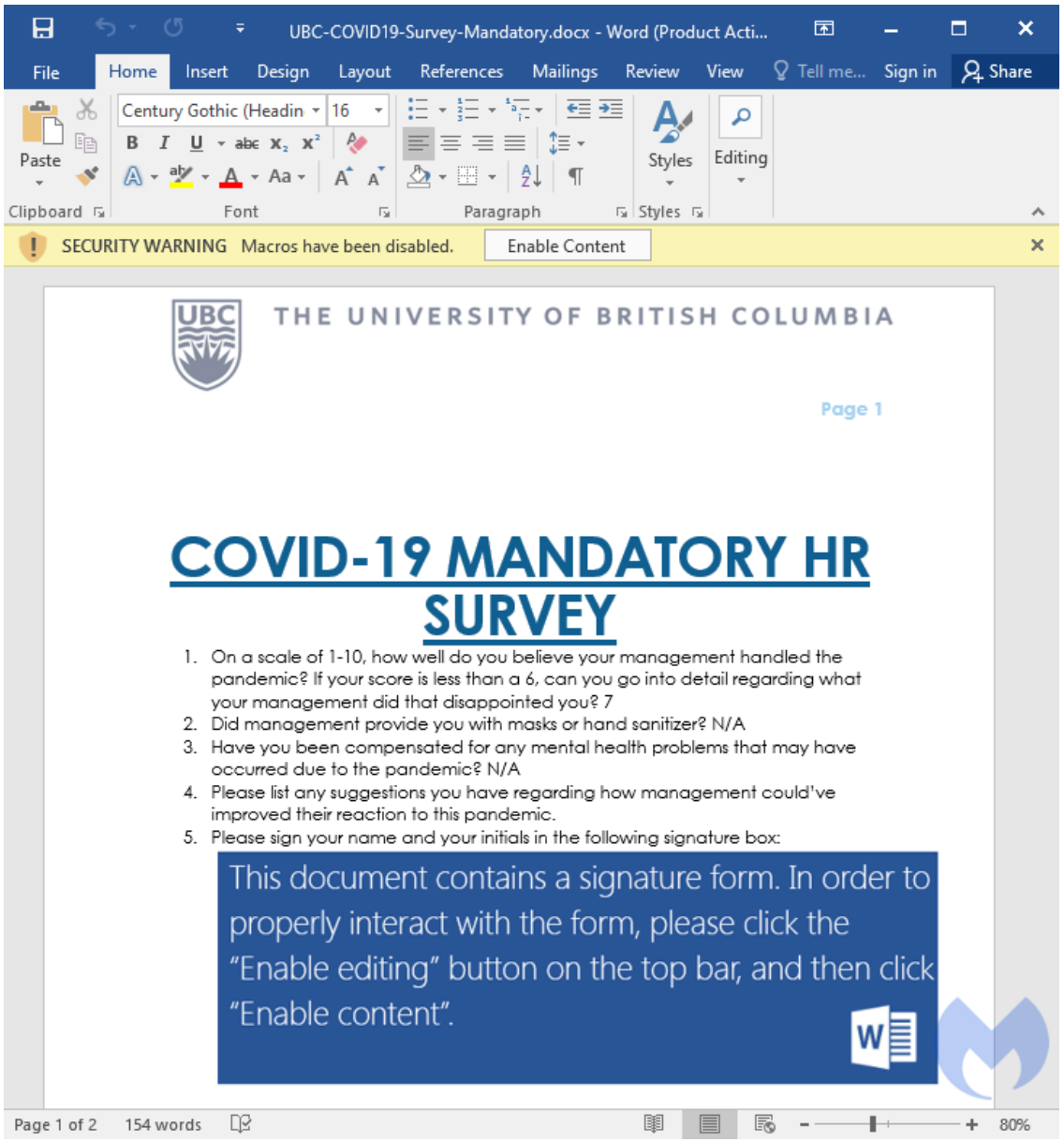
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

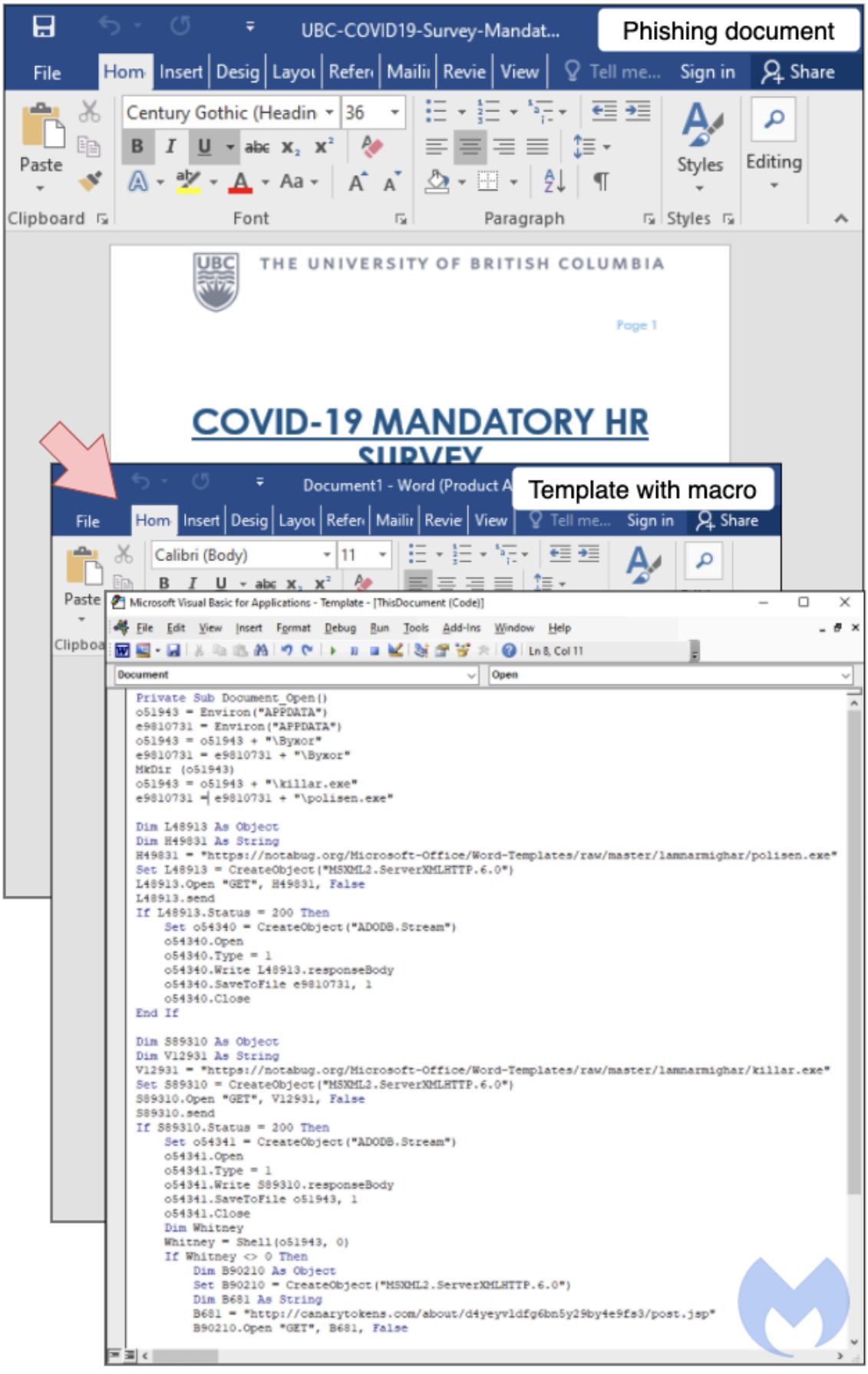
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

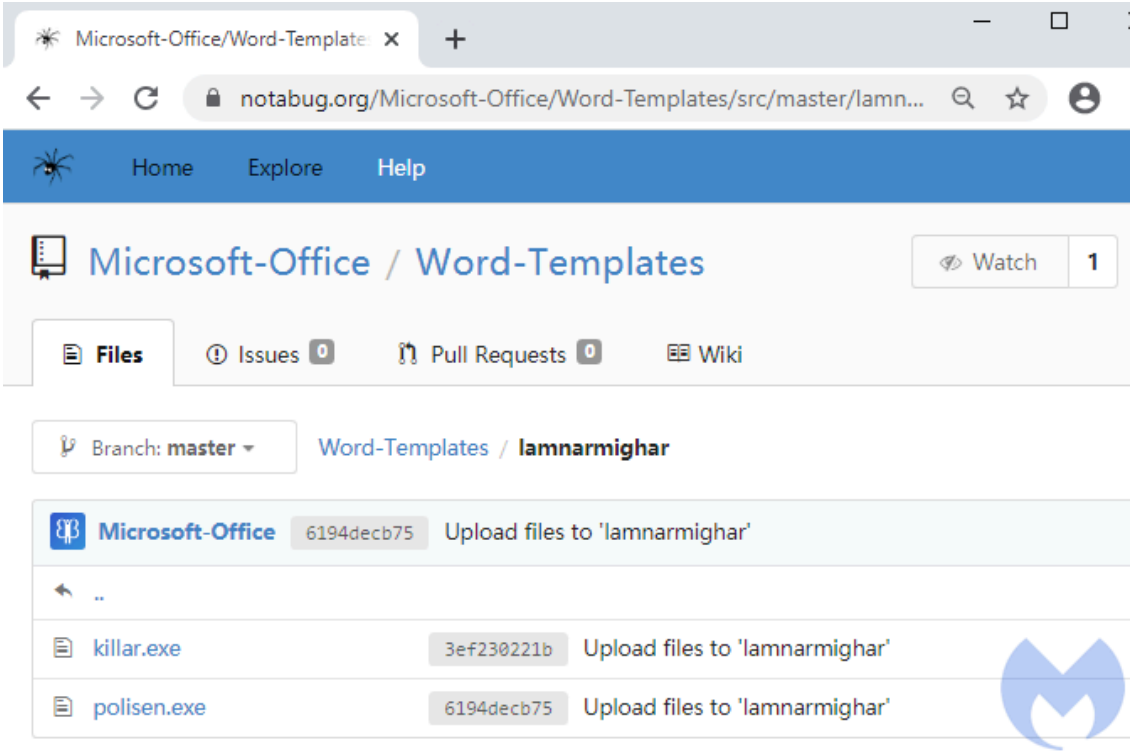
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



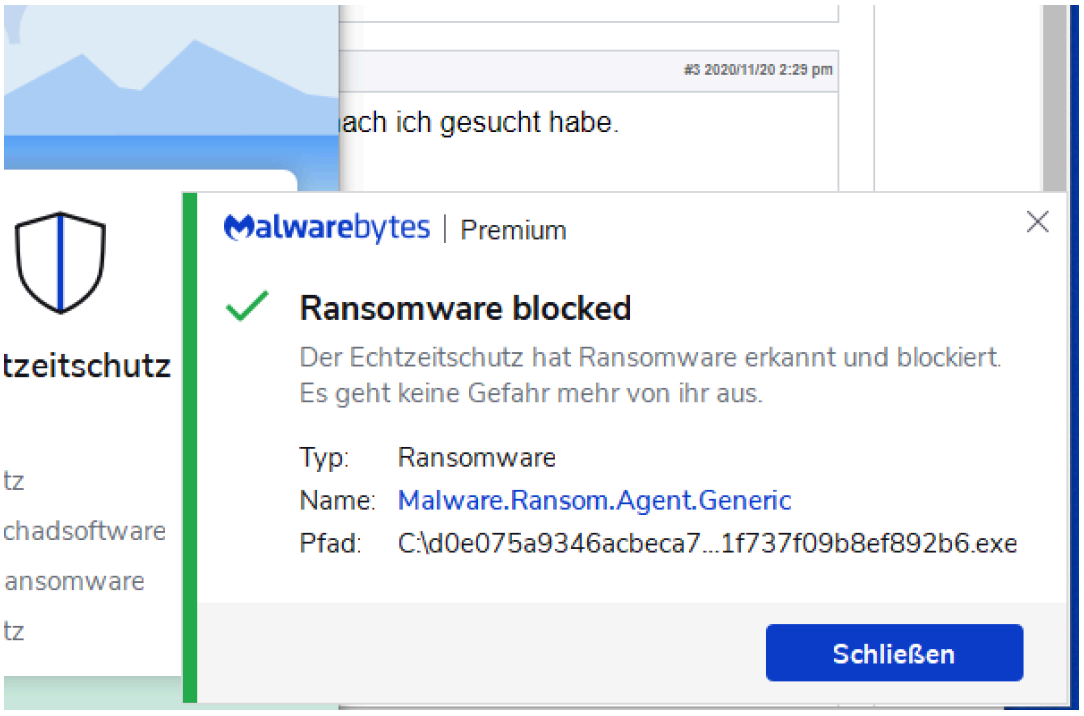
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

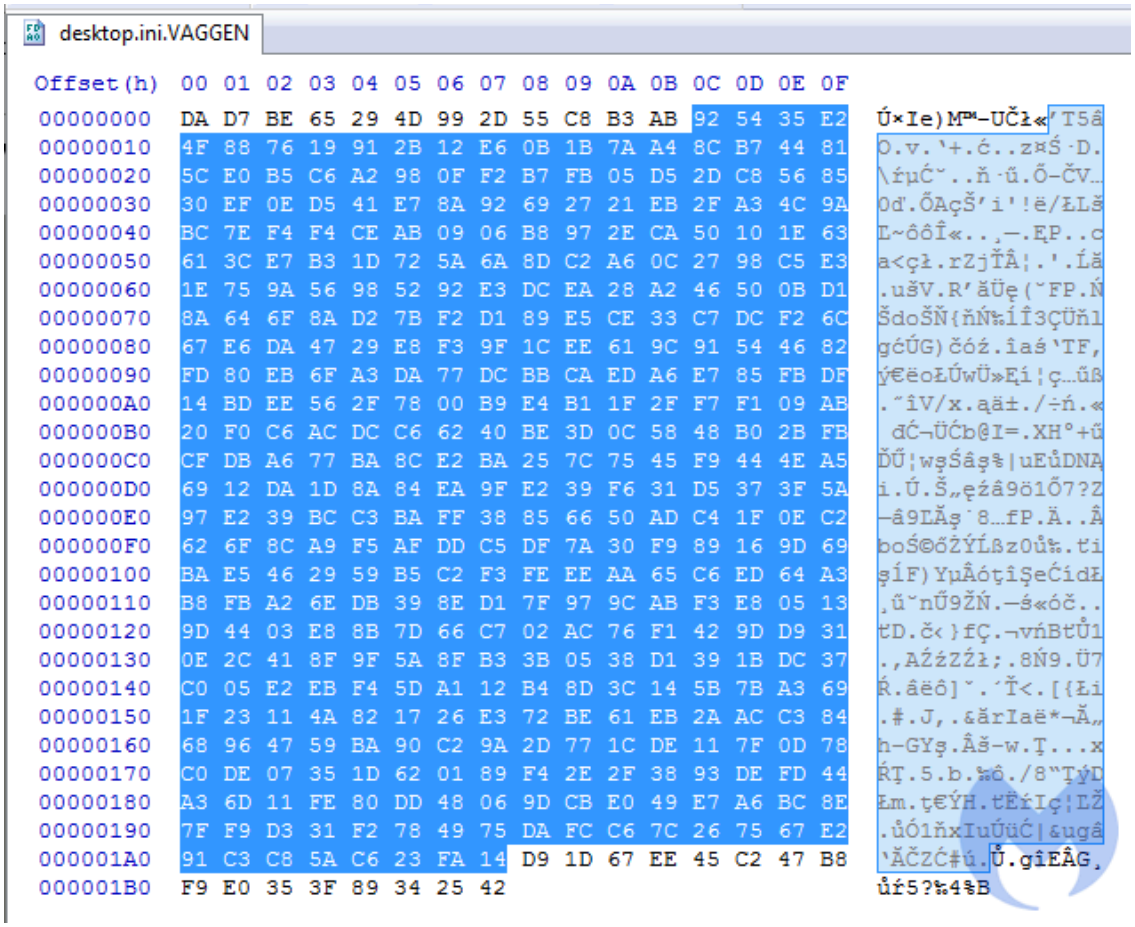
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



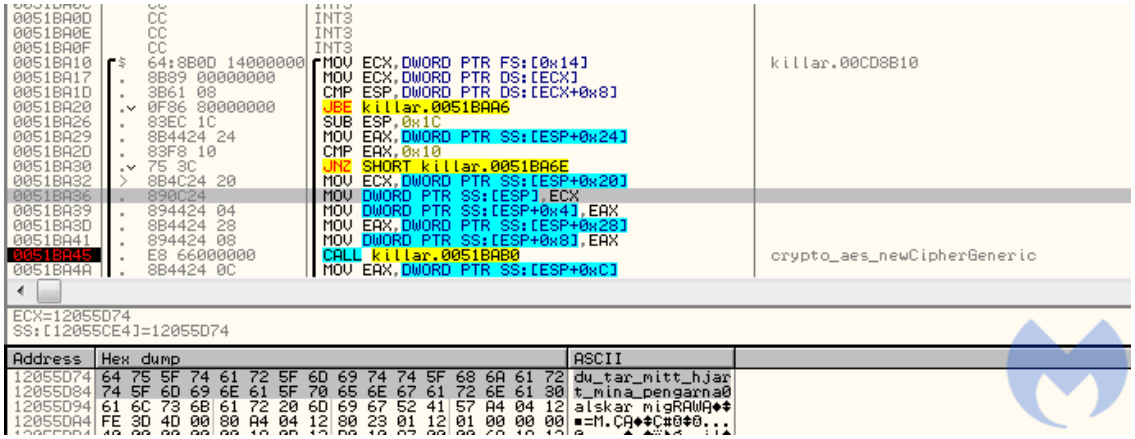
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

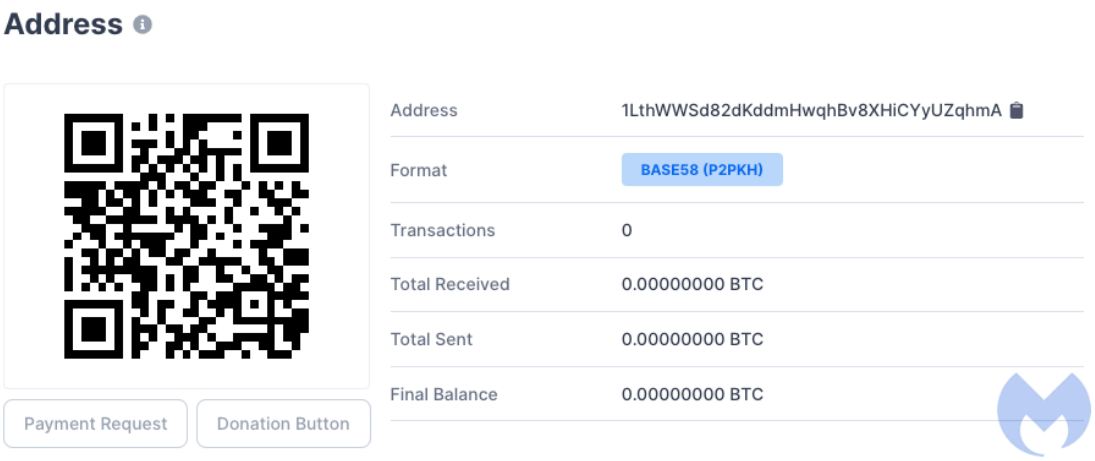
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

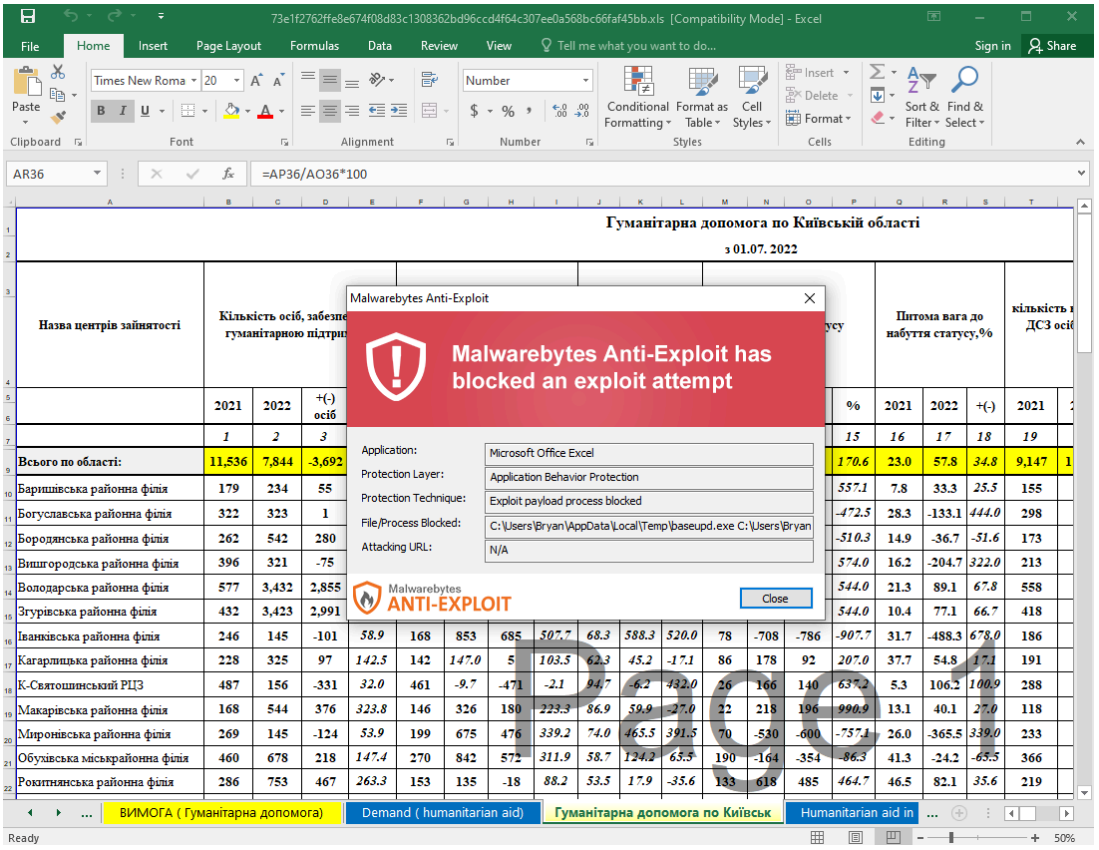
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

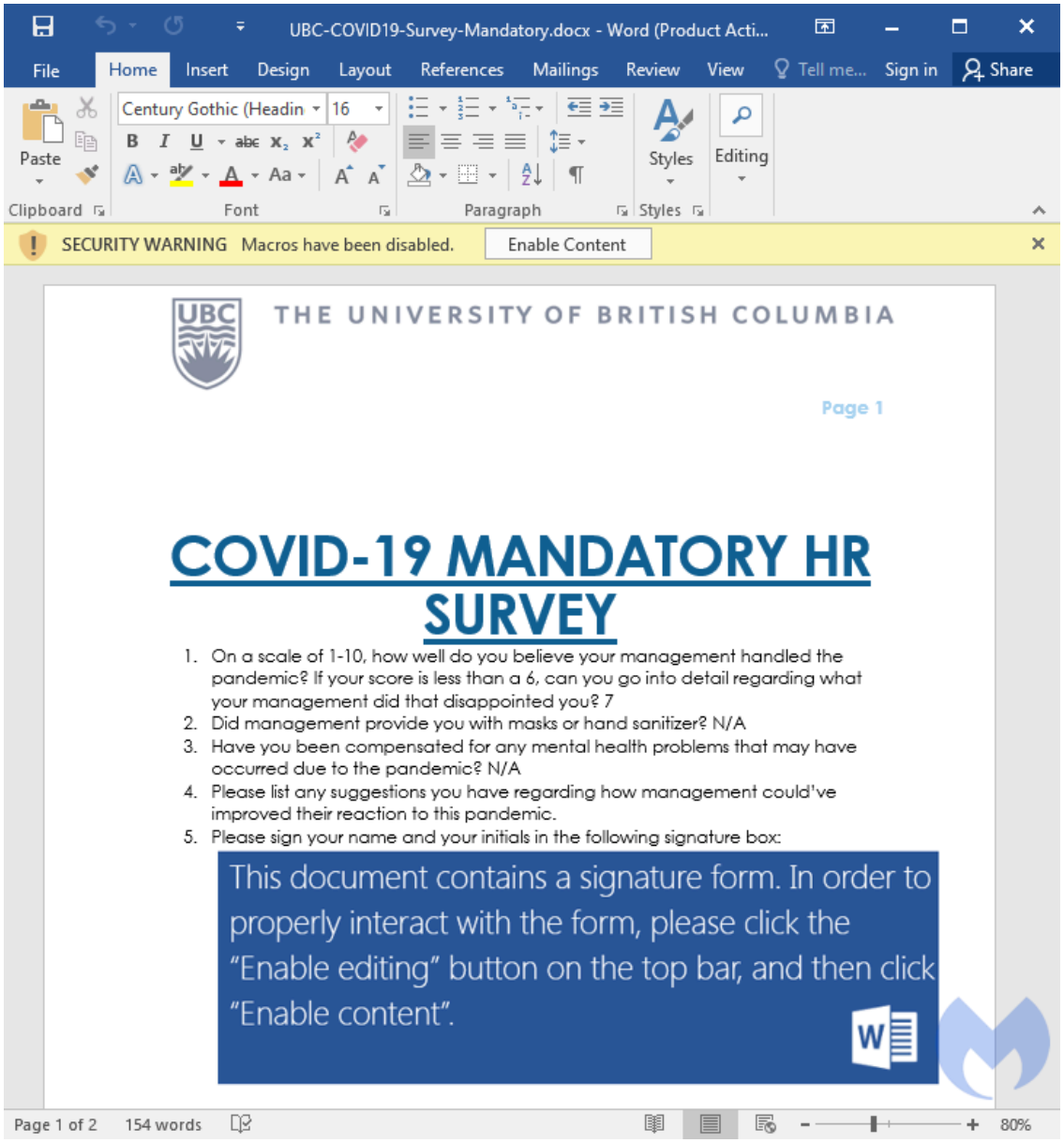
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

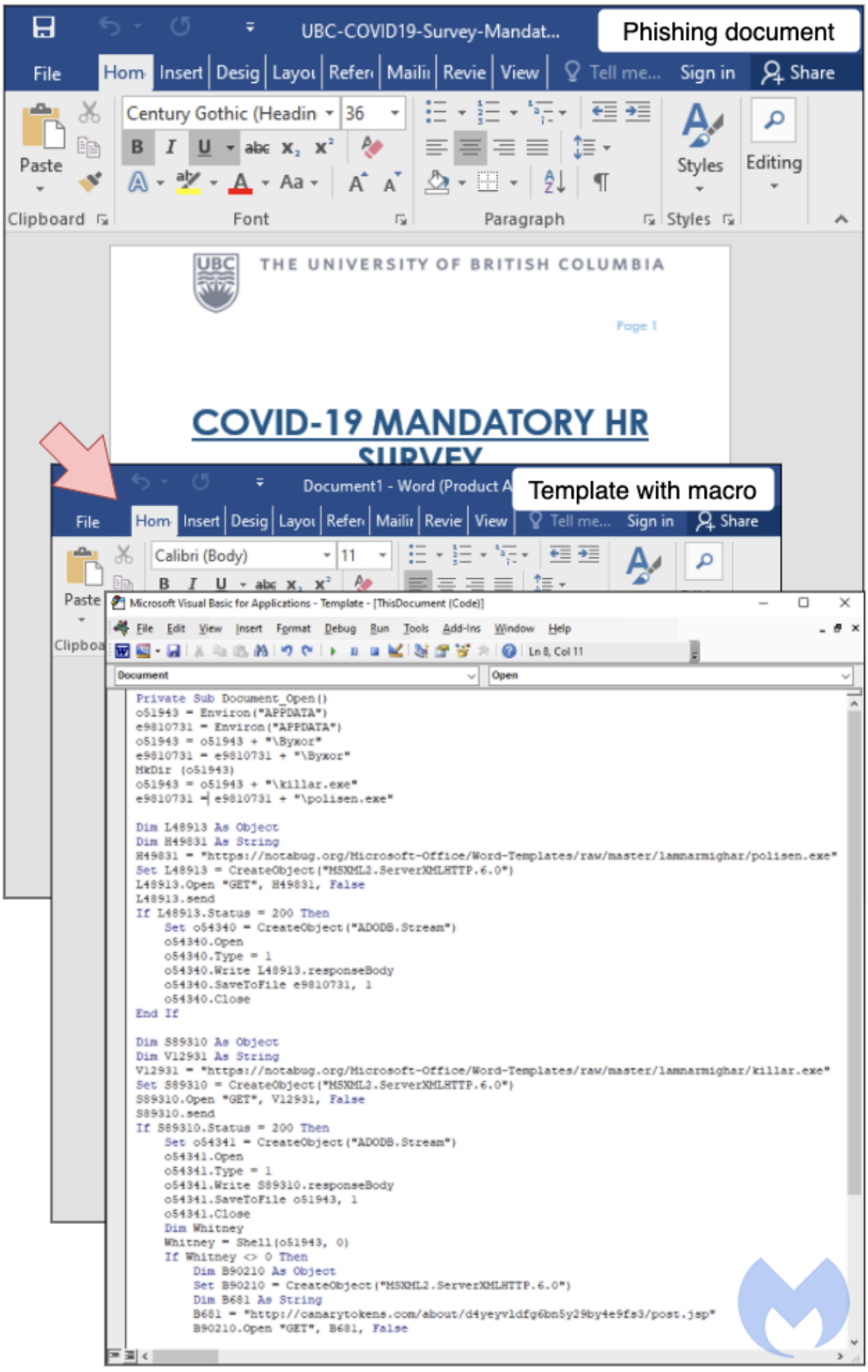
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

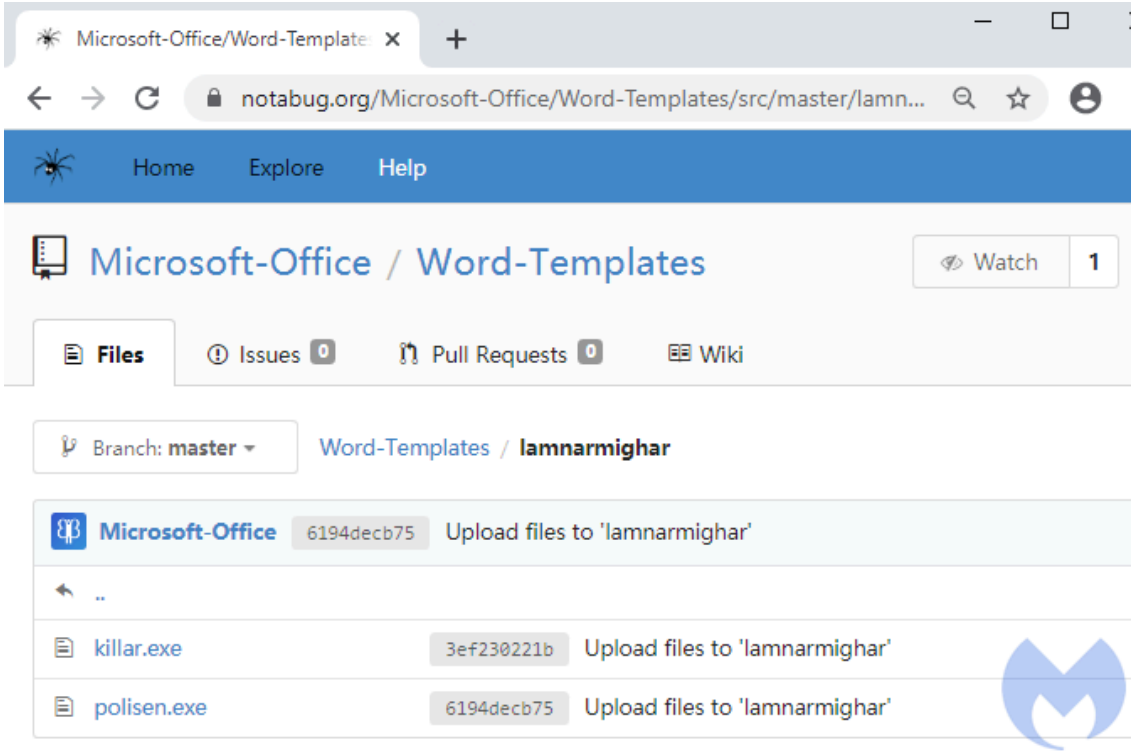
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



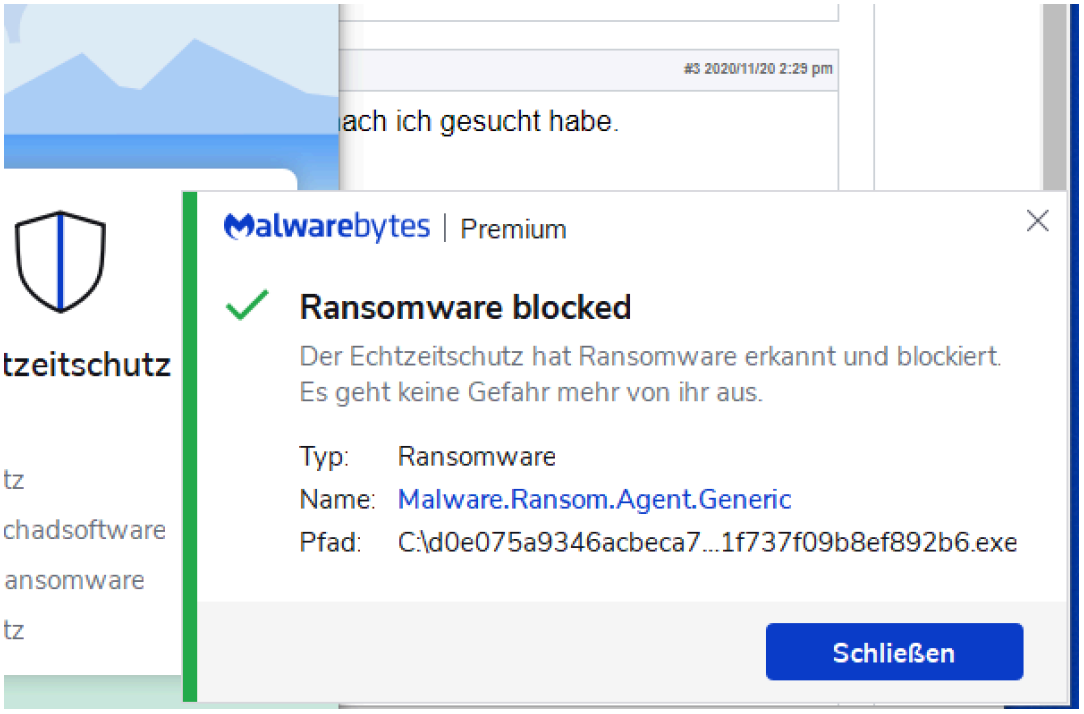
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

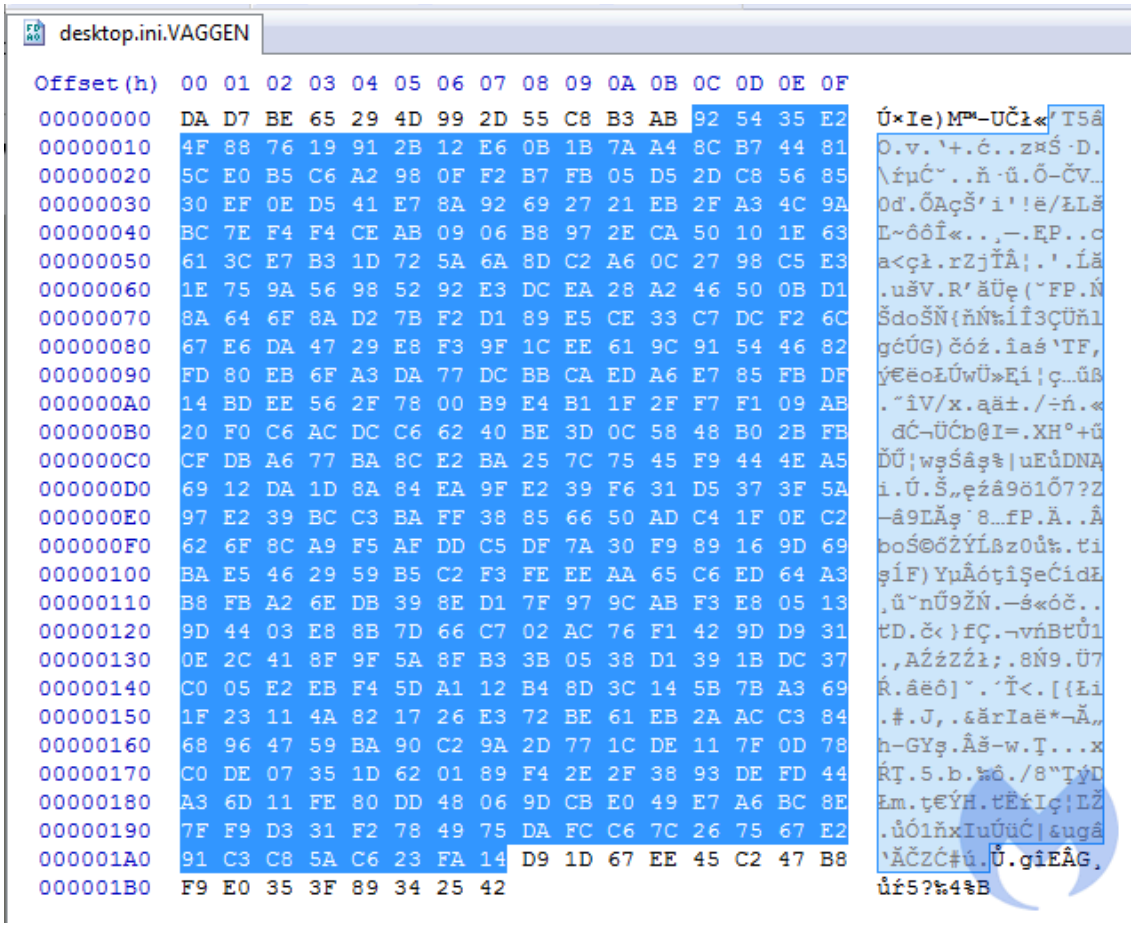
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

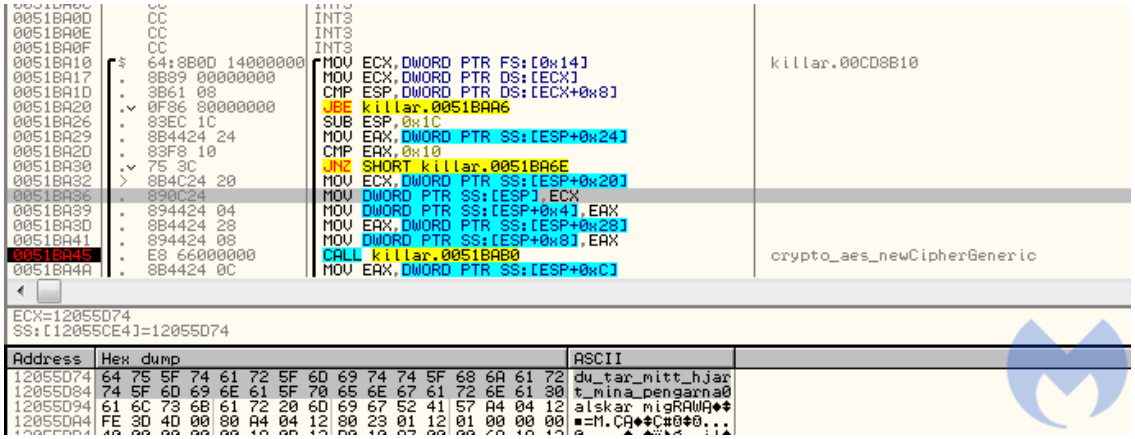


The content of the output file (with .VAGGEN extension) contains:

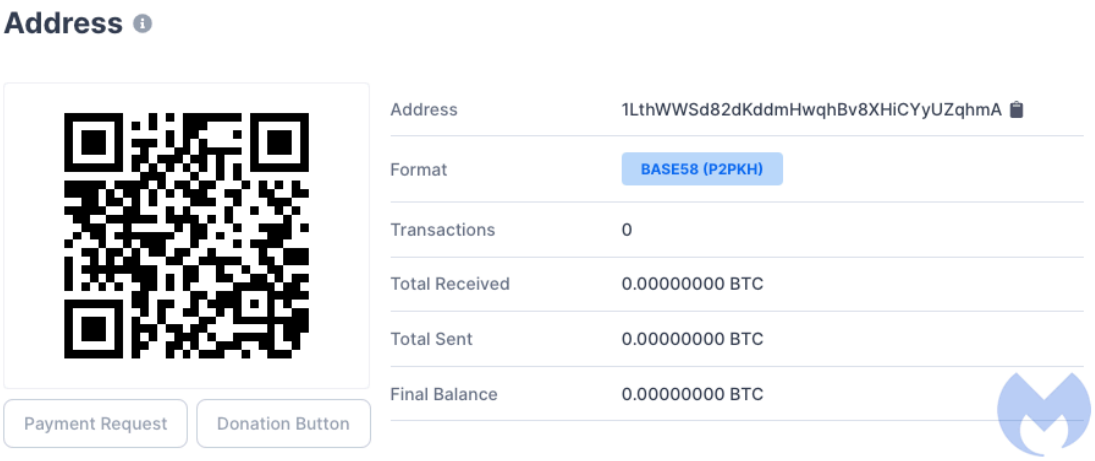
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

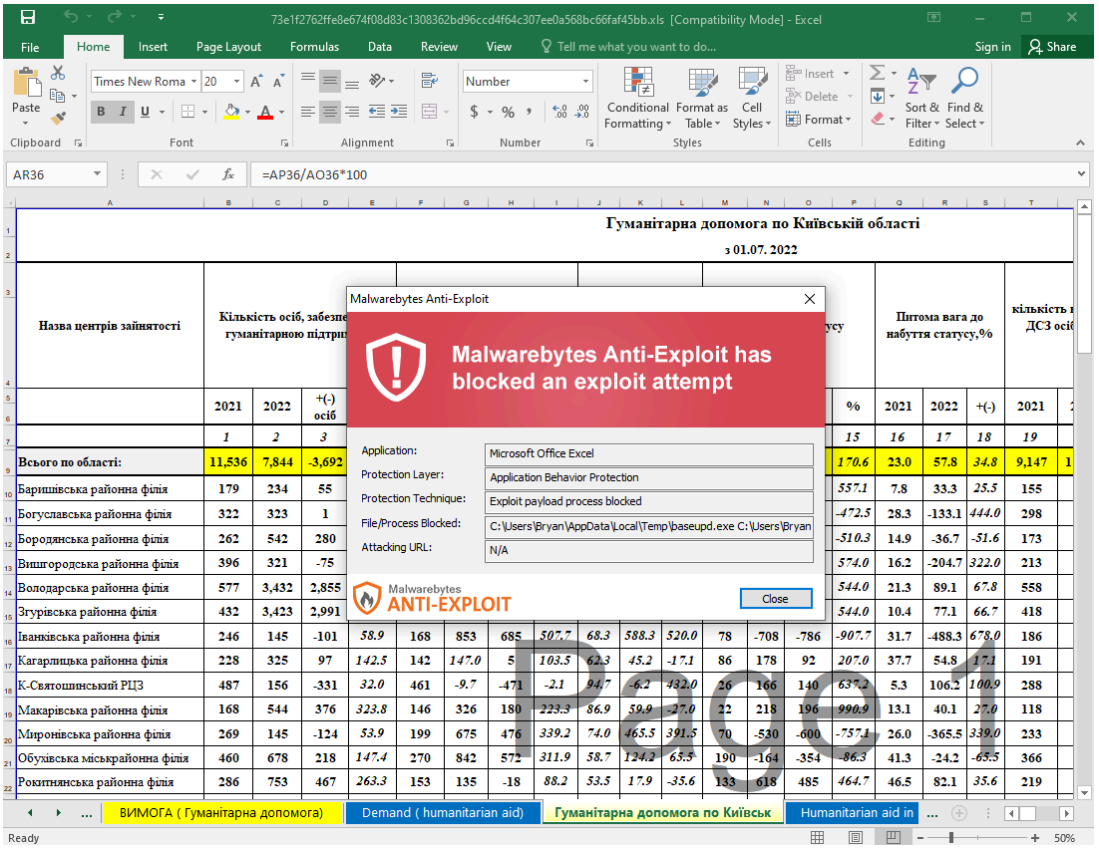
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

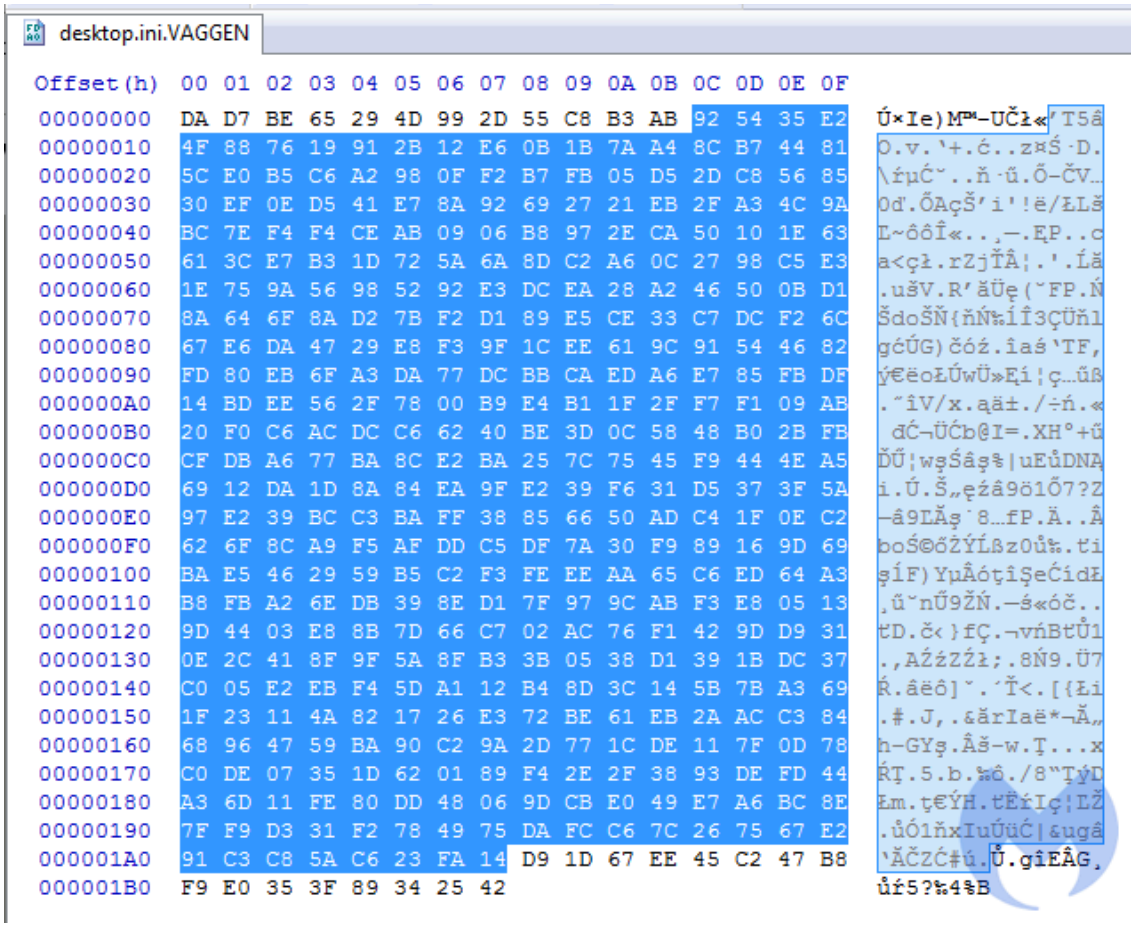
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

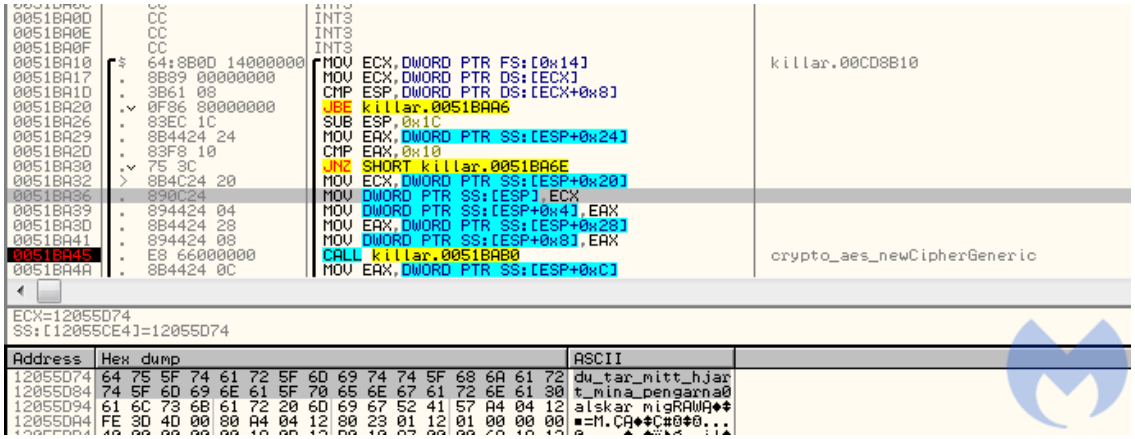
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

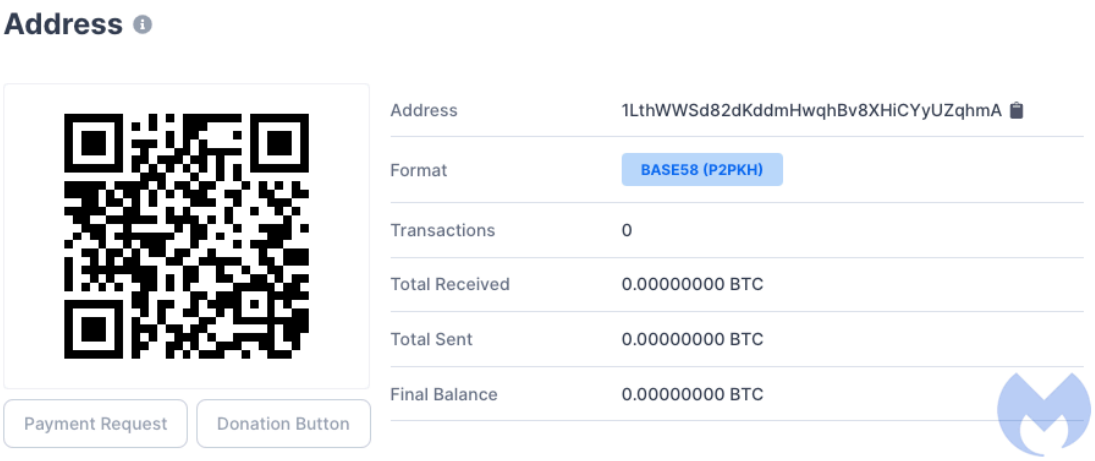
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

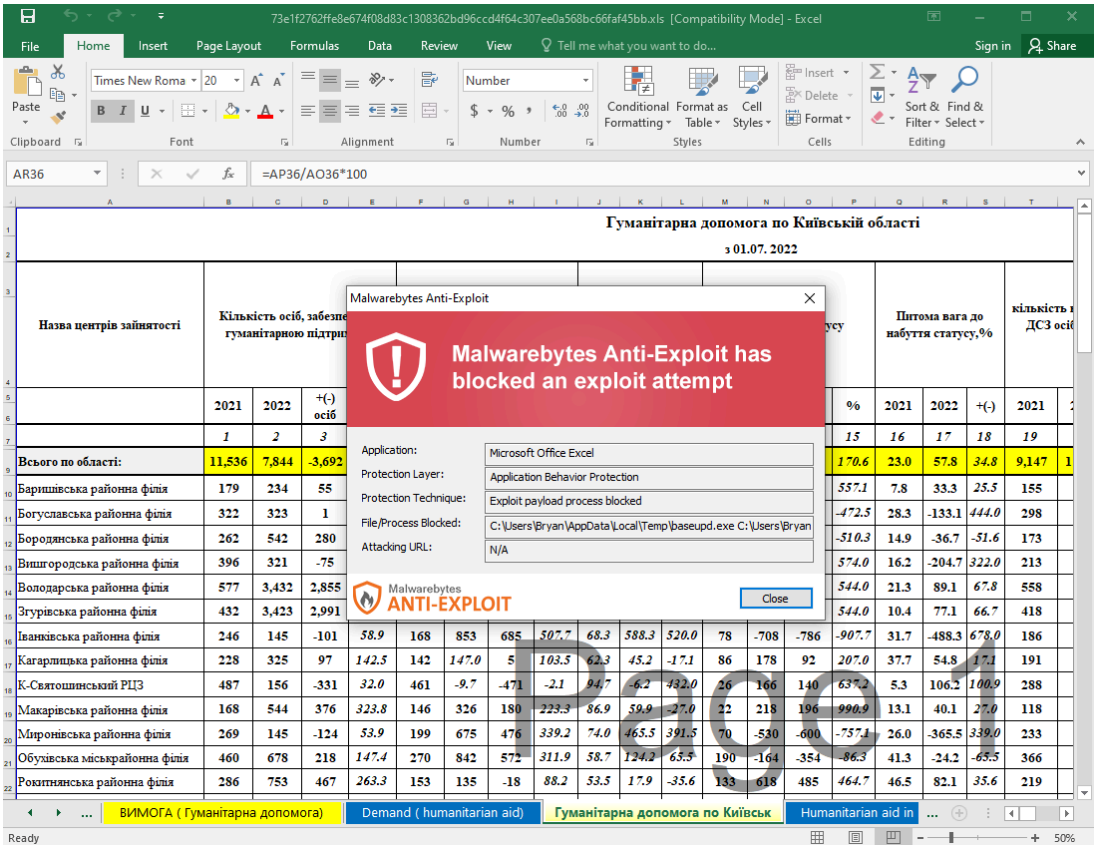
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

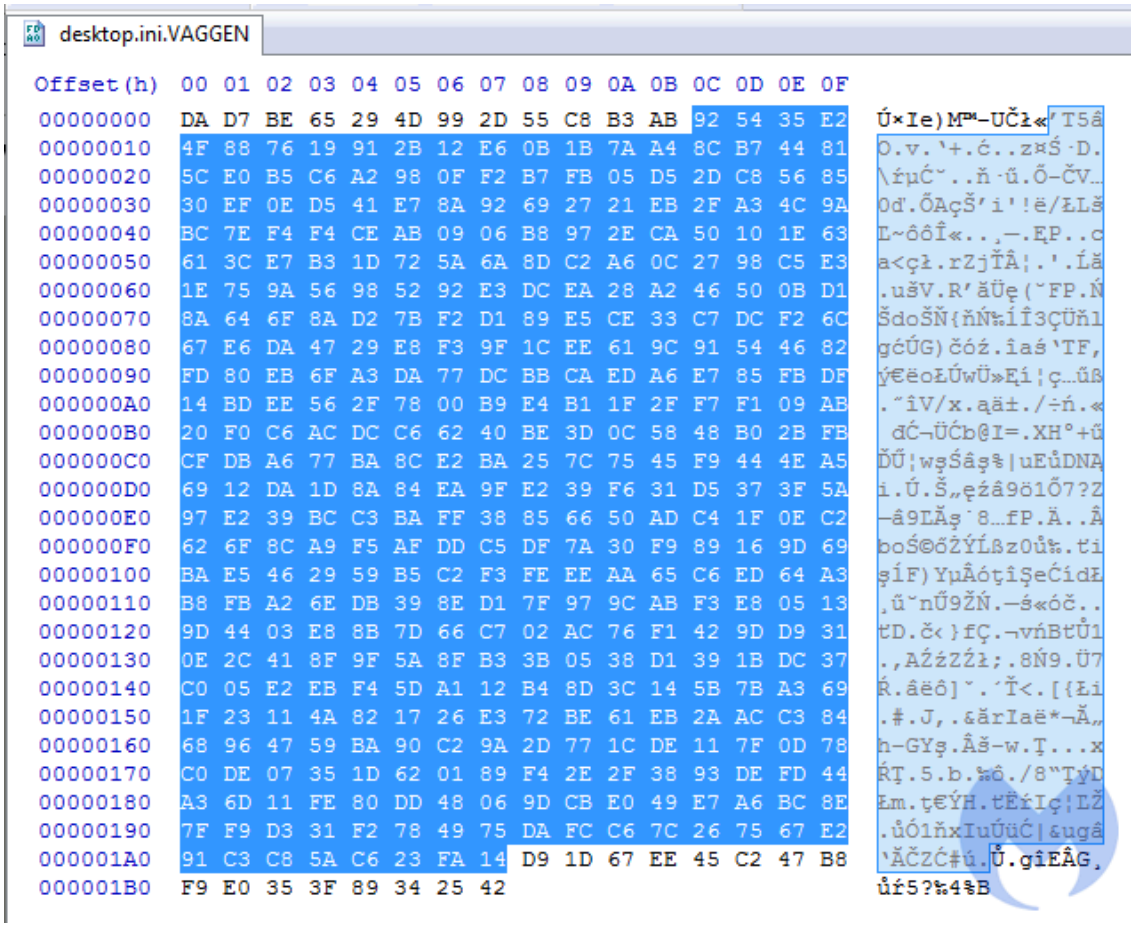
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

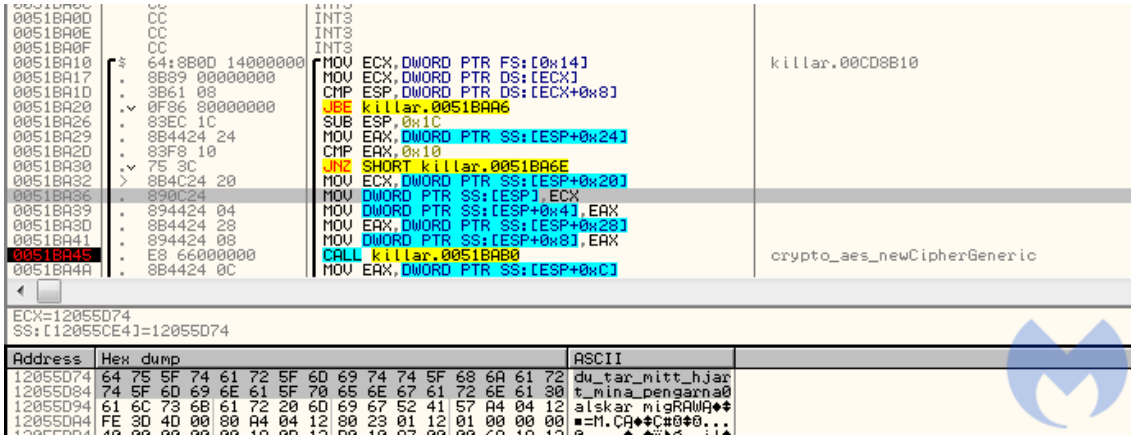


The content of the output file (with .VAGGEN extension) contains:

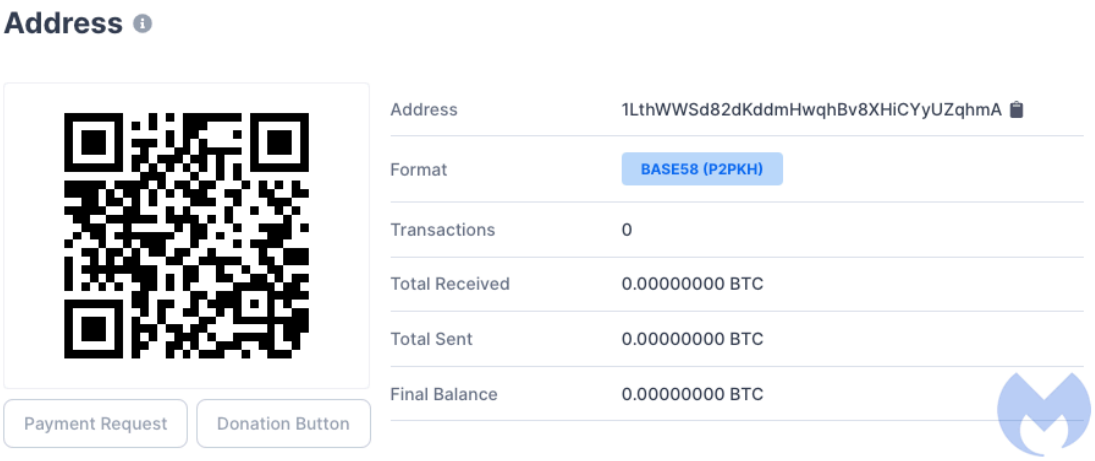
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

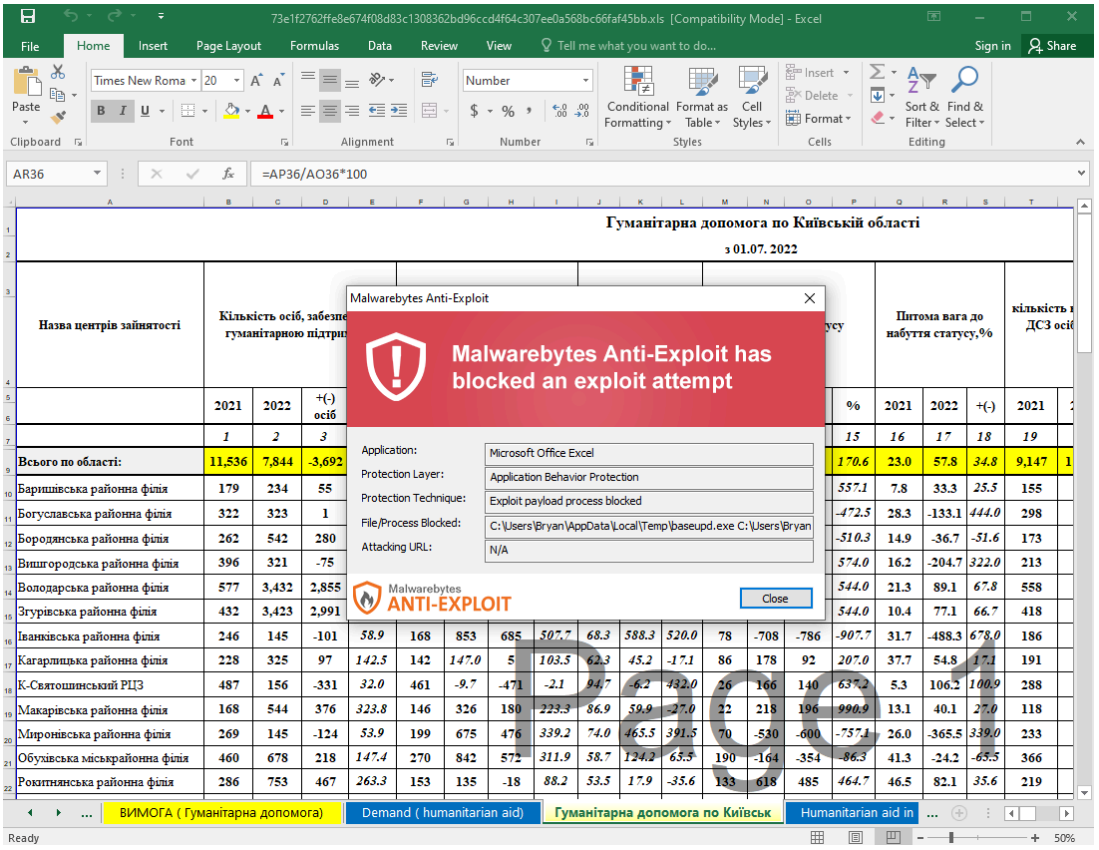
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

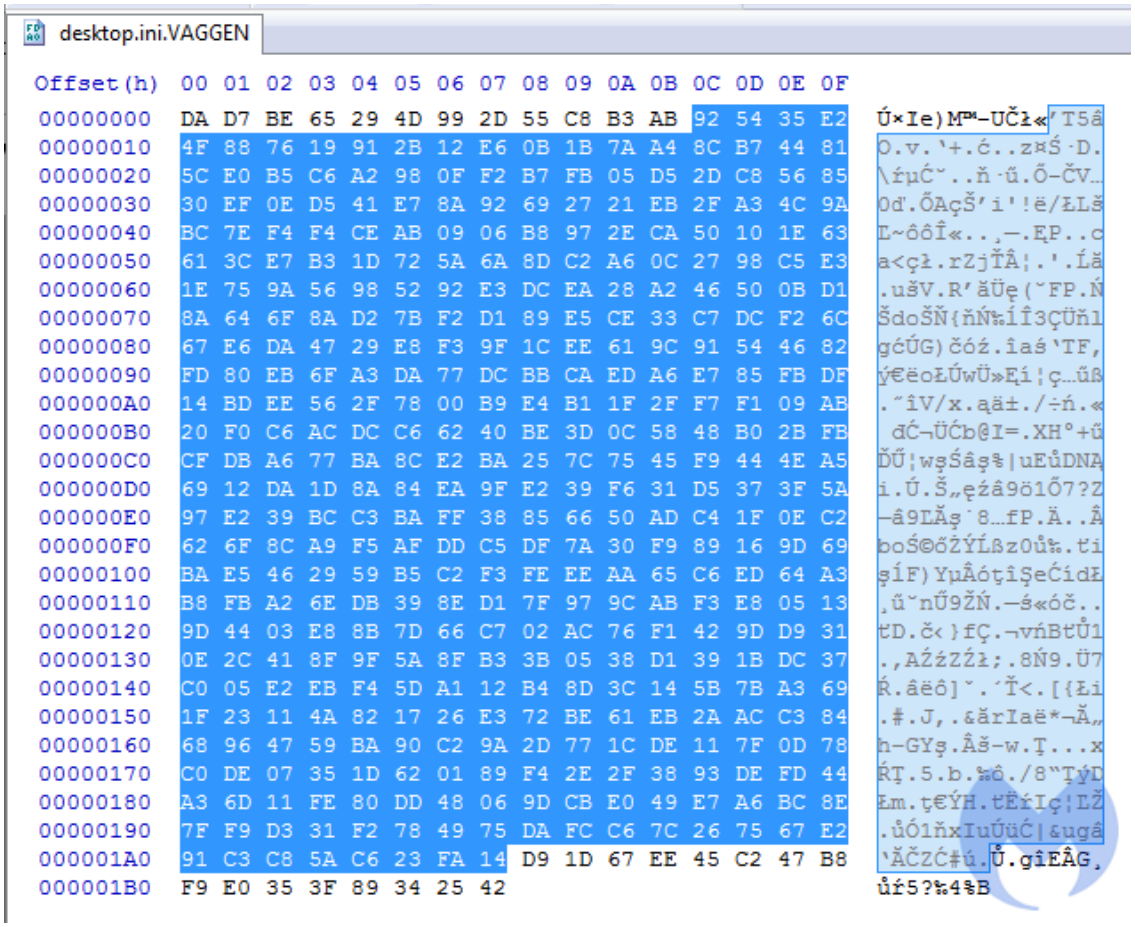
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

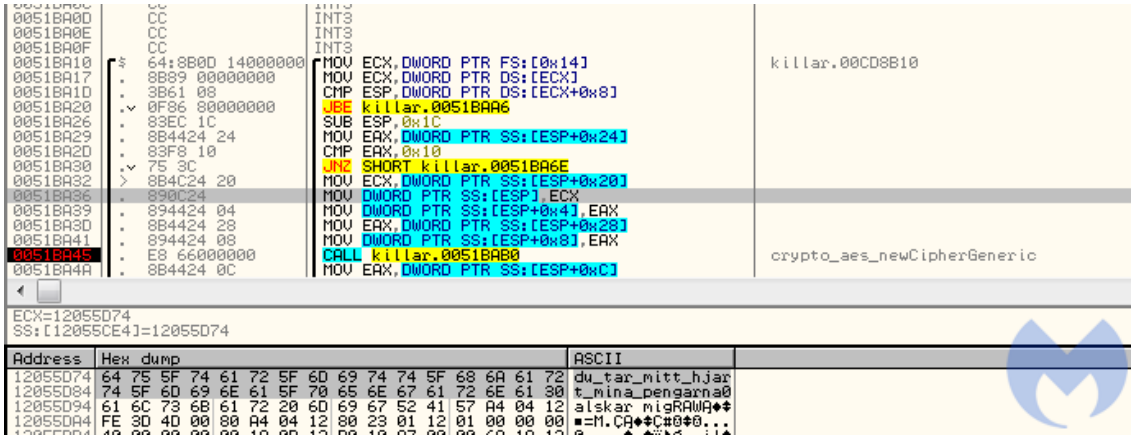


The content of the output file (with .VAGGEN extension) contains:

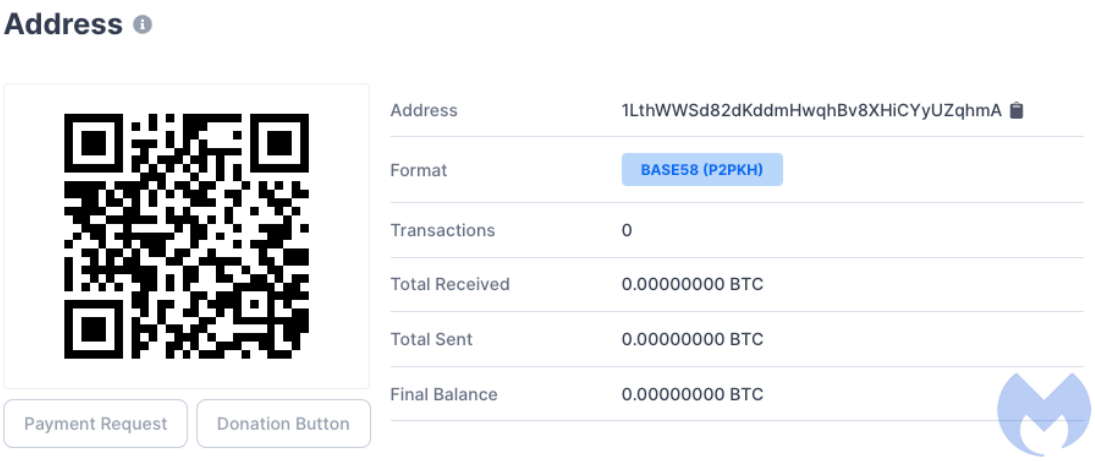
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

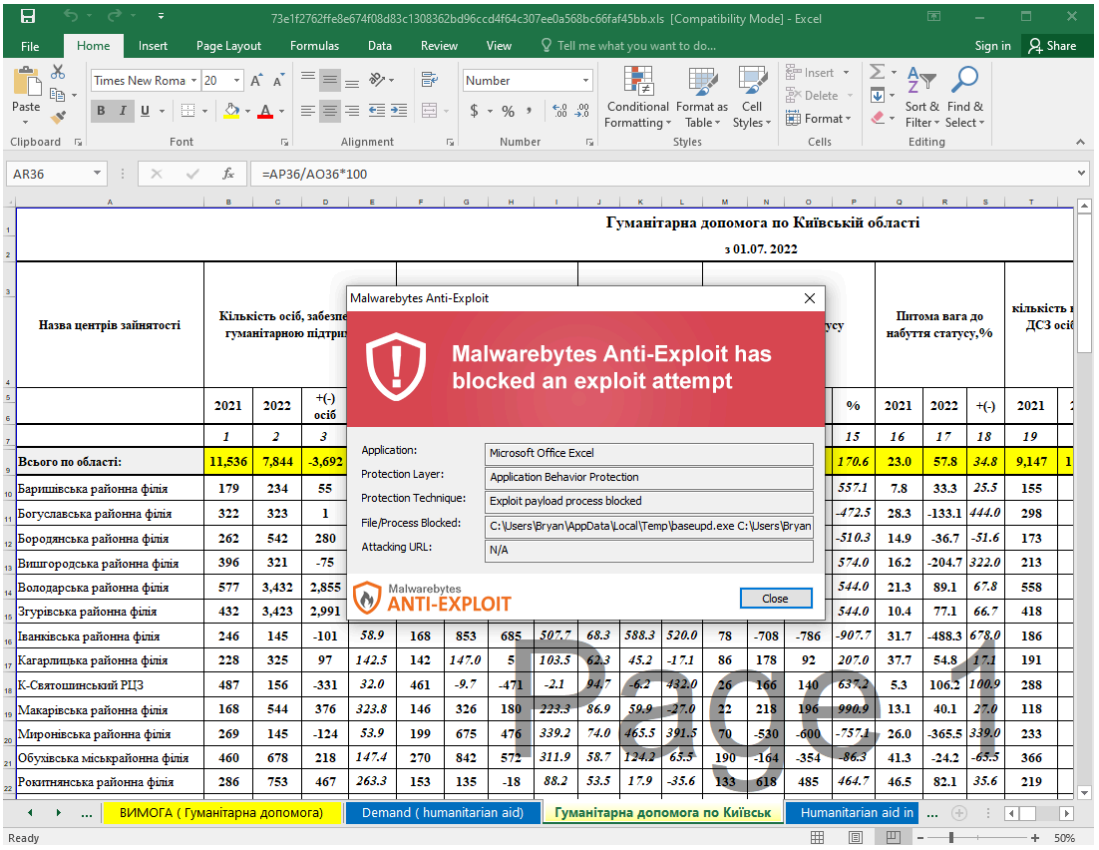
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

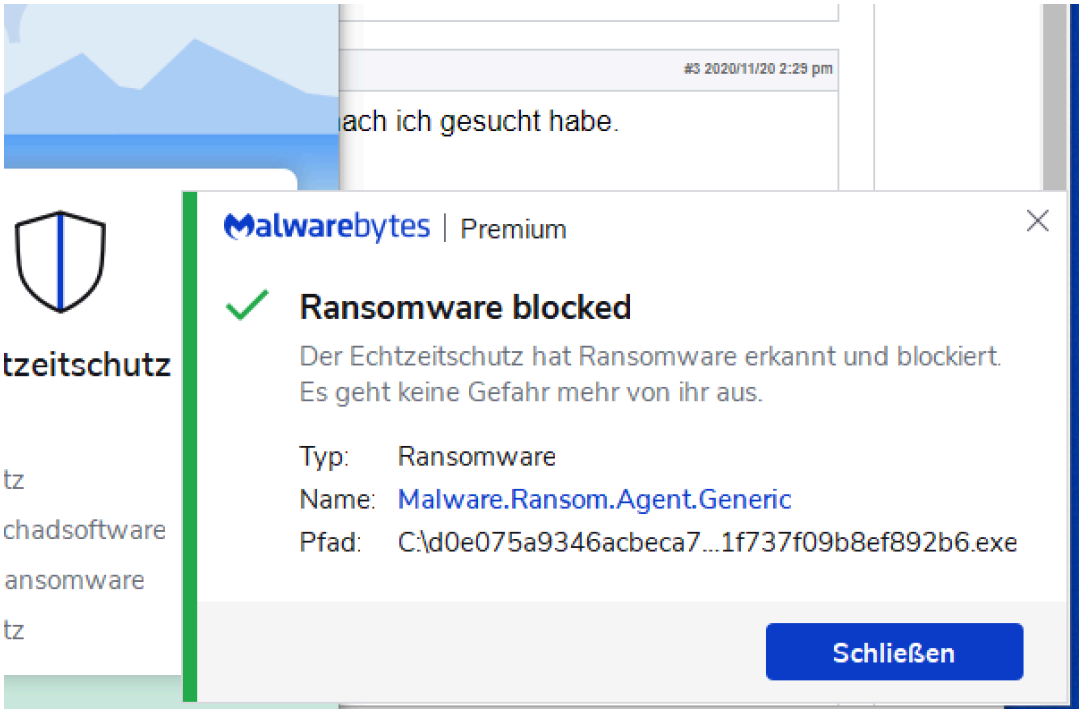
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

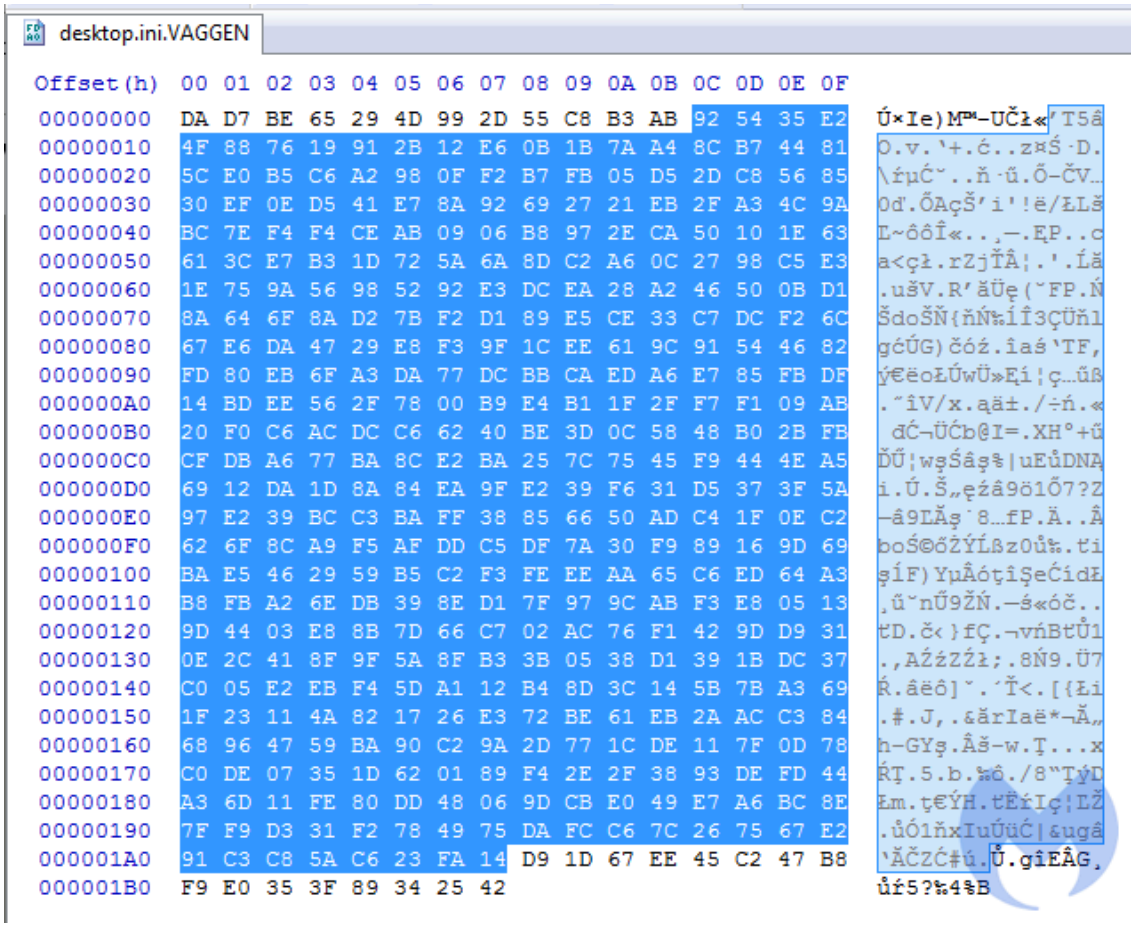
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



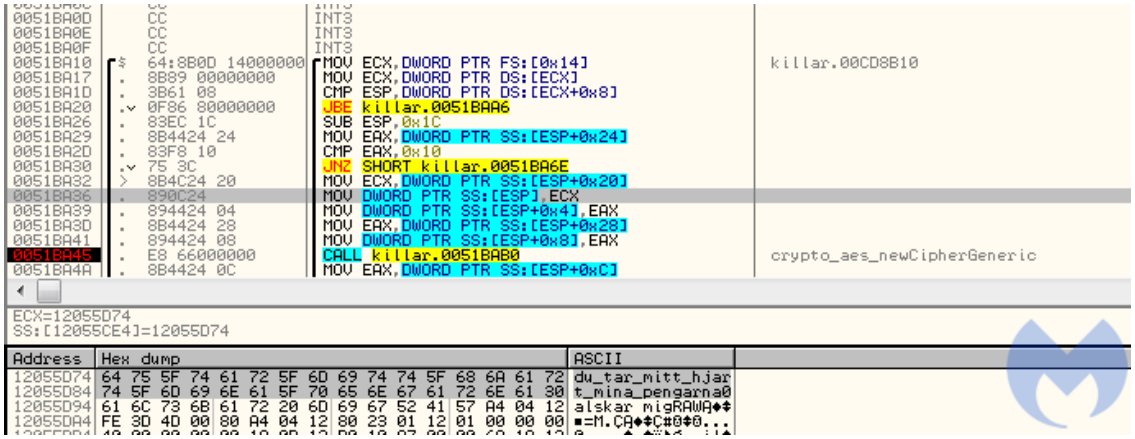
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

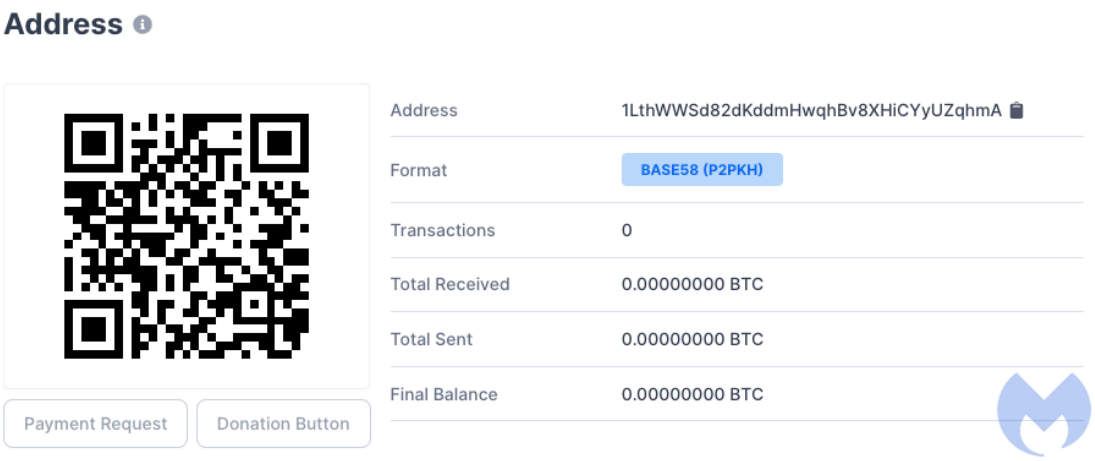
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

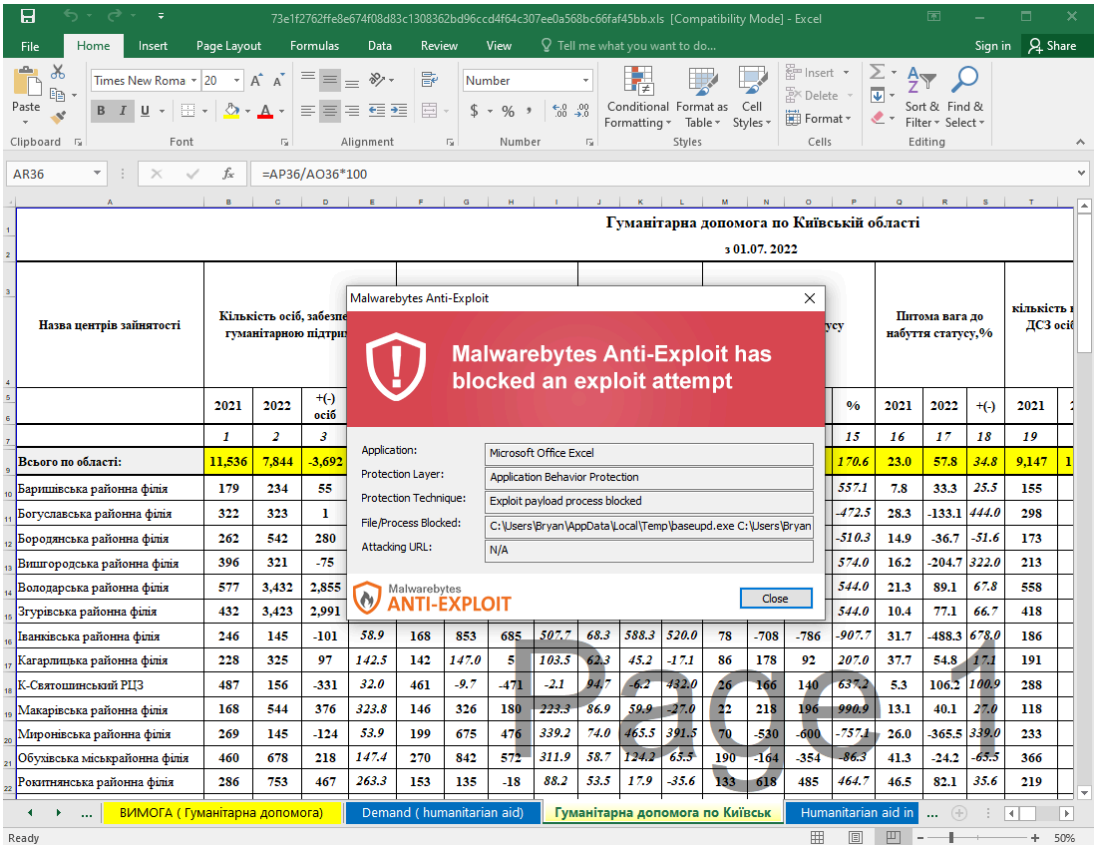
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

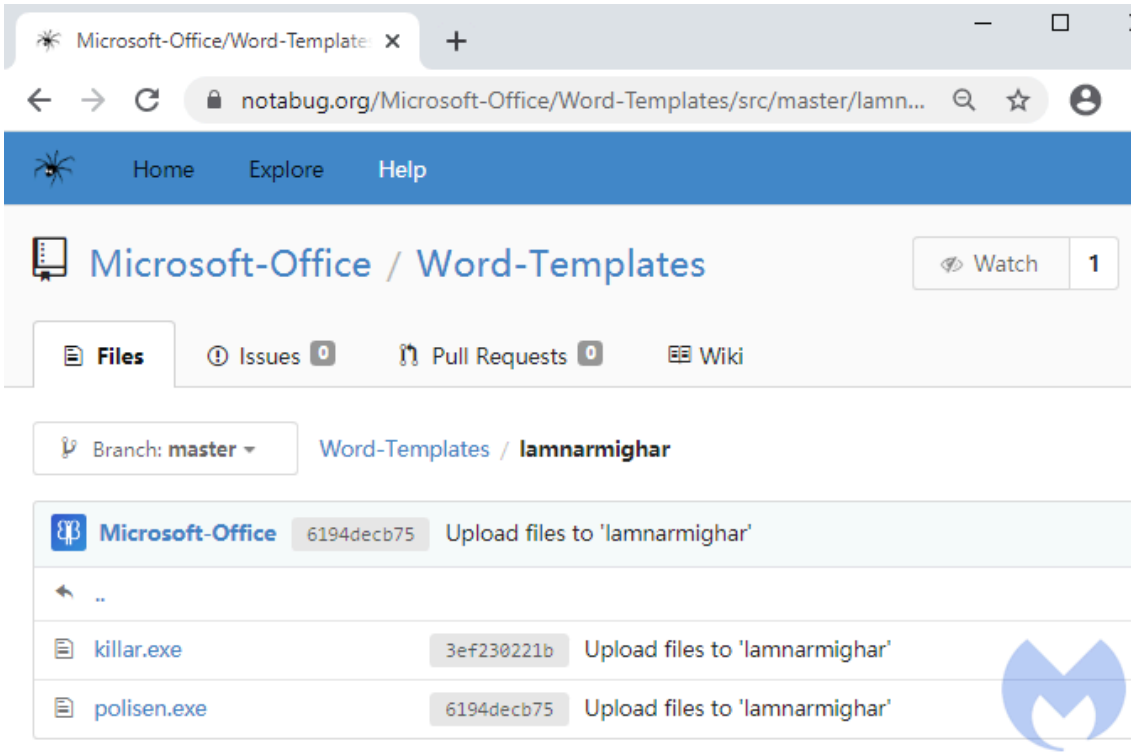
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



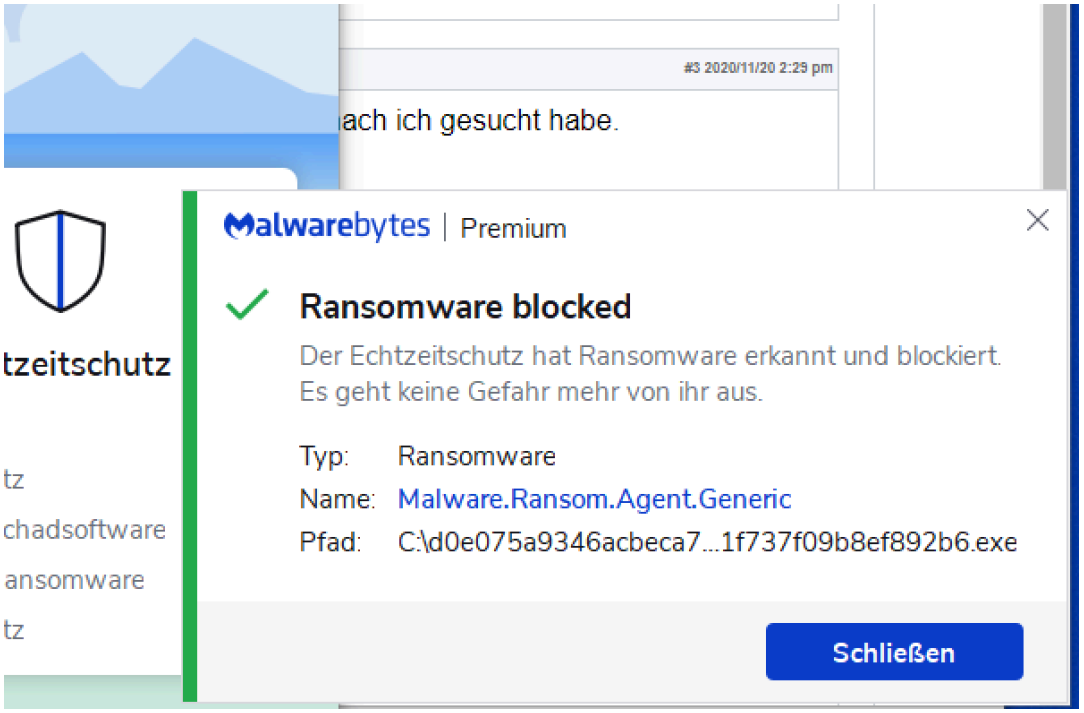
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

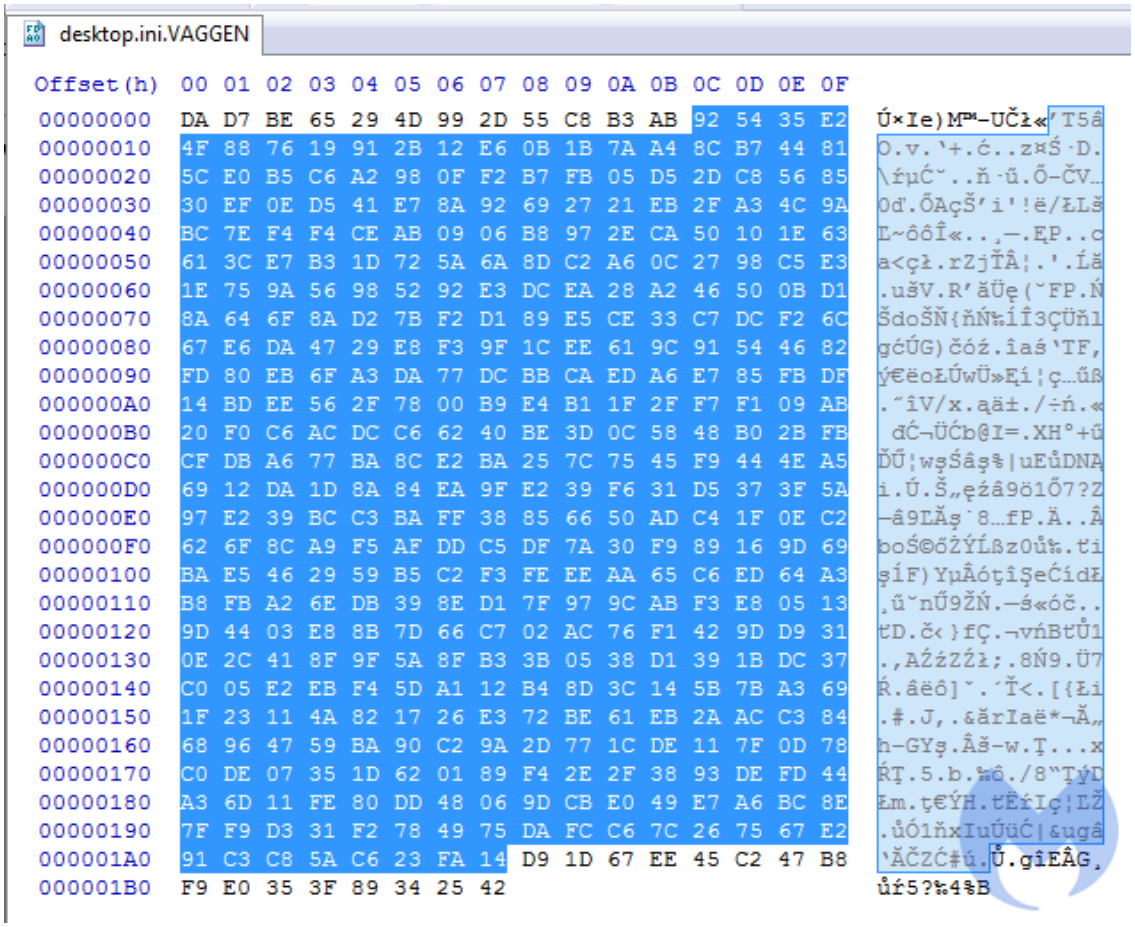
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

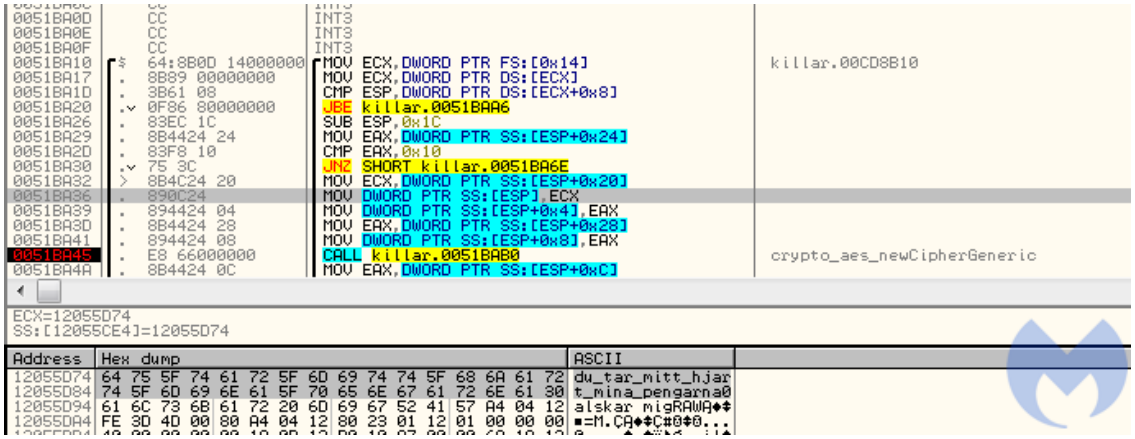


The content of the output file (with .VAGGEN extension) contains:

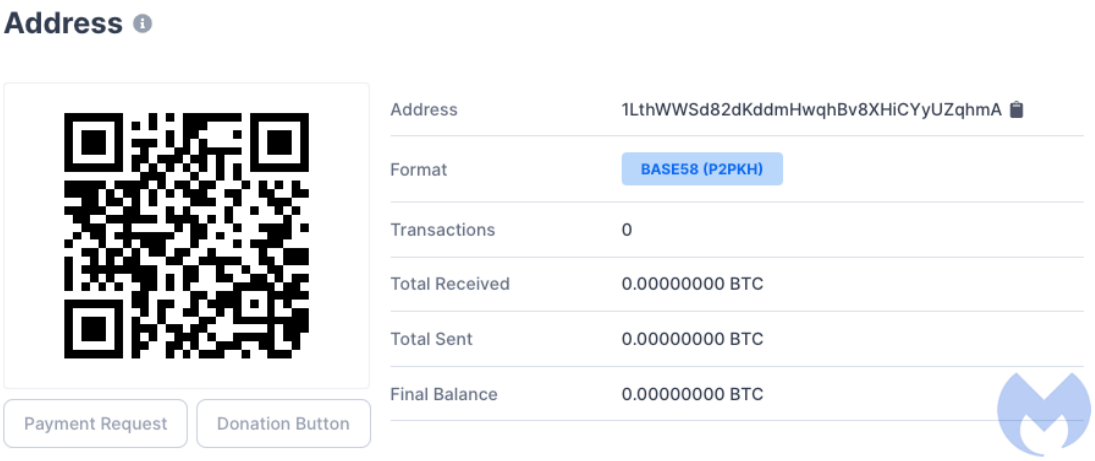
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

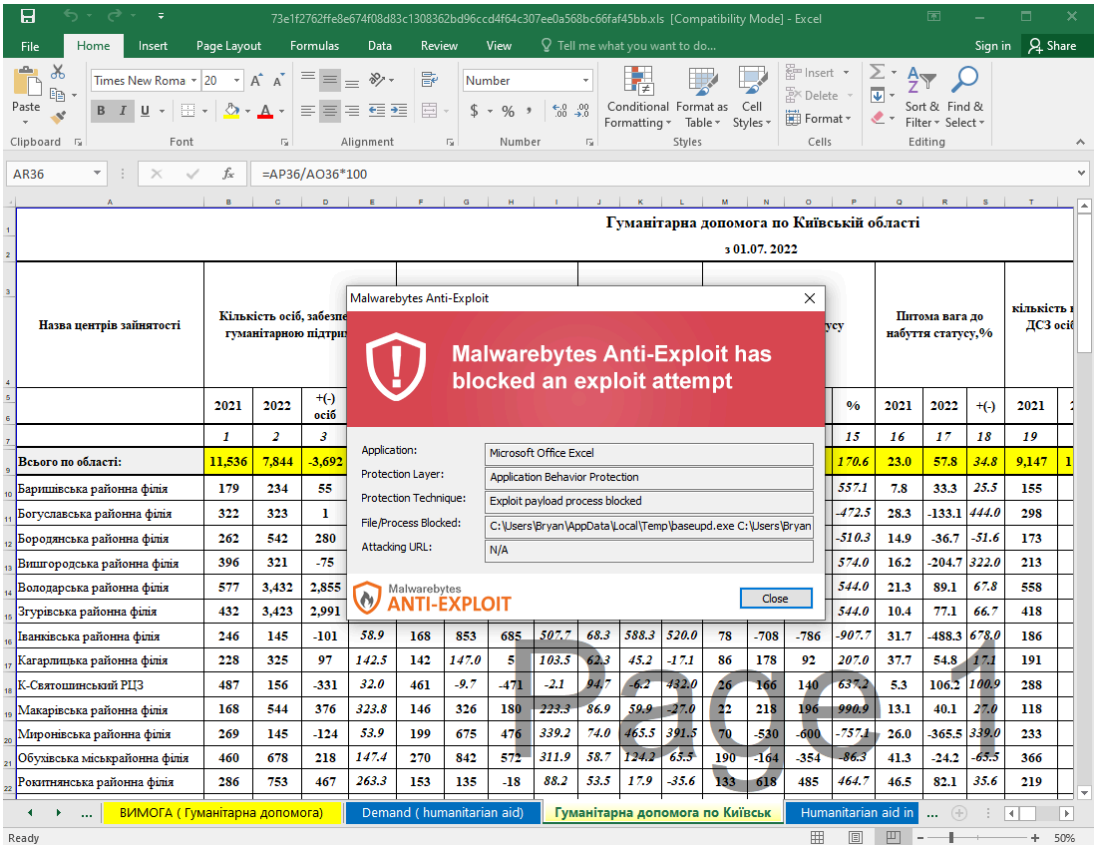
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

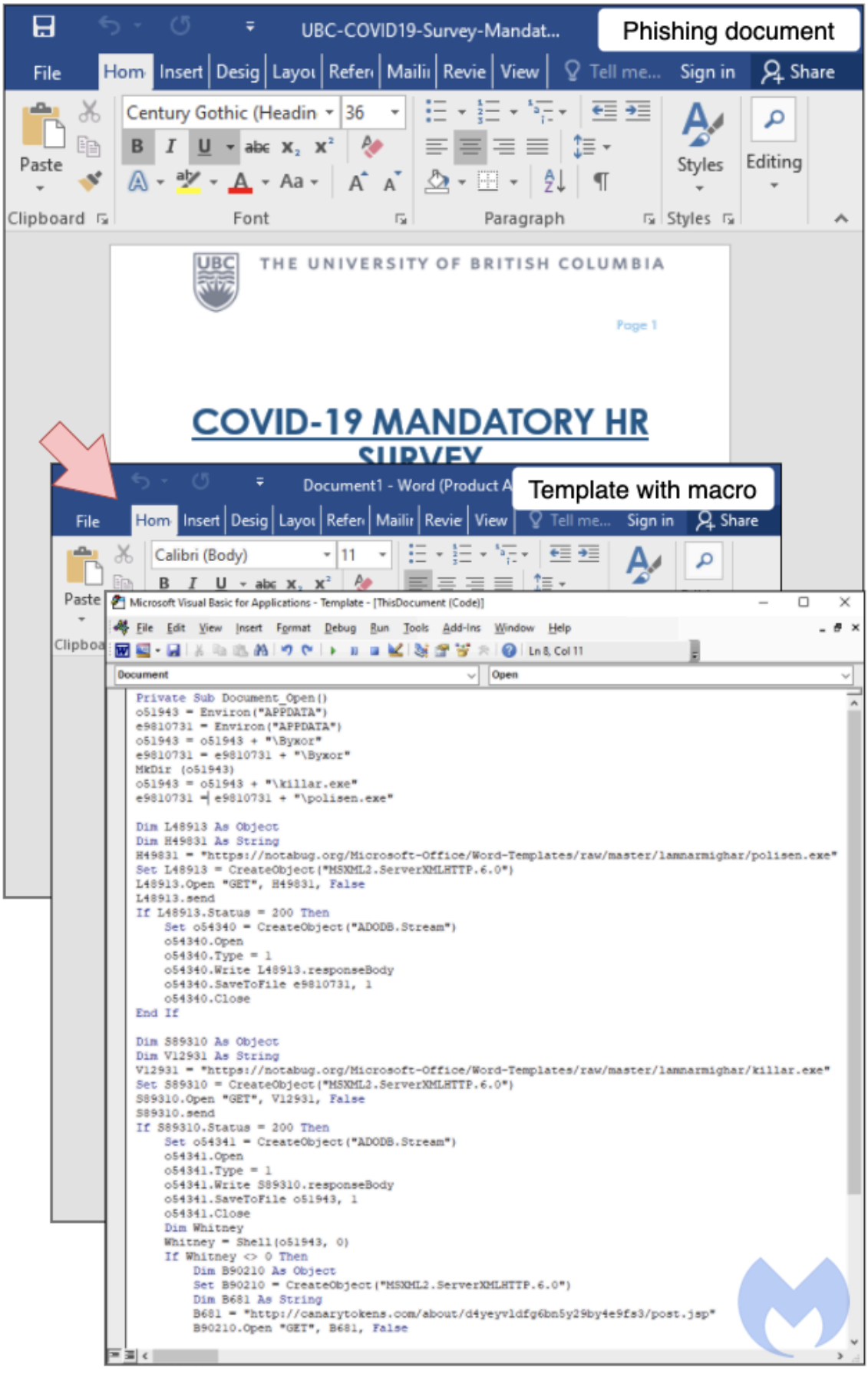
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

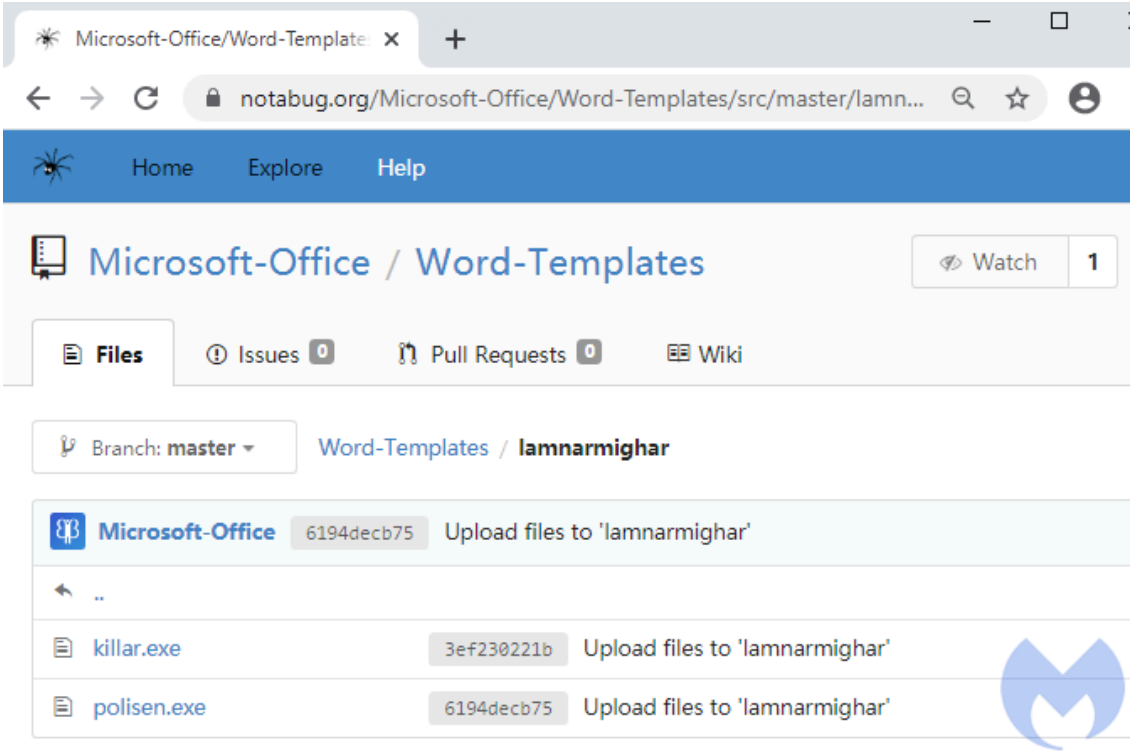
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



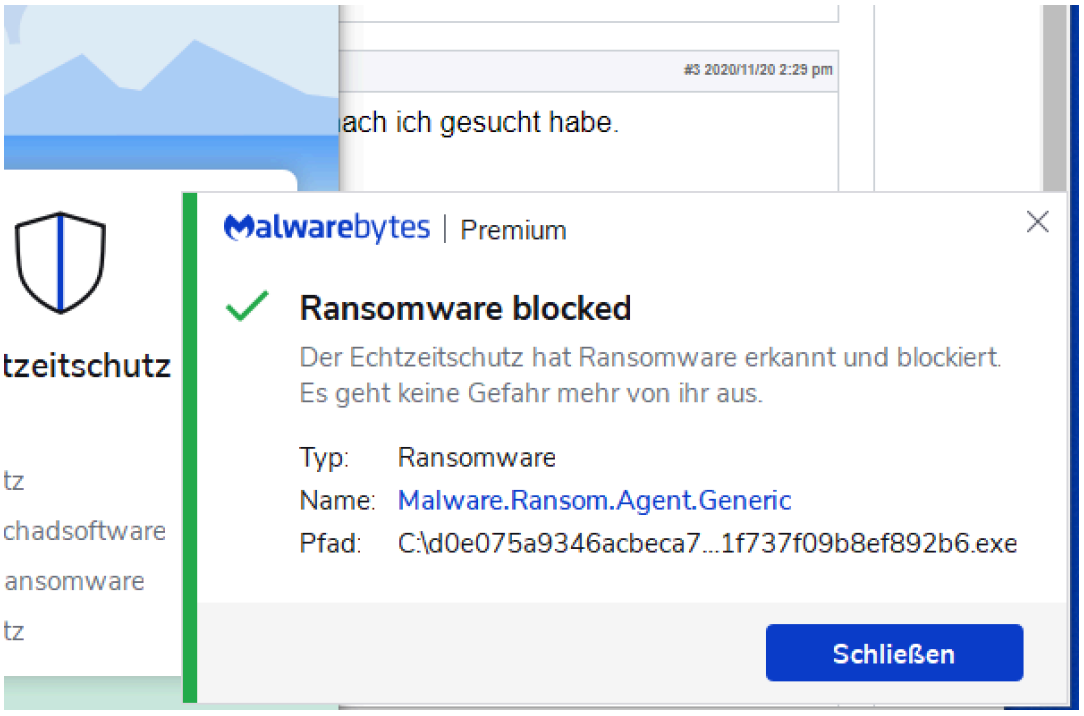
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

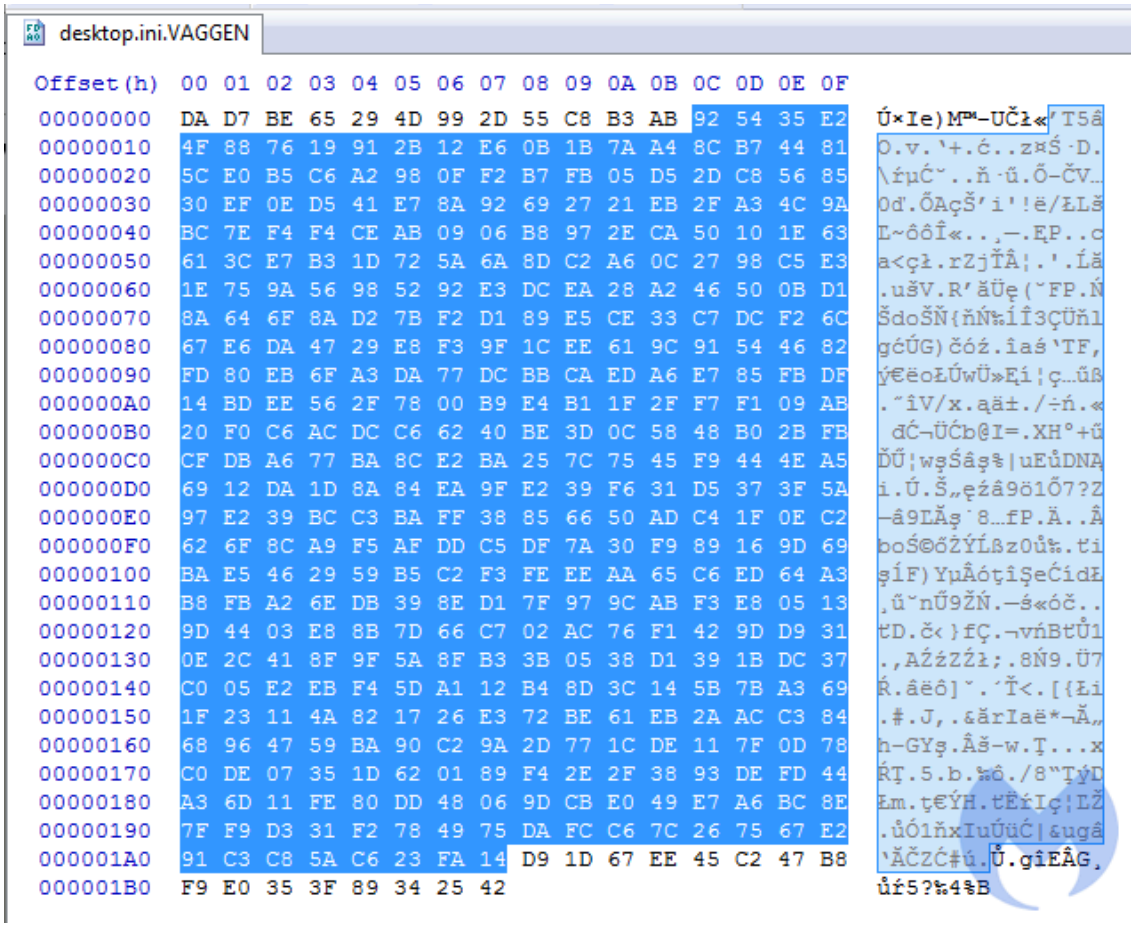
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

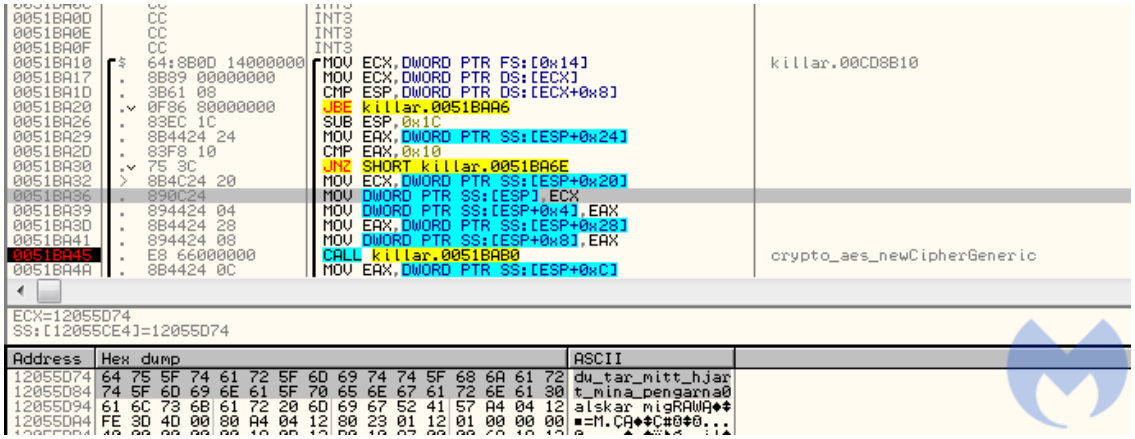


The content of the output file (with .VAGGEN extension) contains:

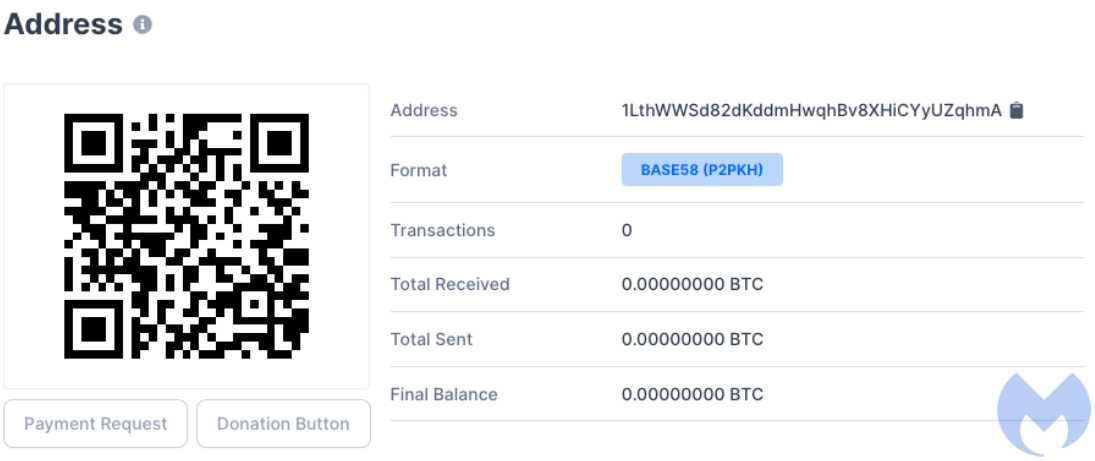
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

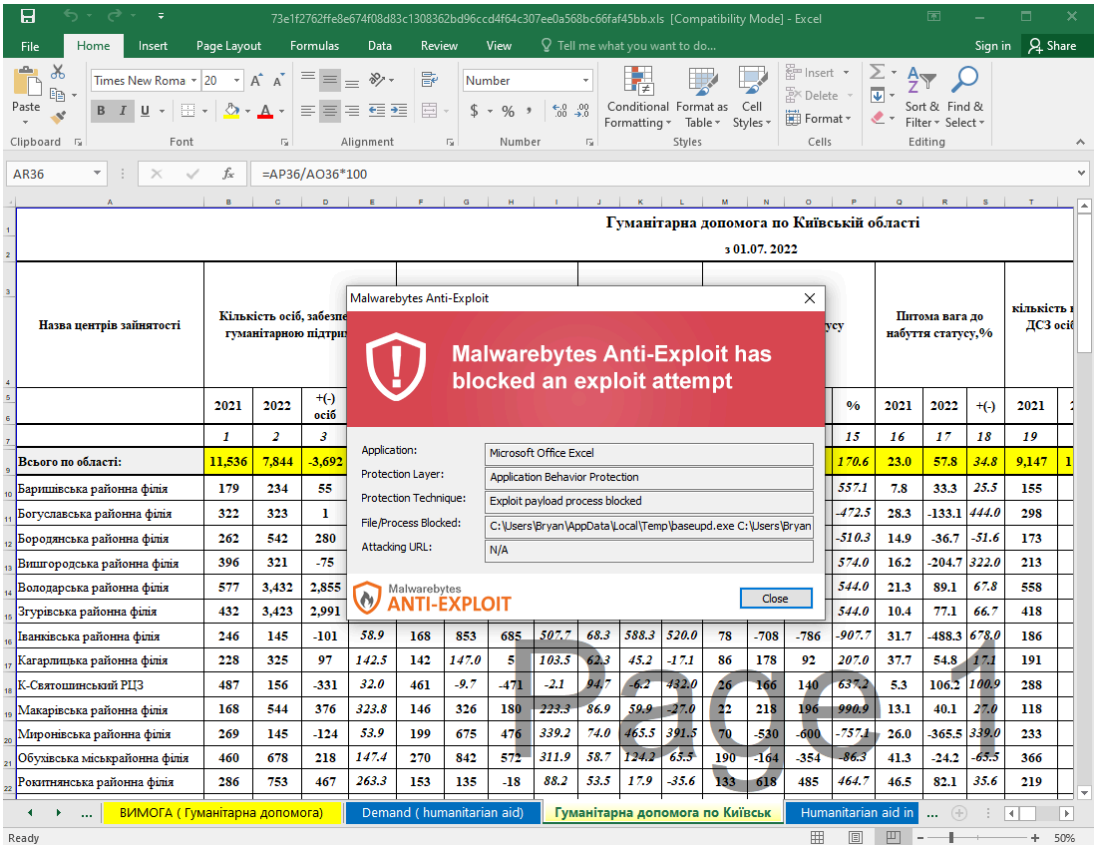
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

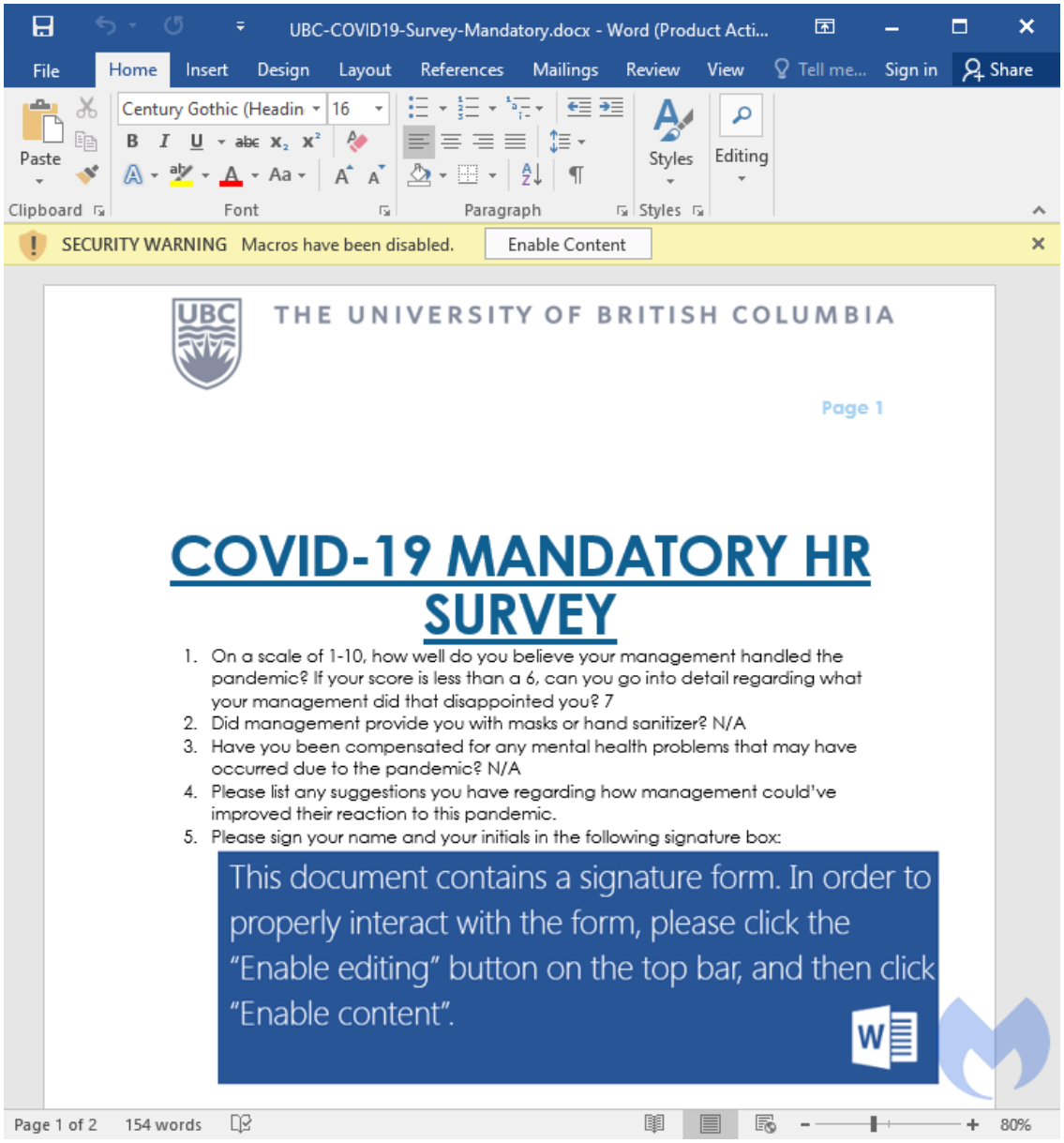
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

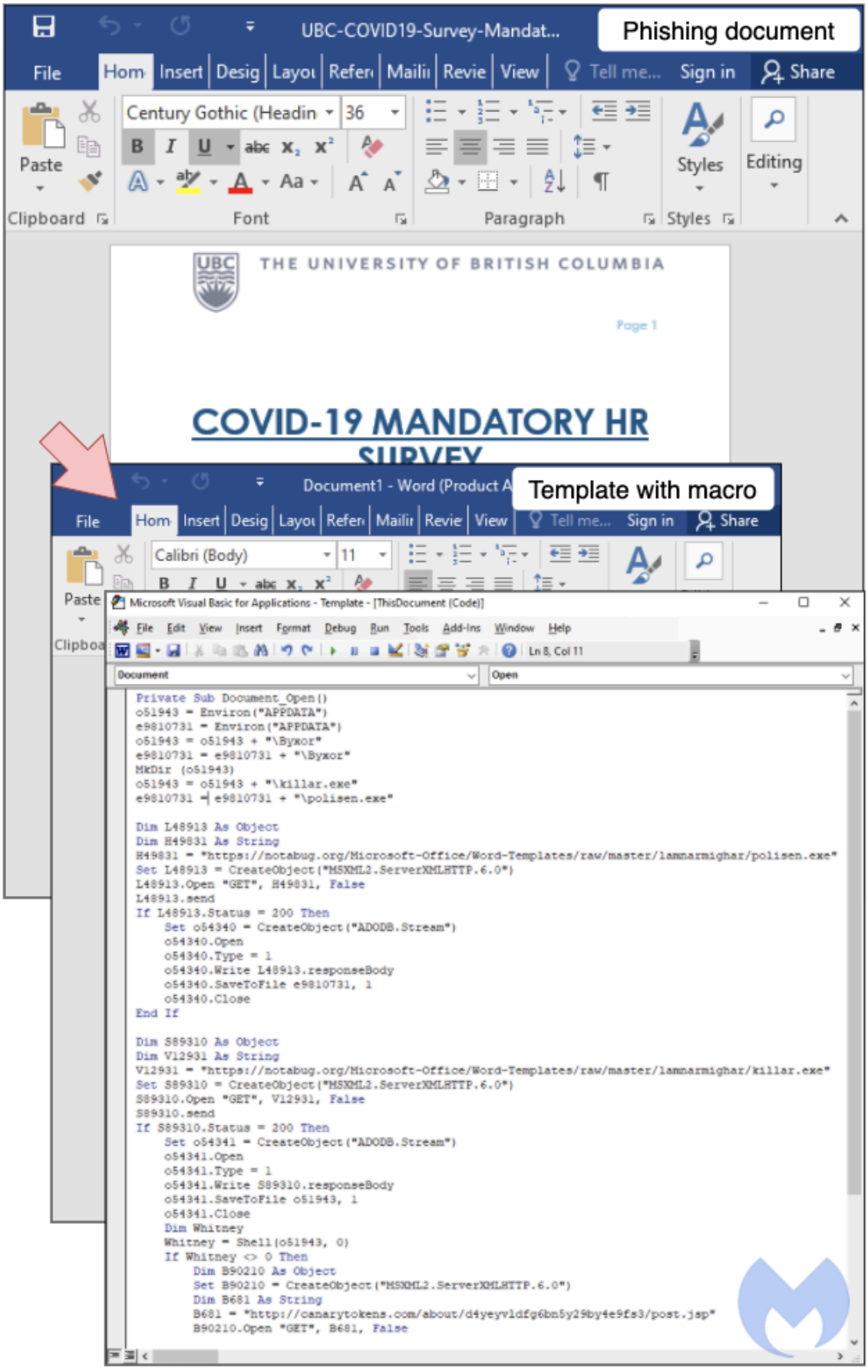
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

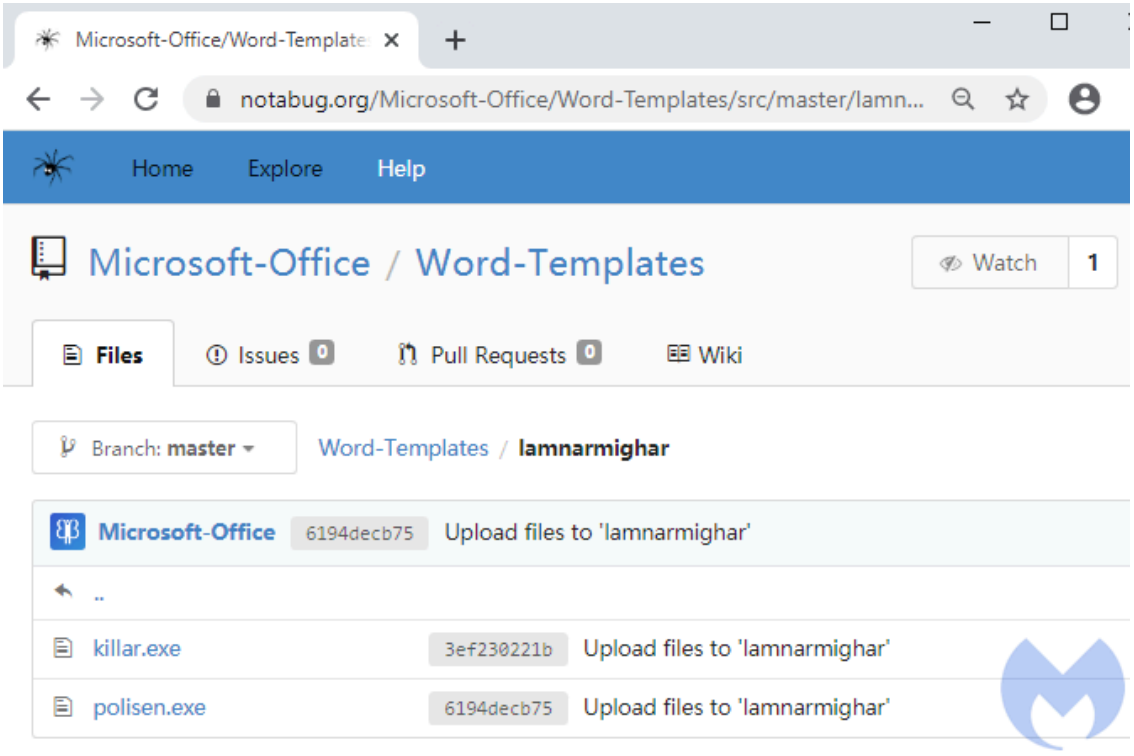
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



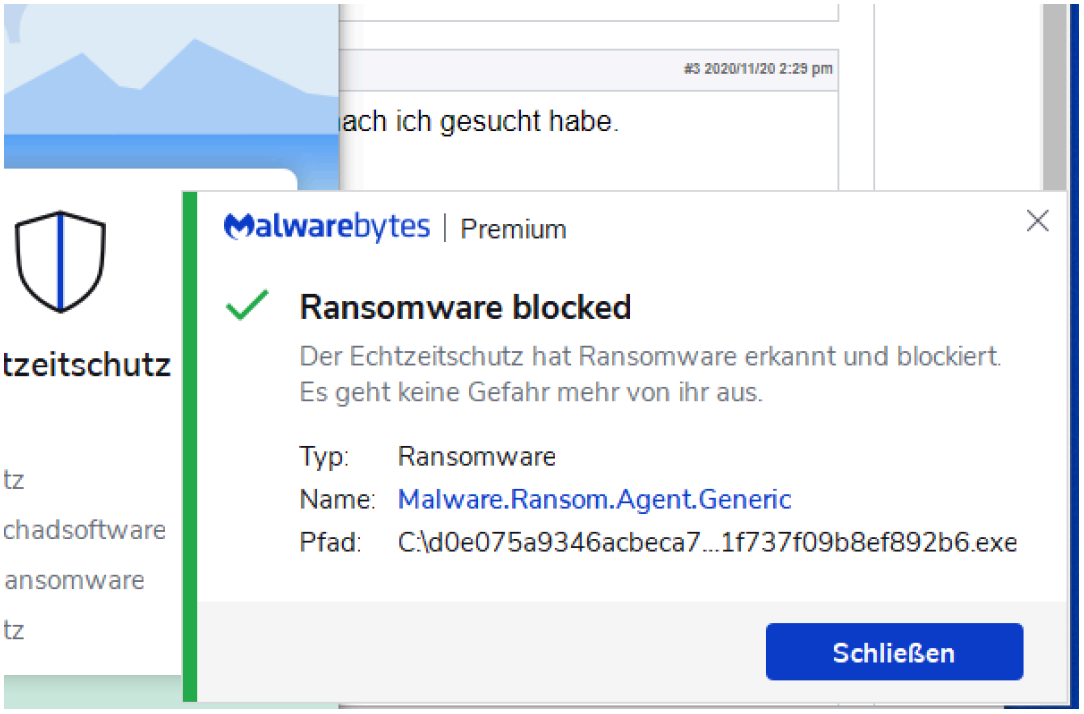
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

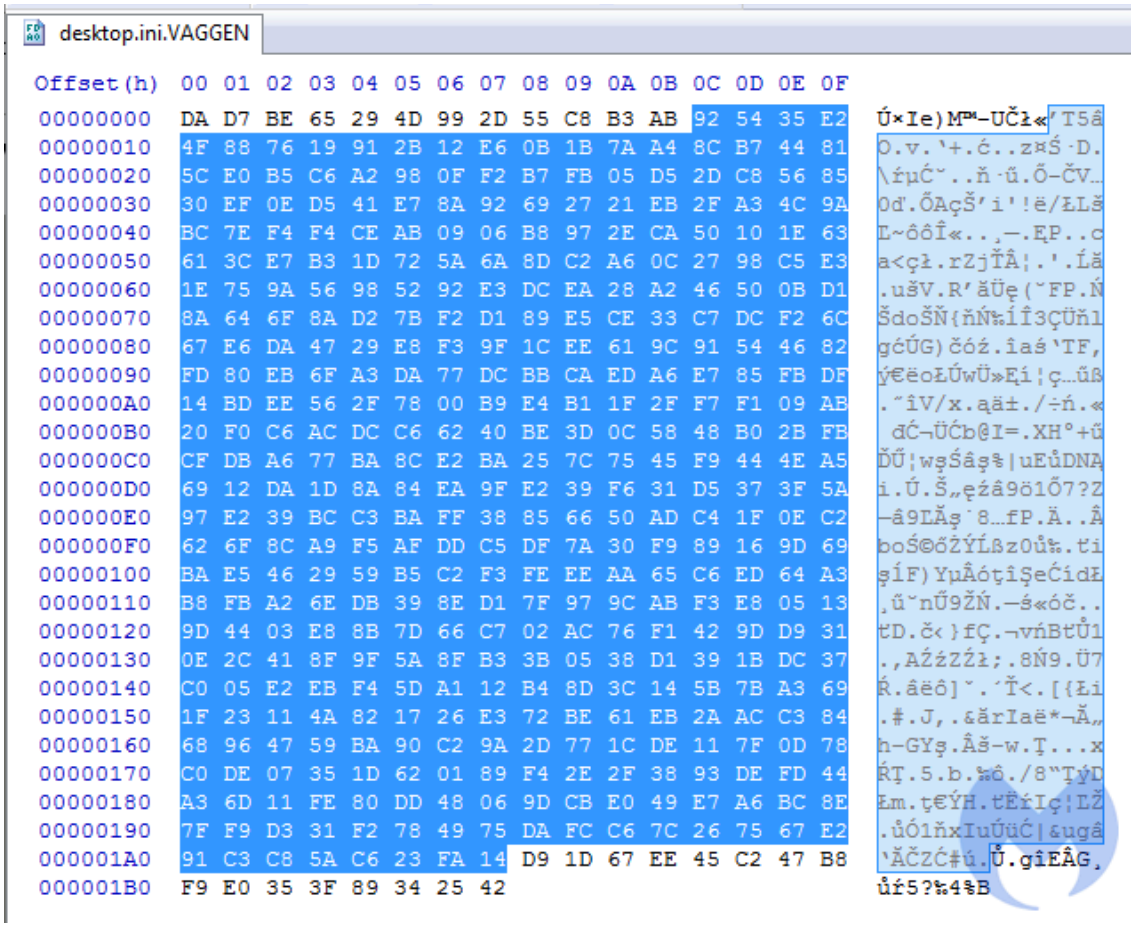
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

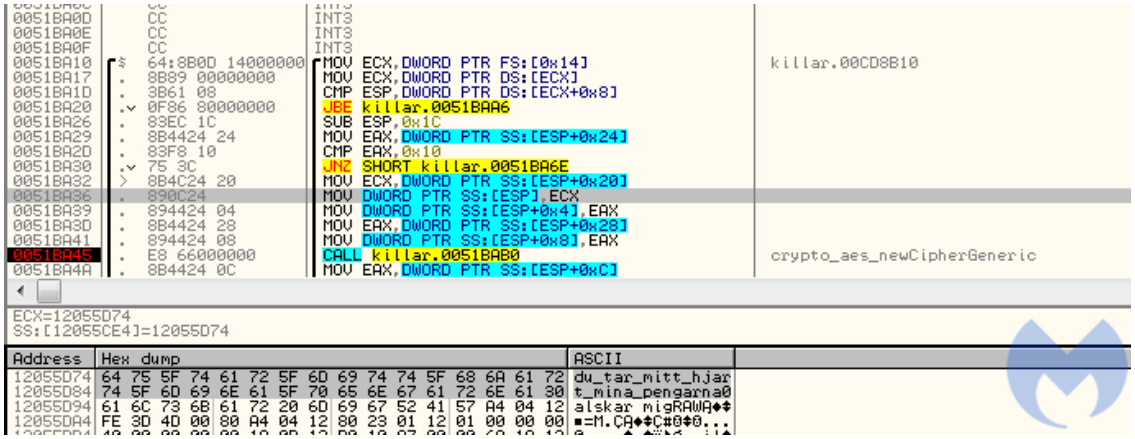


The content of the output file (with .VAGGEN extension) contains:

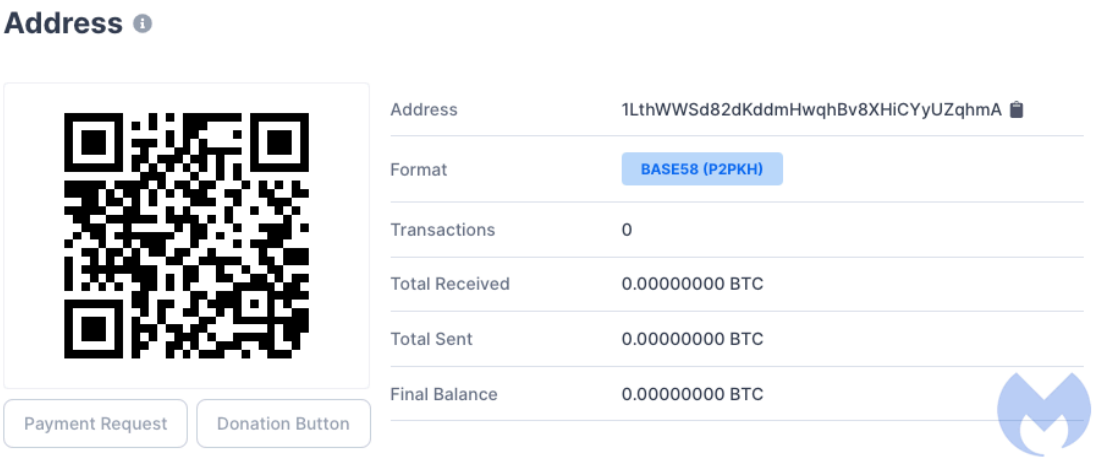
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

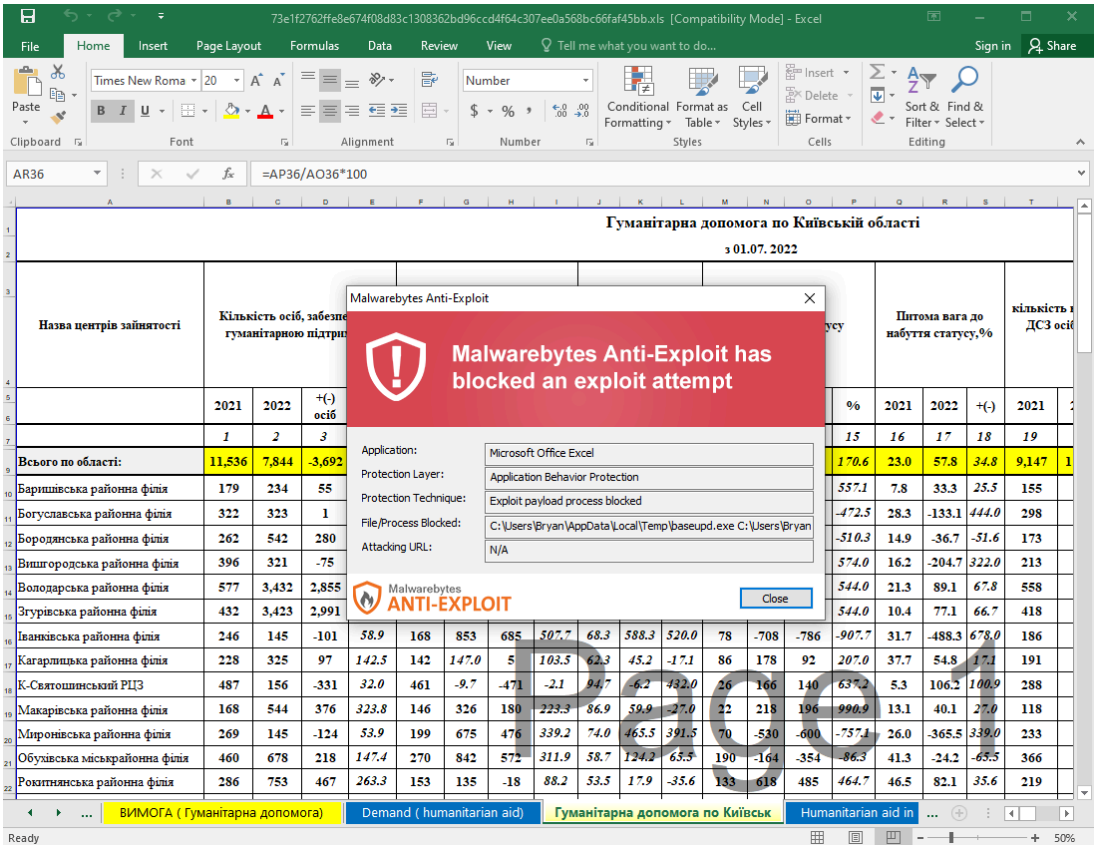
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

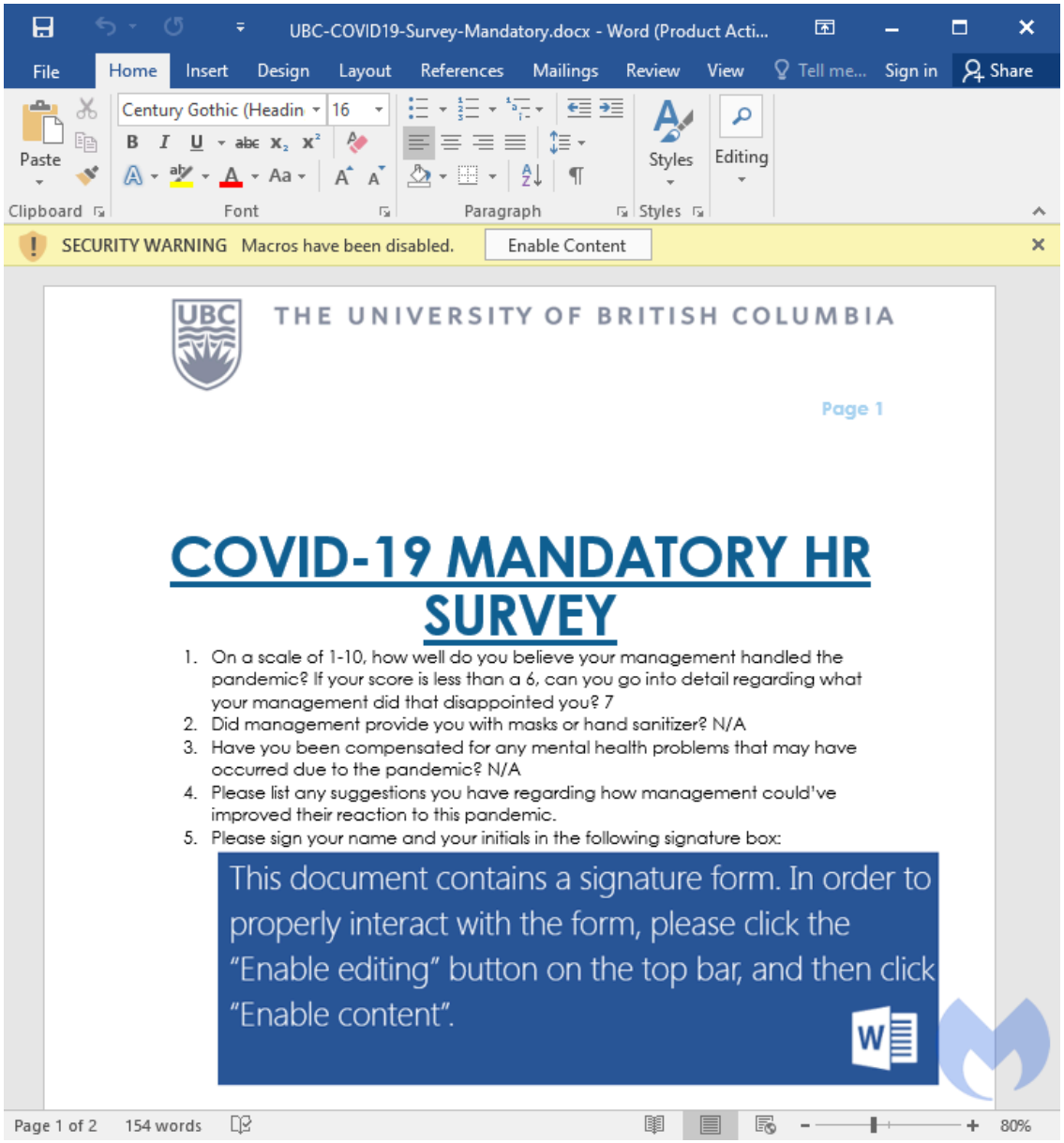
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

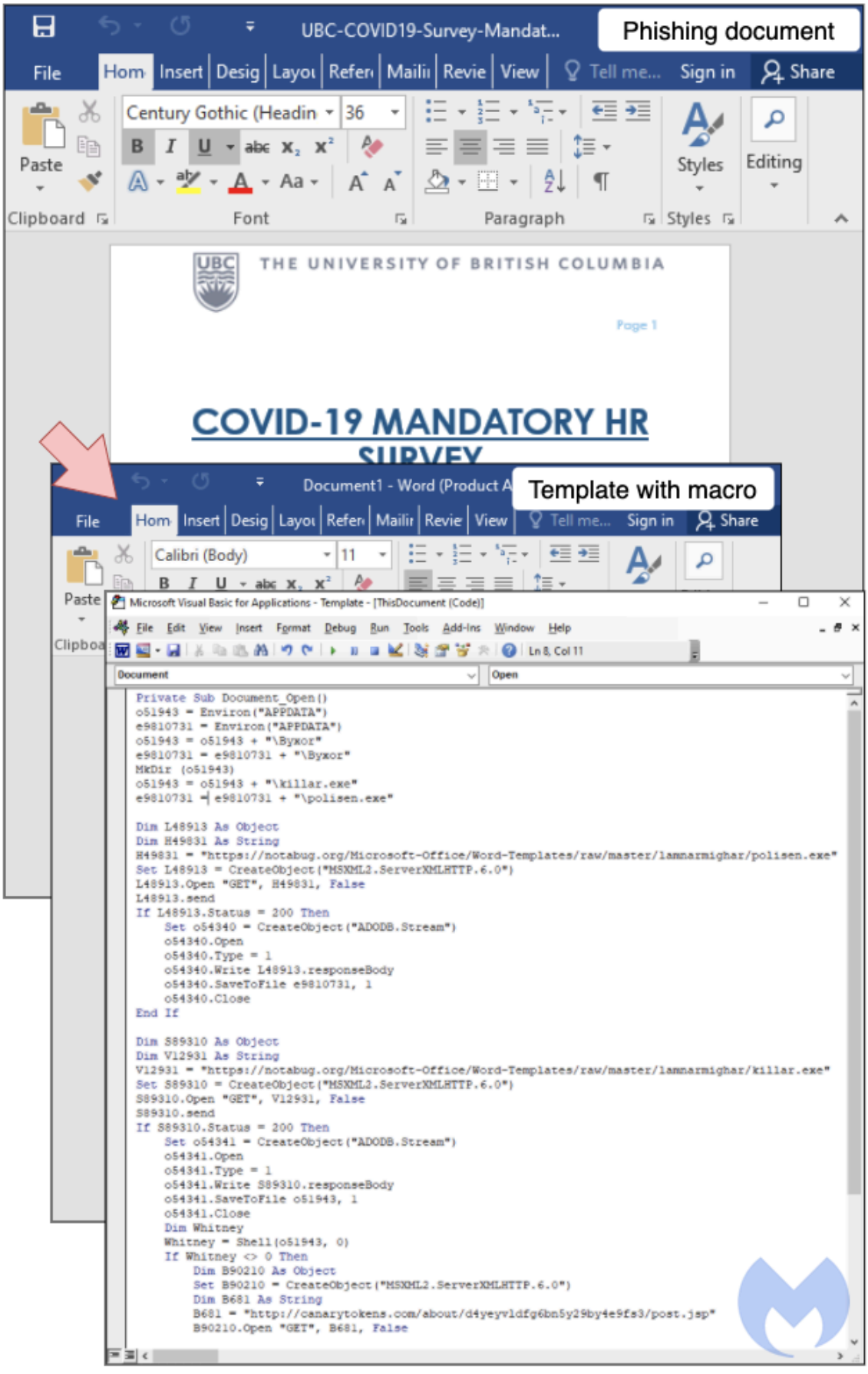
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

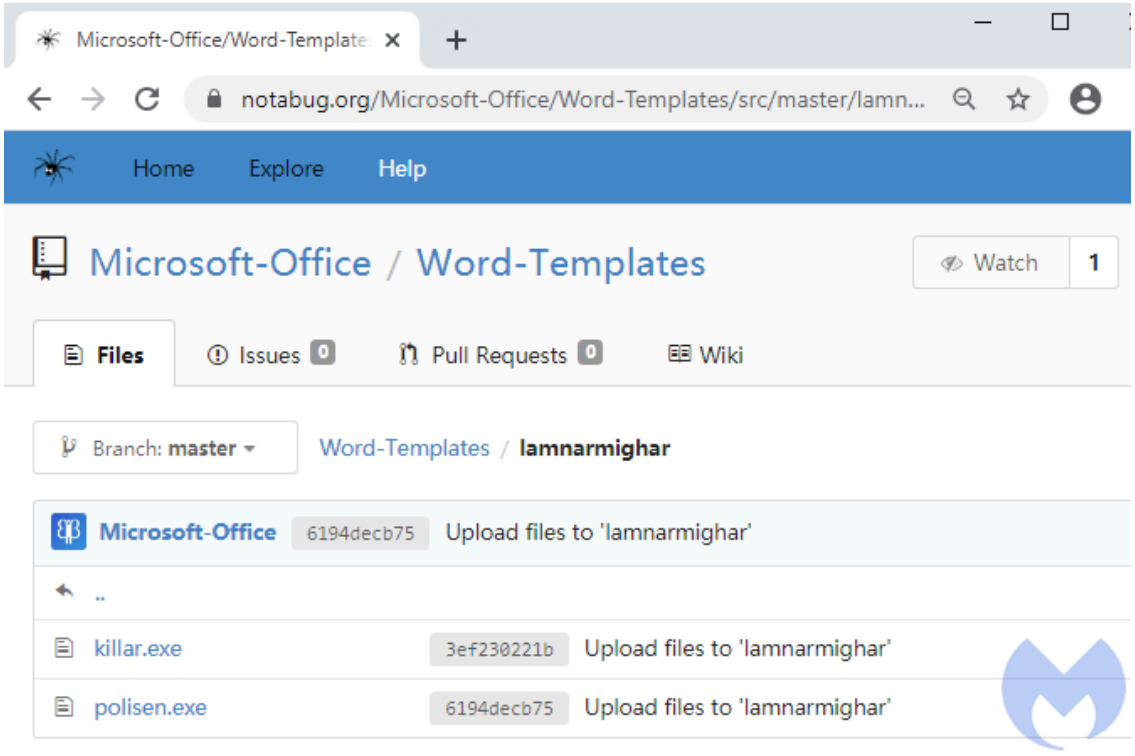
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



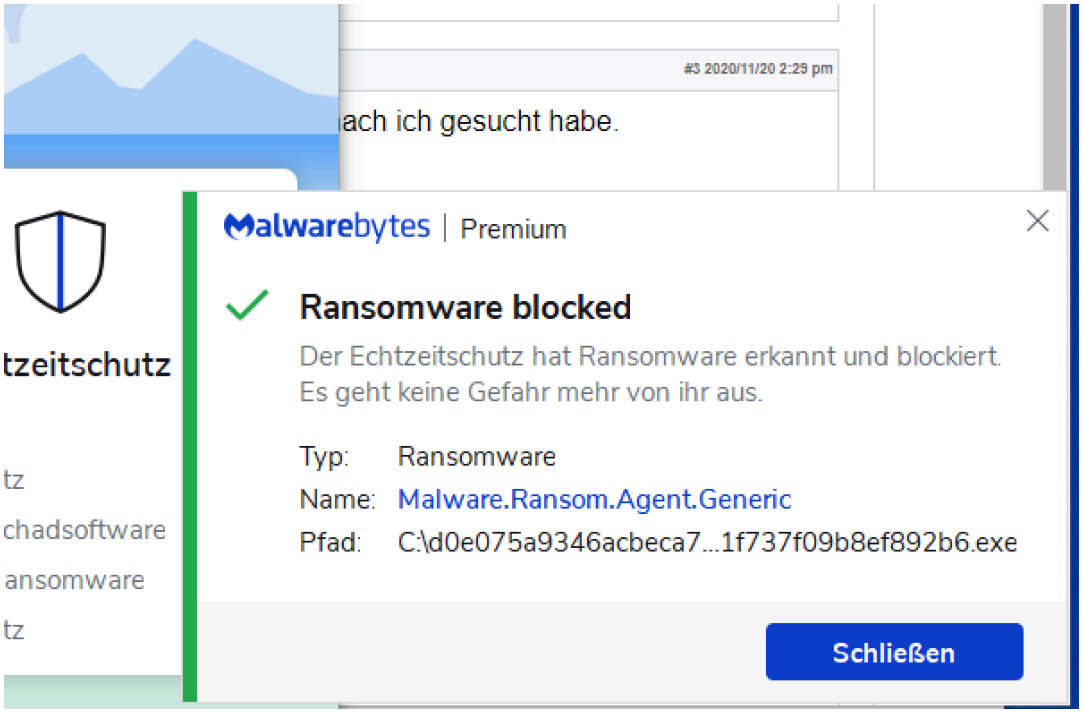
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

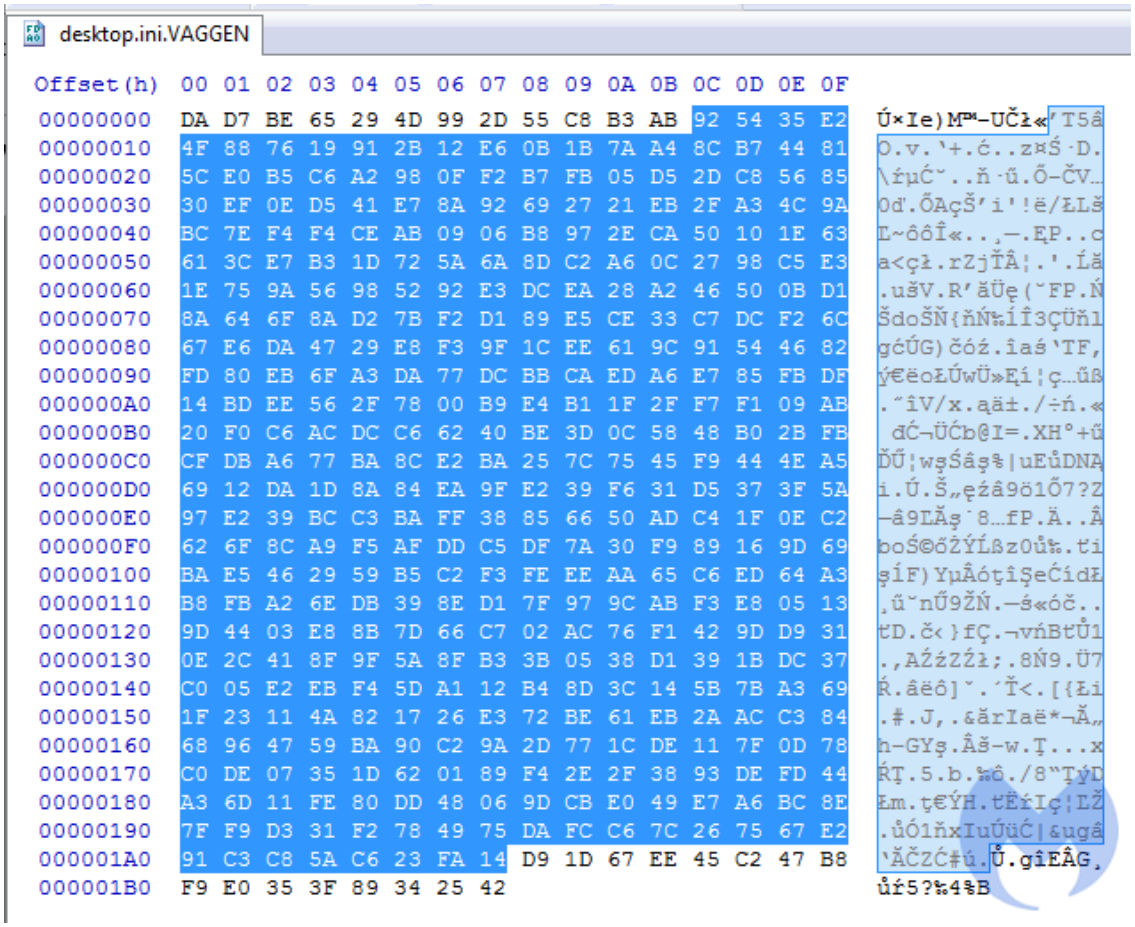
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

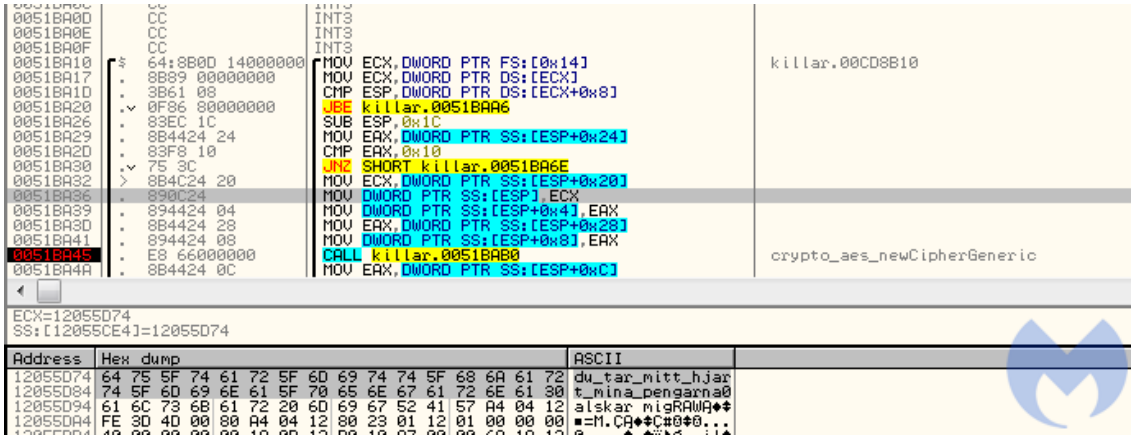


The content of the output file (with .VAGGEN extension) contains:

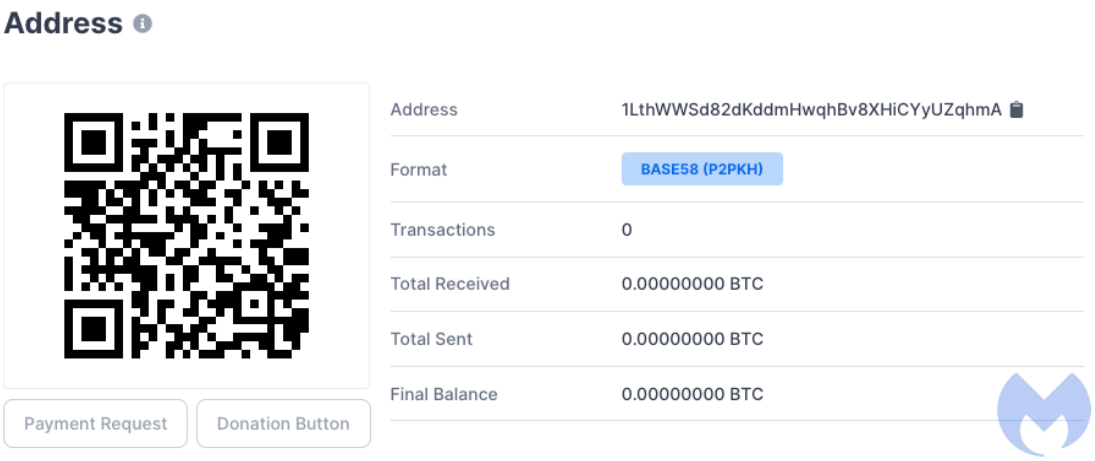
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

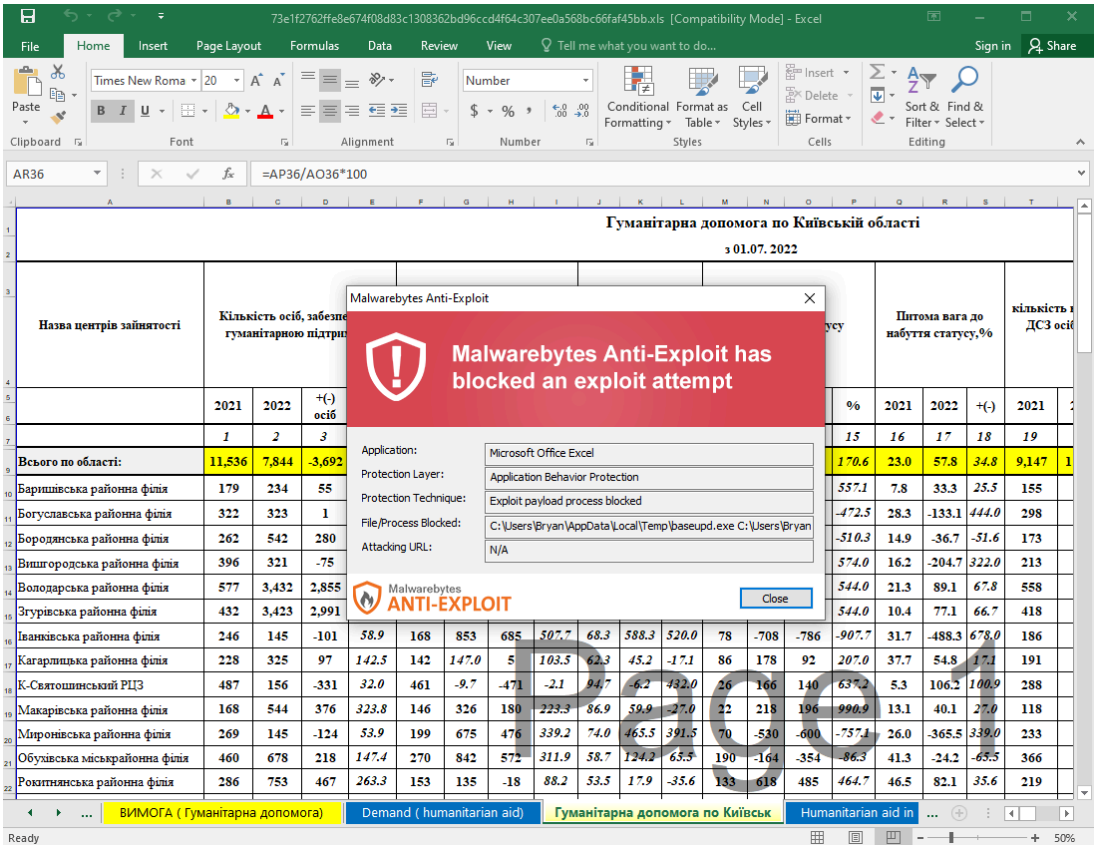
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

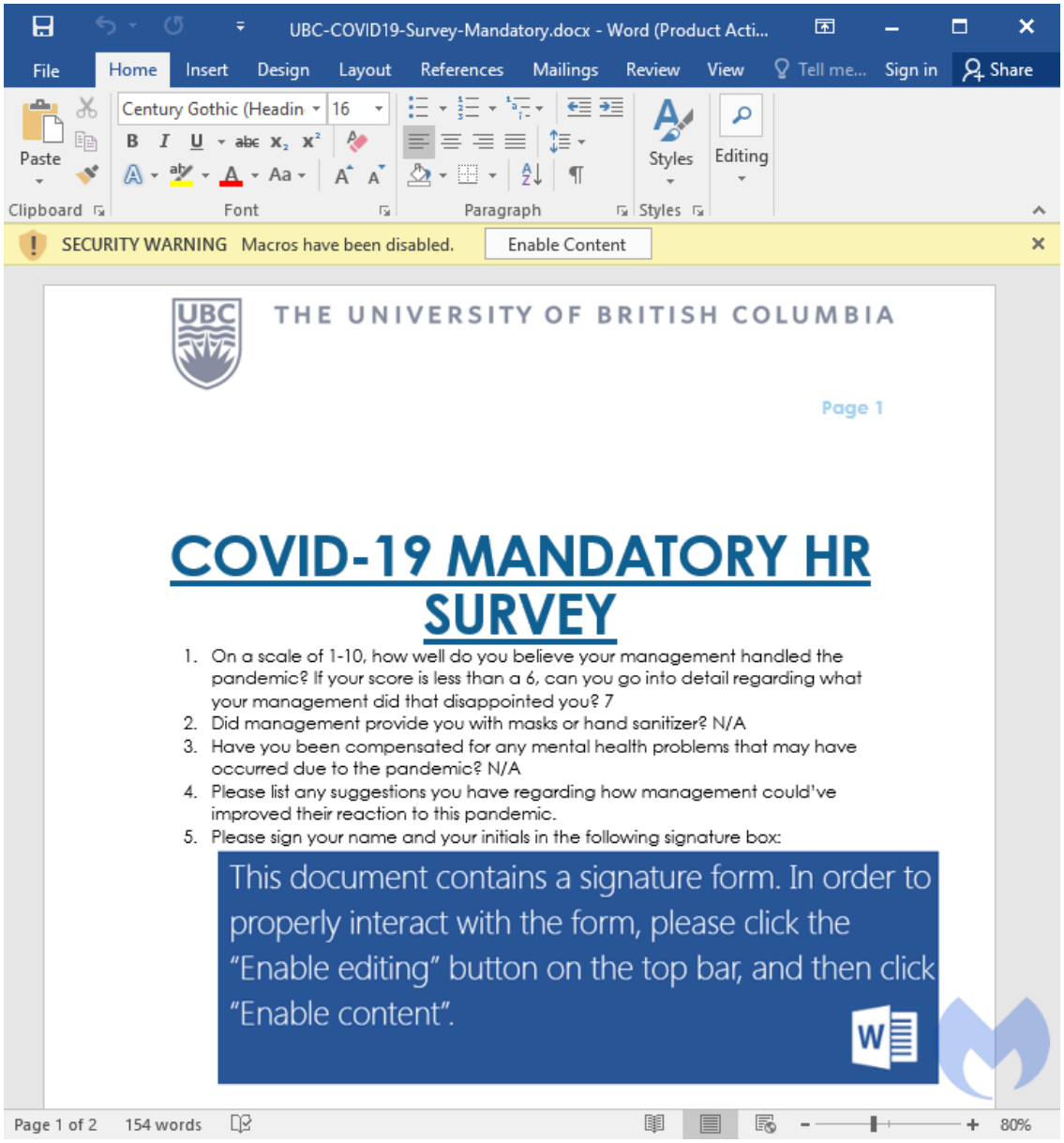
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

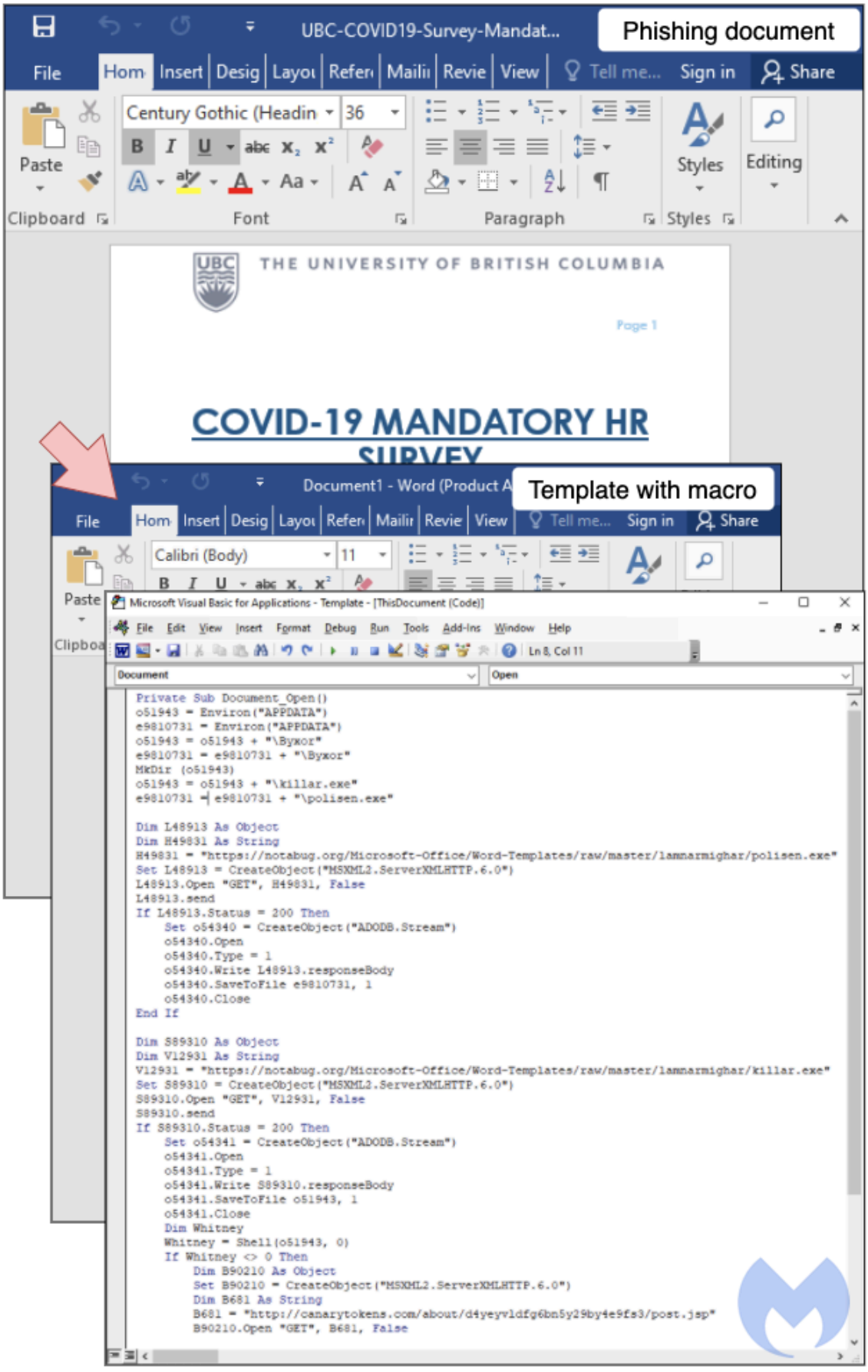
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

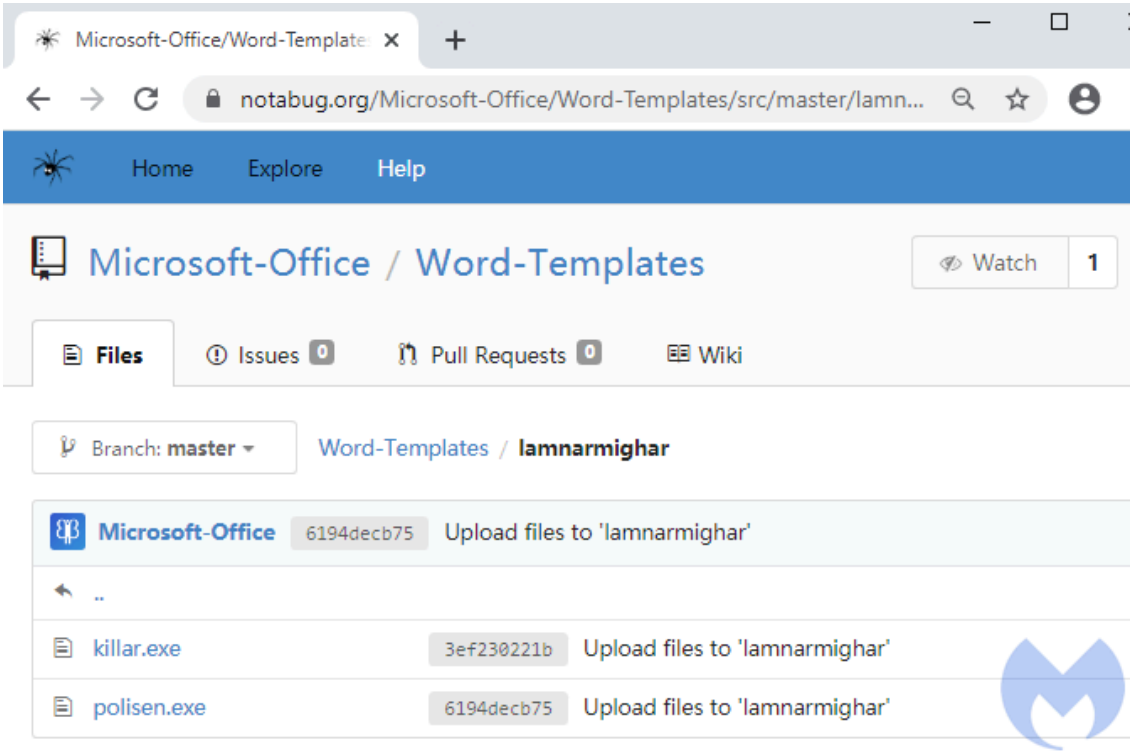
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



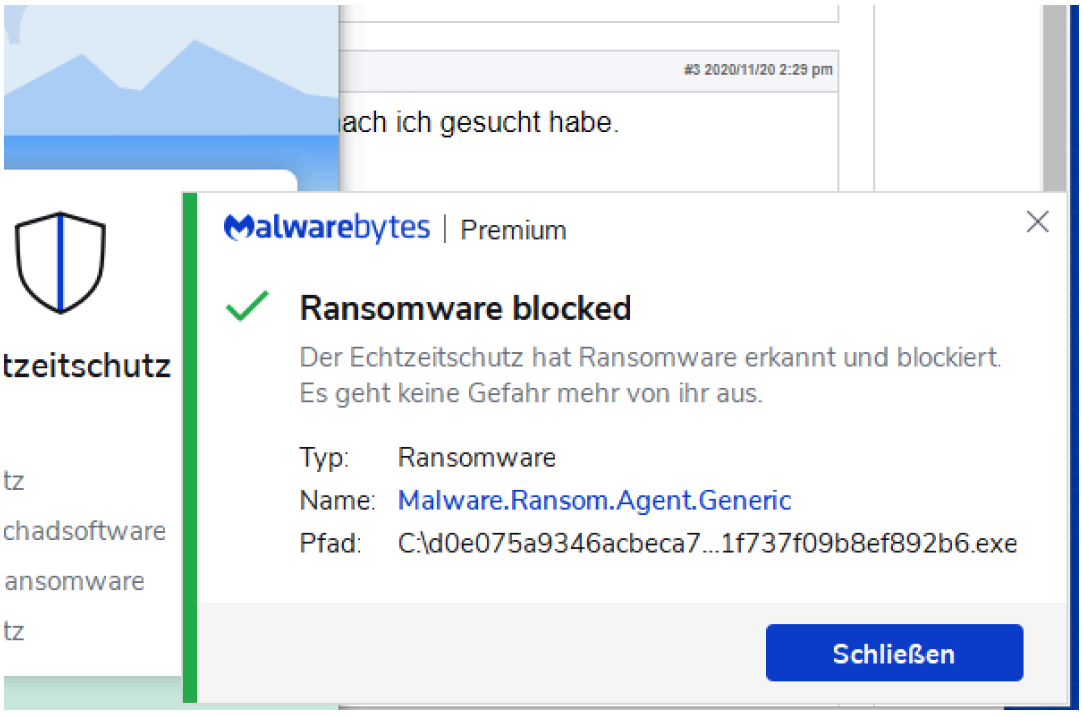
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

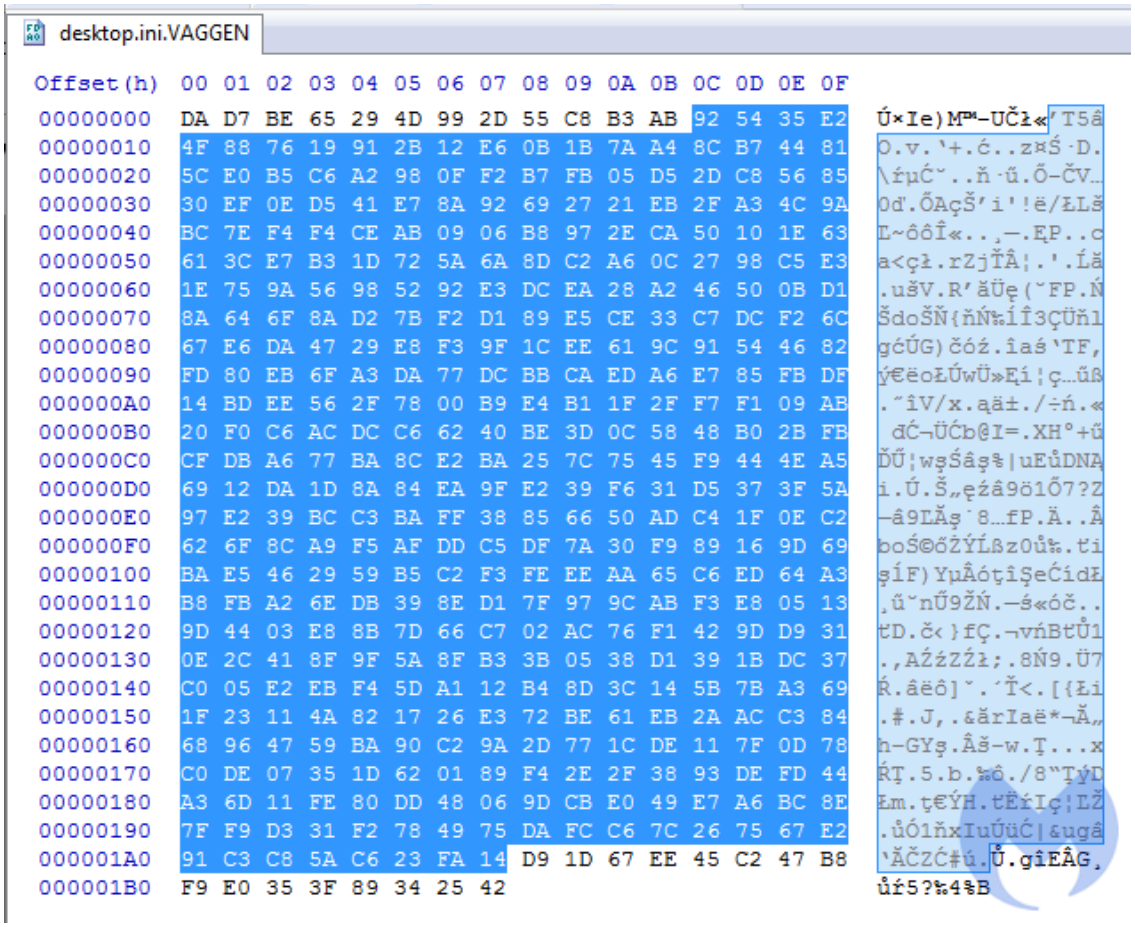
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

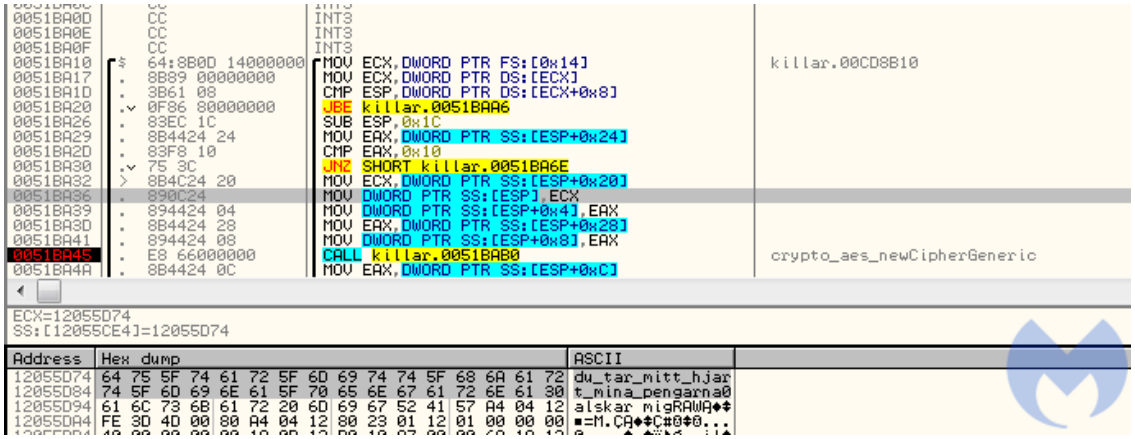


The content of the output file (with .VAGGEN extension) contains:

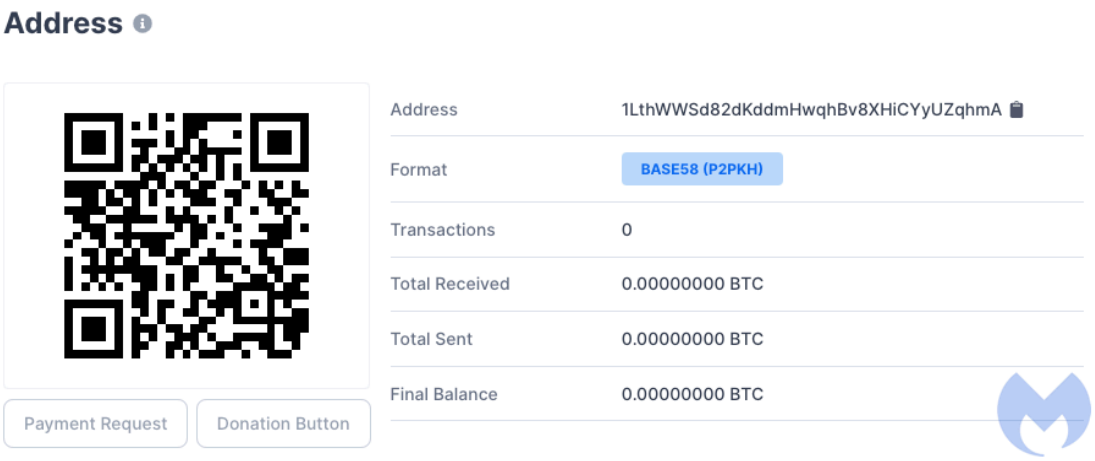
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

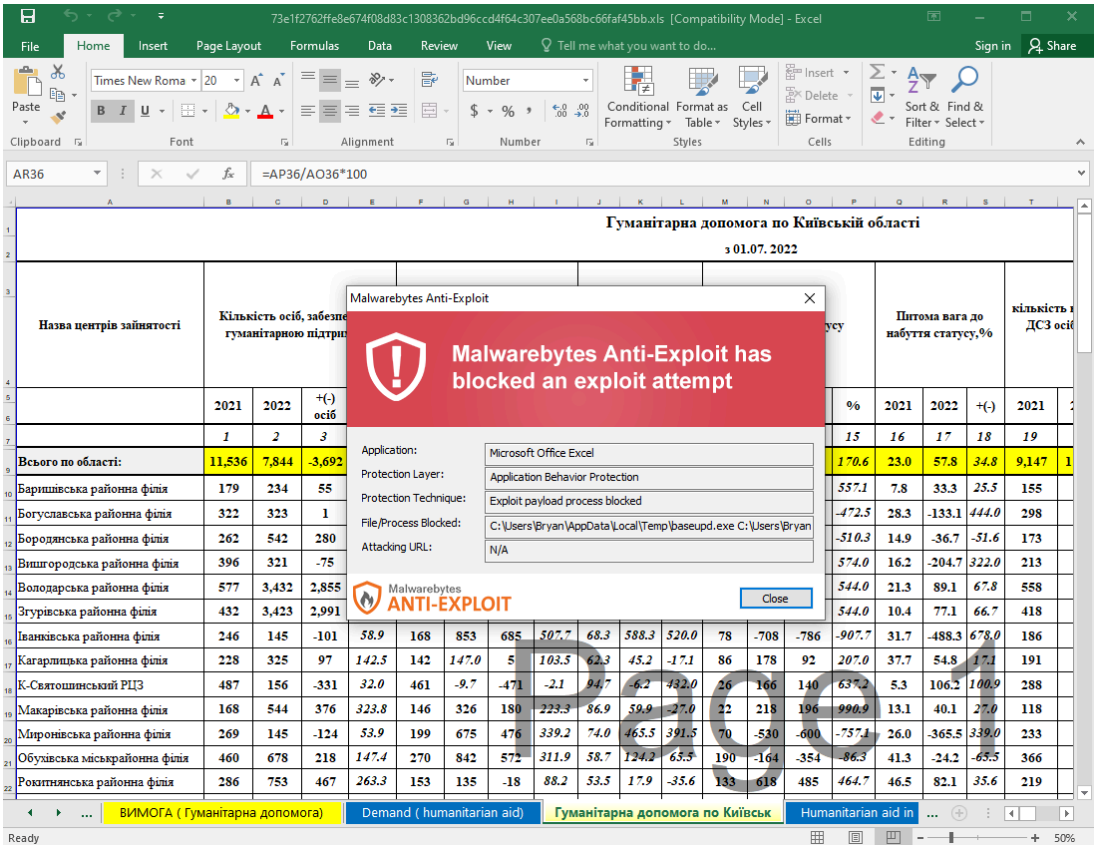
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

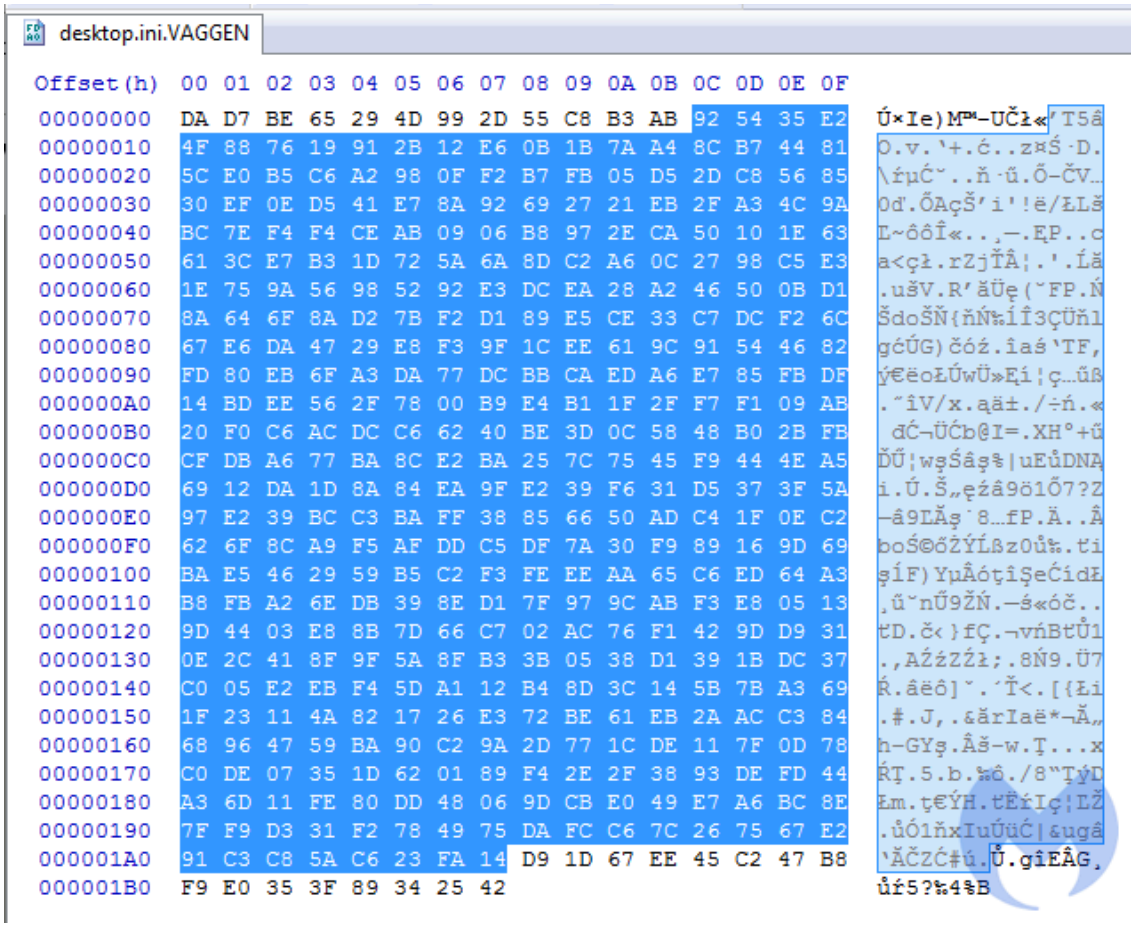
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

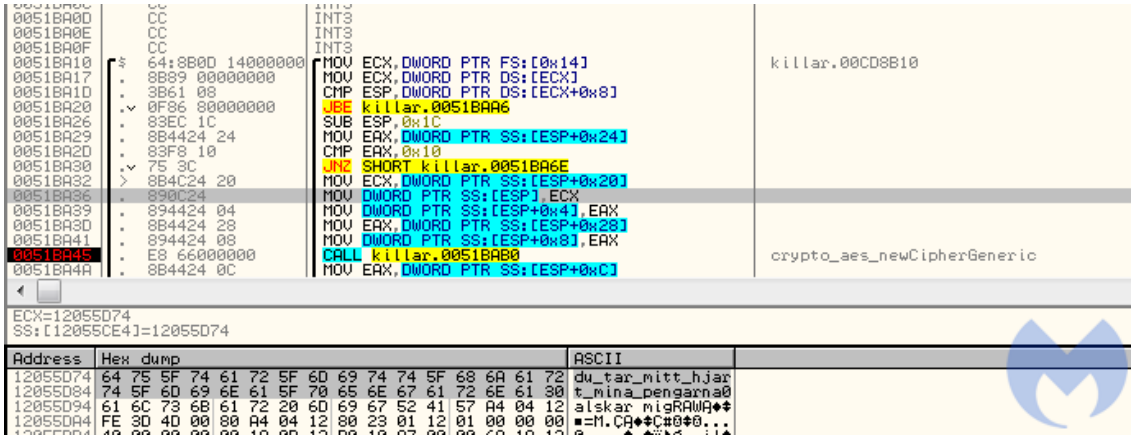


The content of the output file (with .VAGGEN extension) contains:

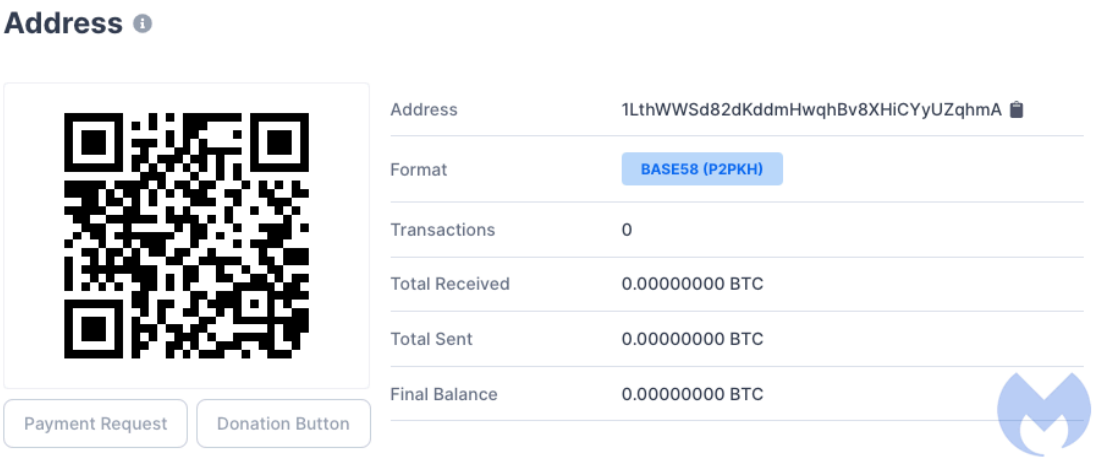
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

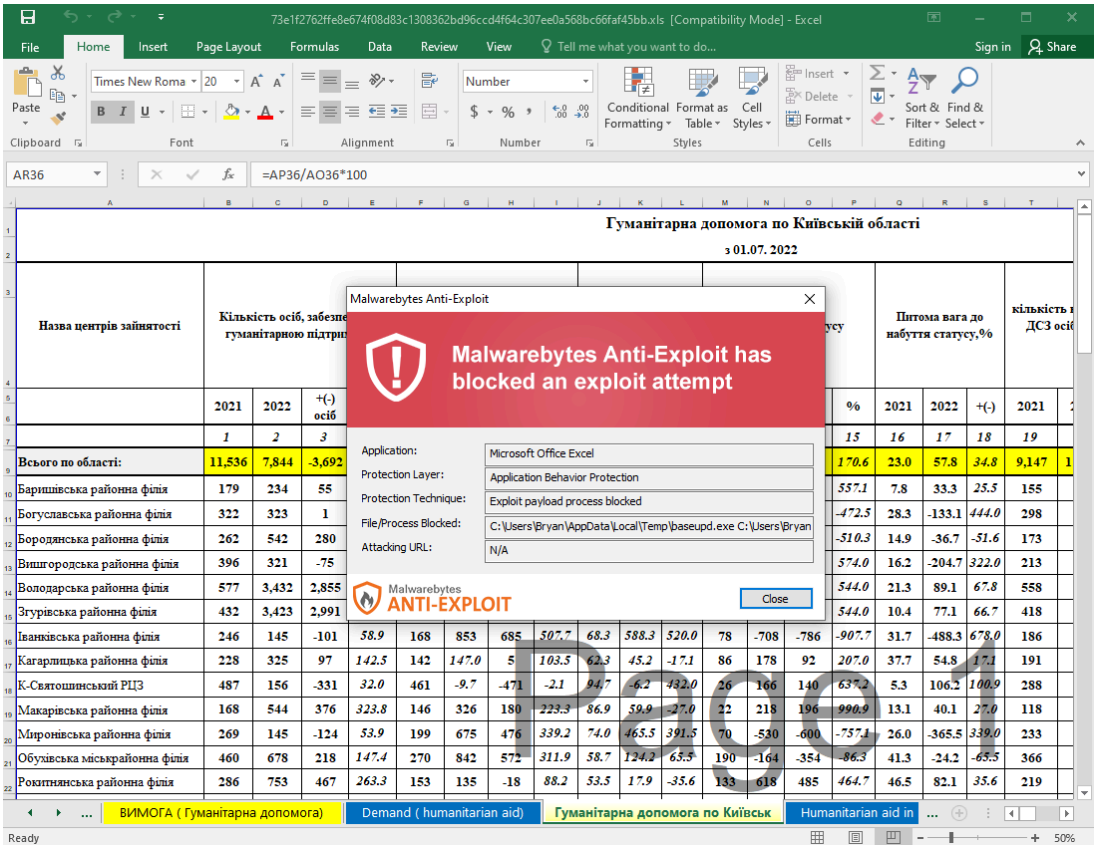
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

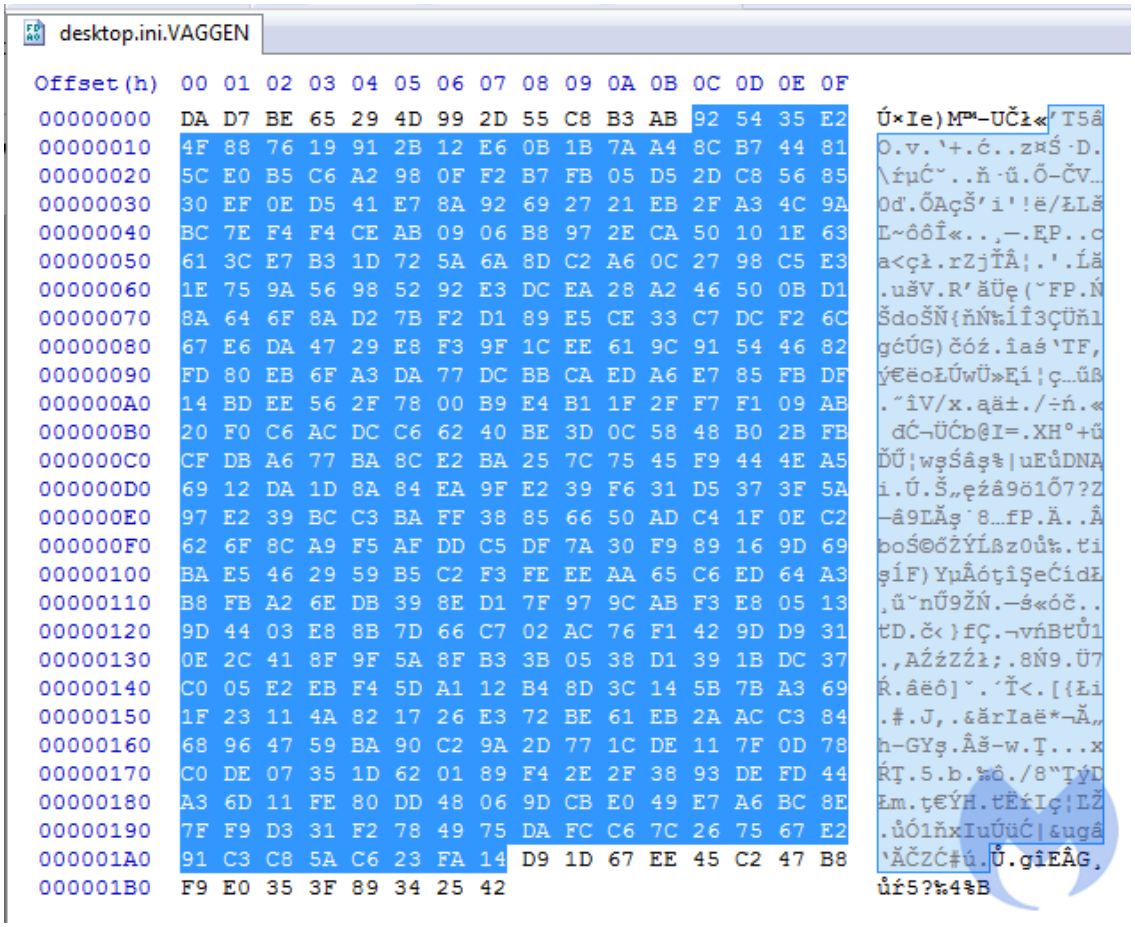
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



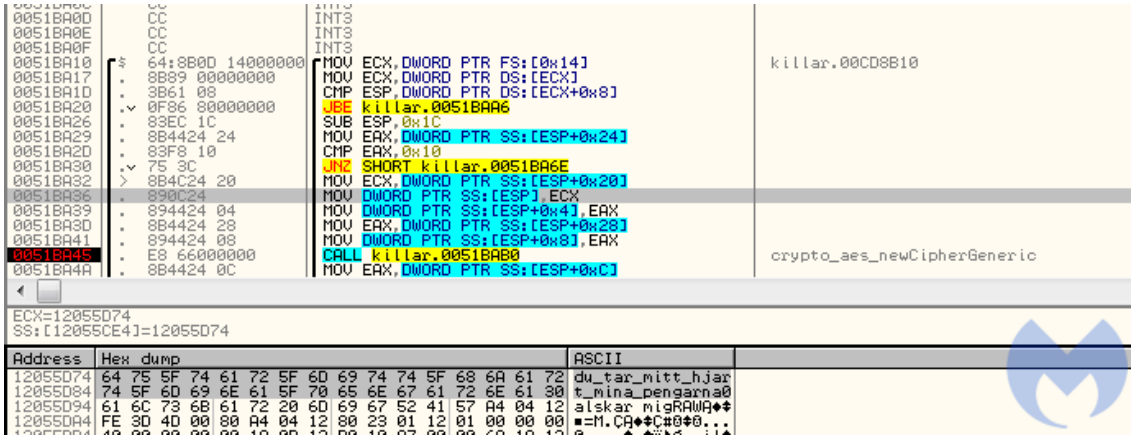
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

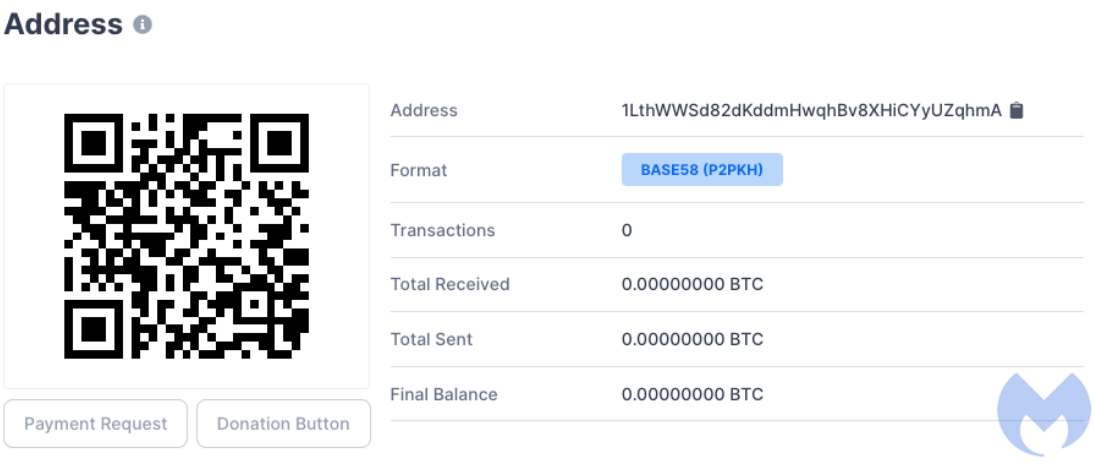
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

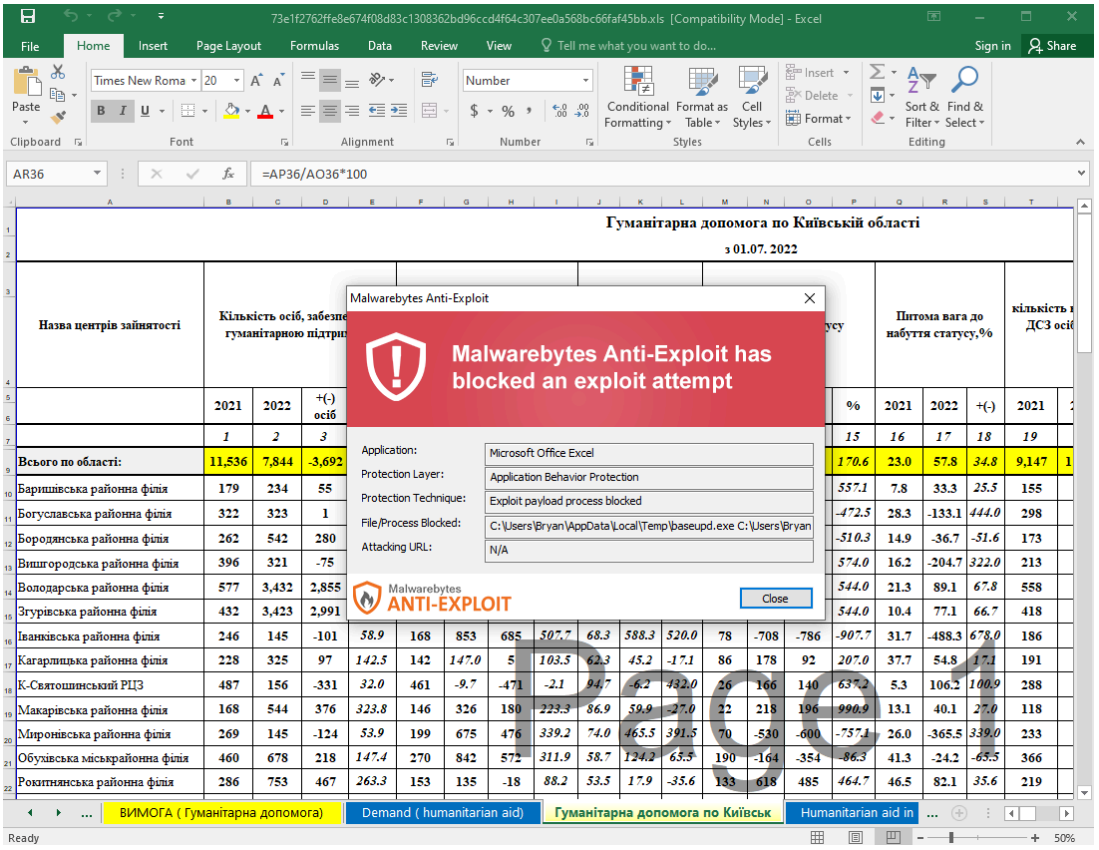
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

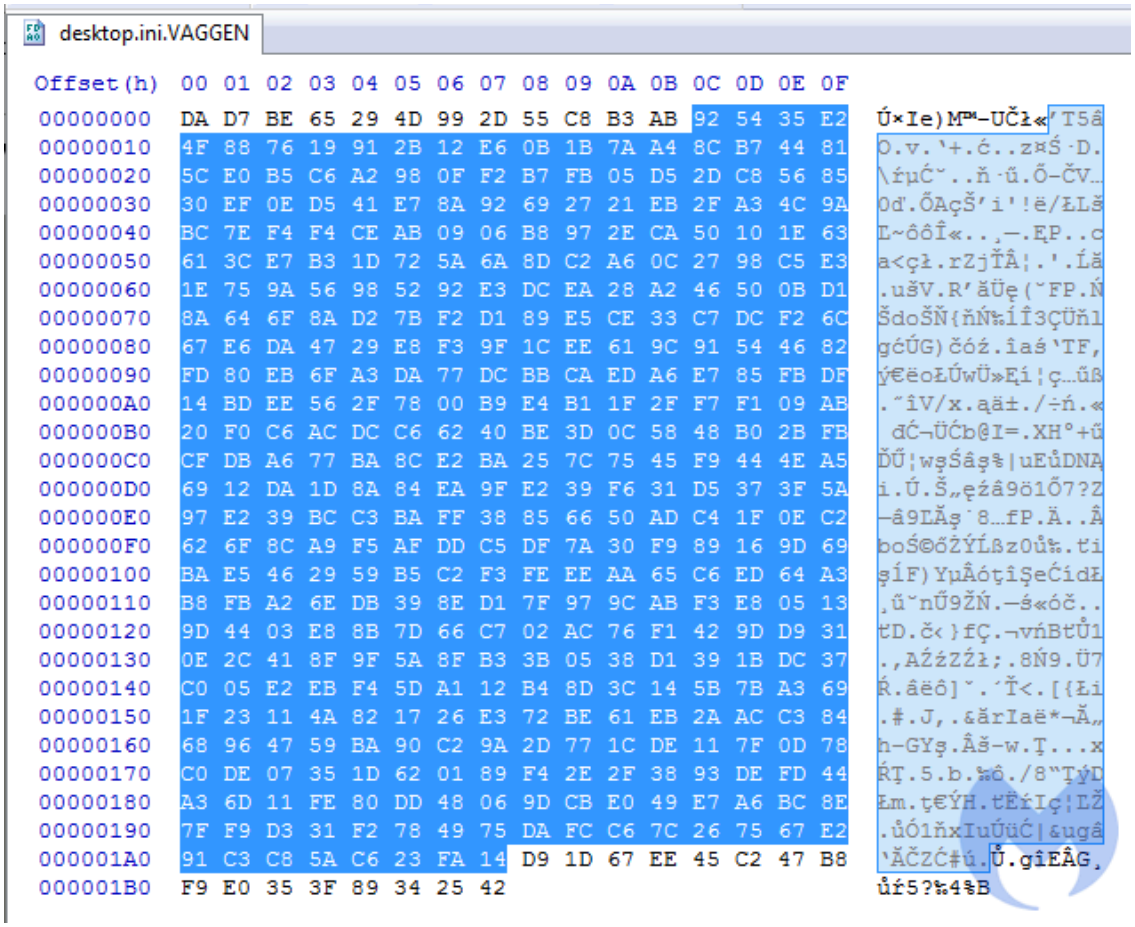
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



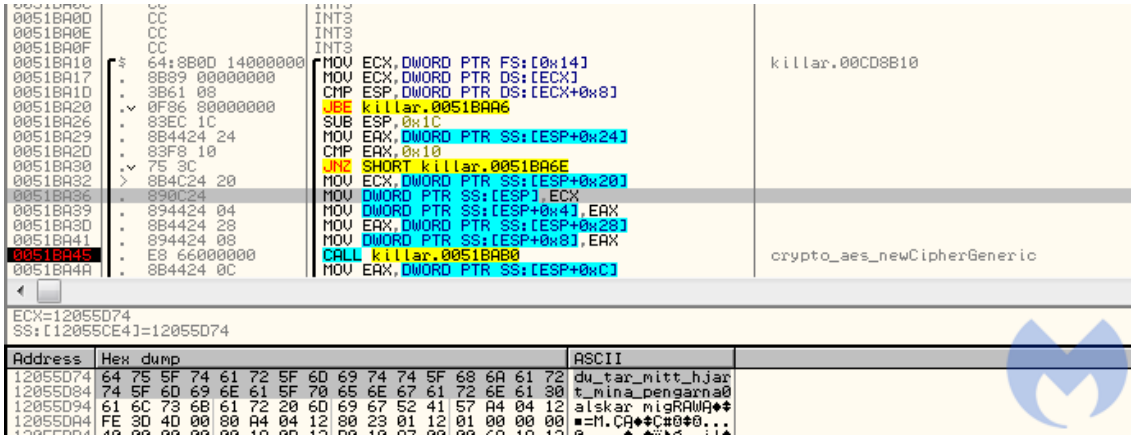
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

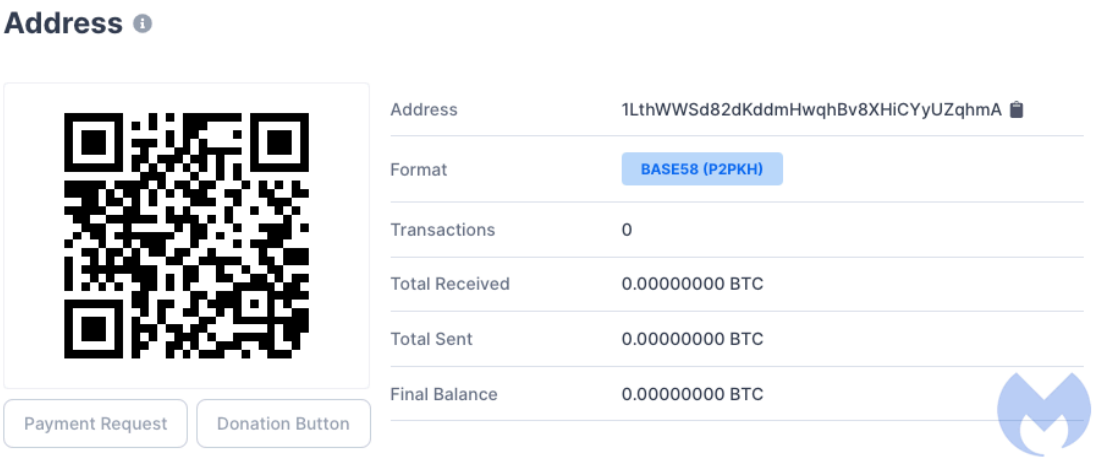
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

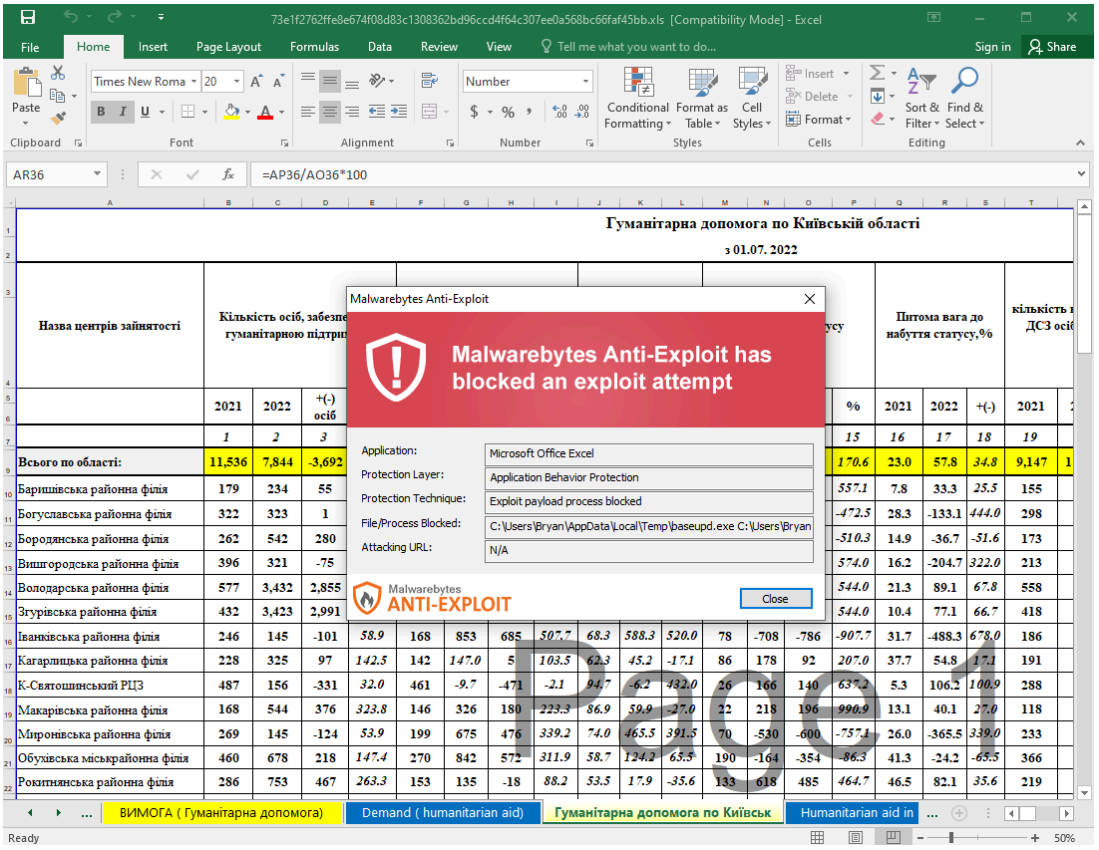
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

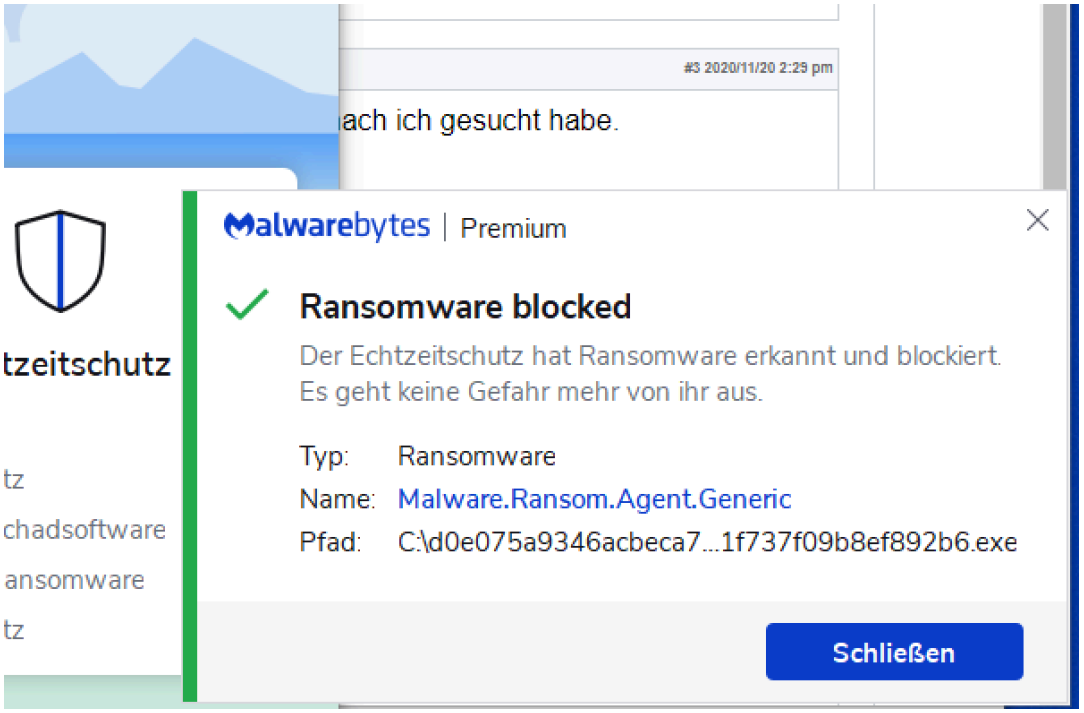
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

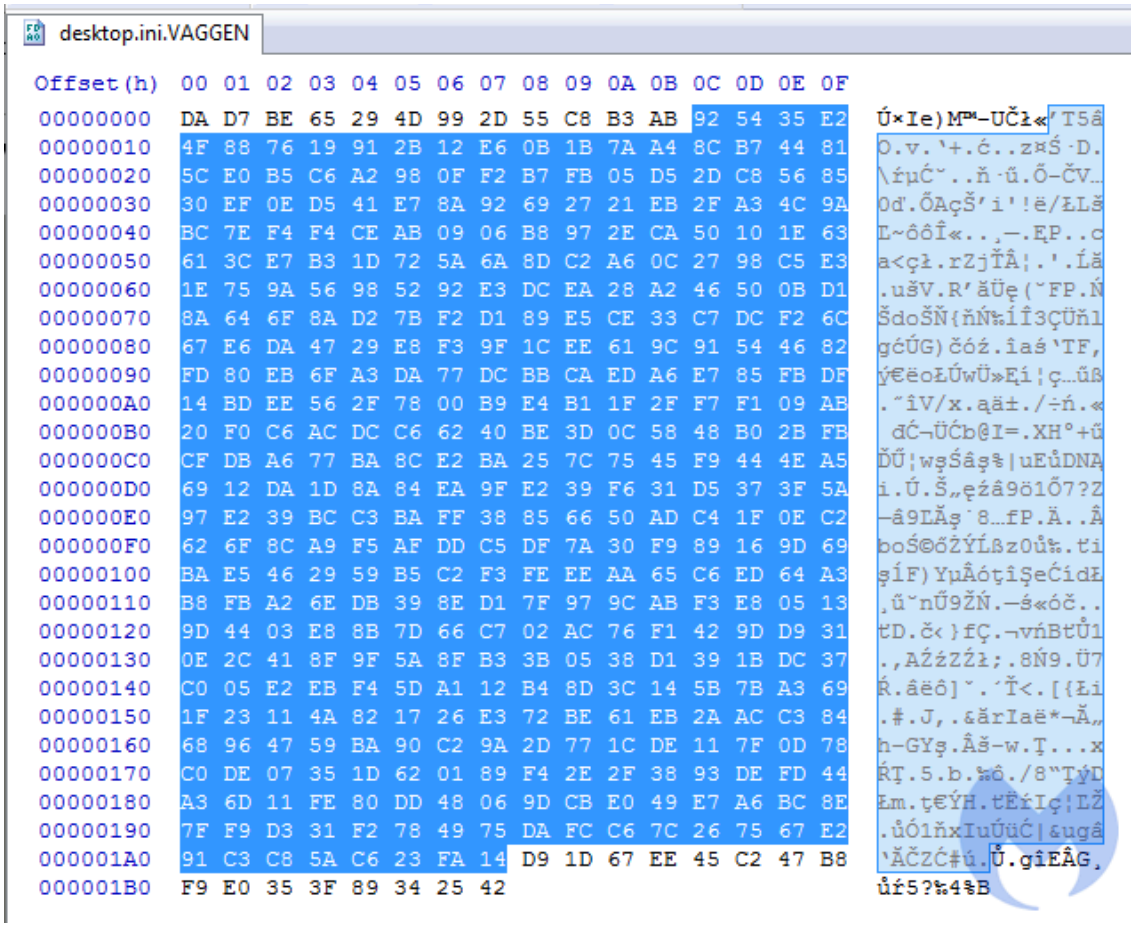
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

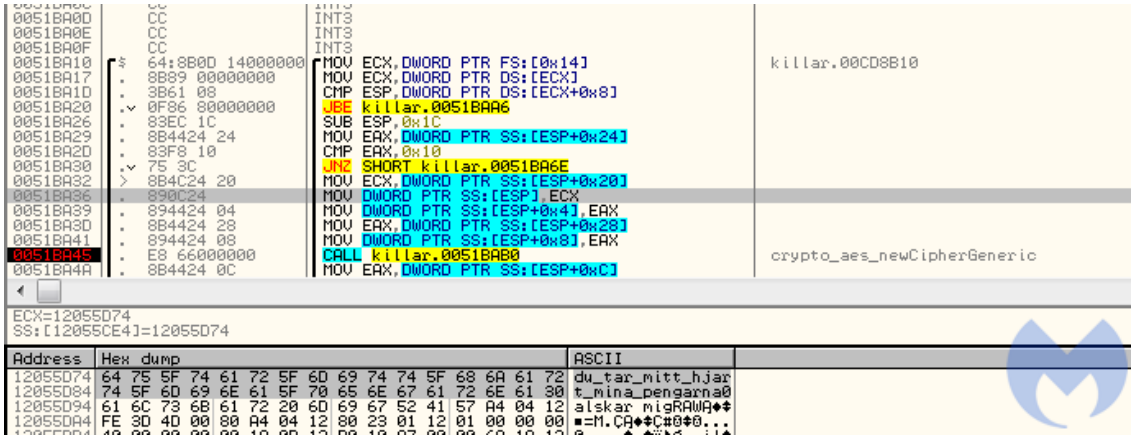


The content of the output file (with .VAGGEN extension) contains:

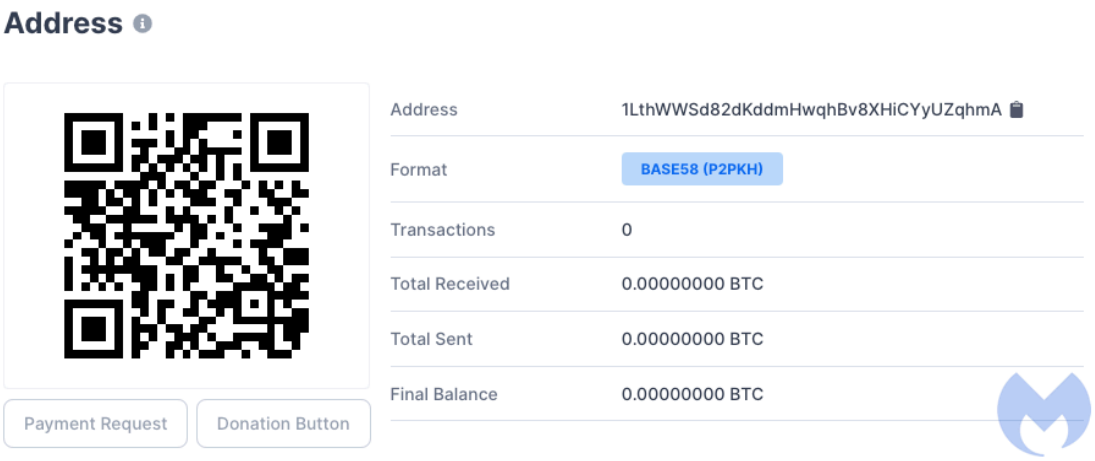
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

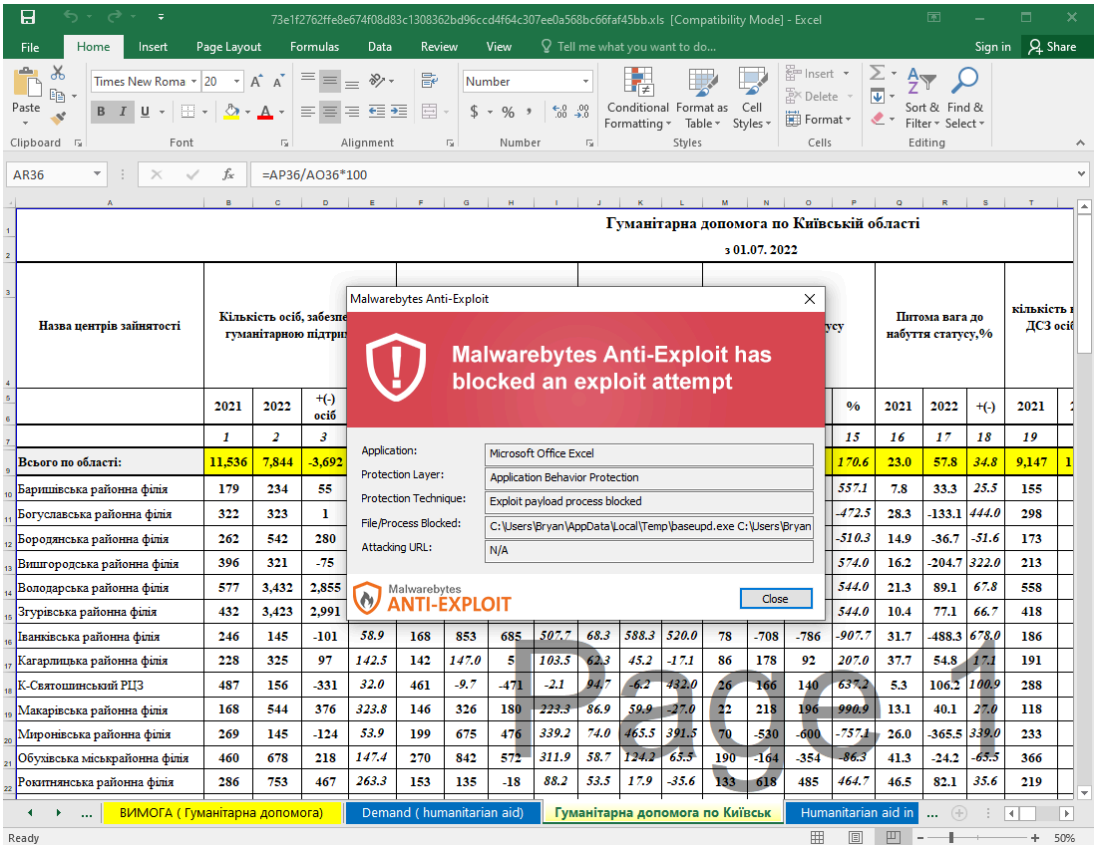
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

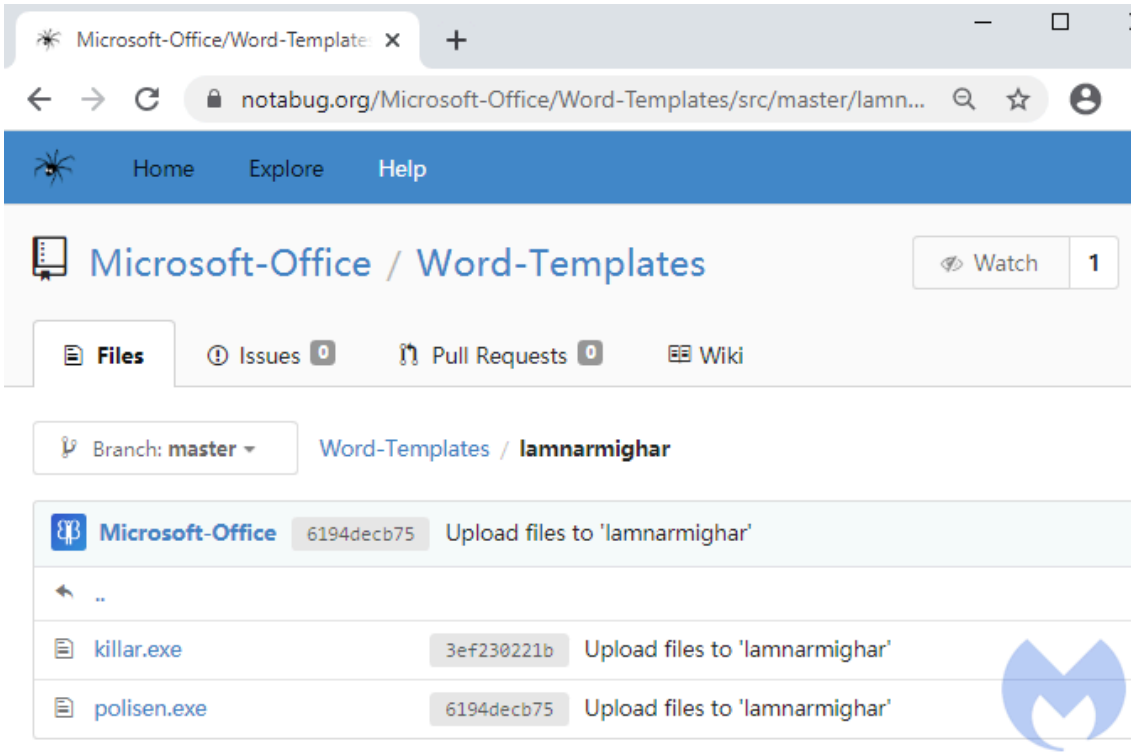
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



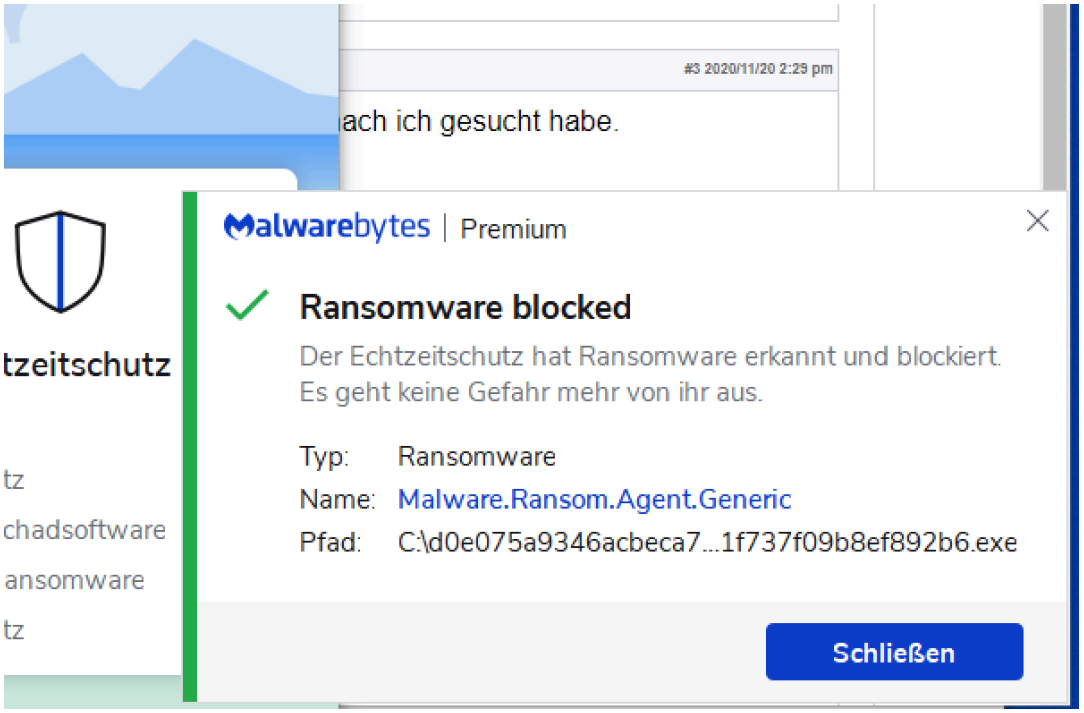
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

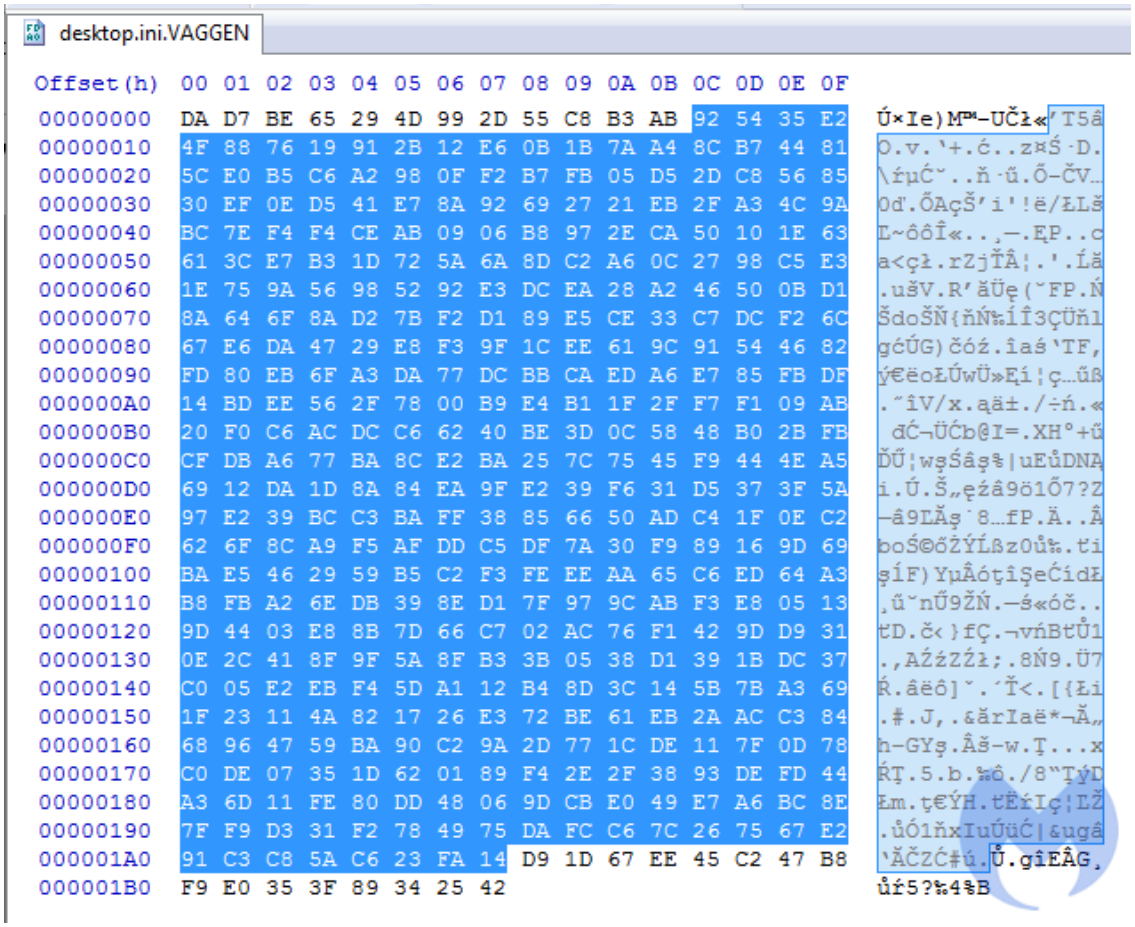
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



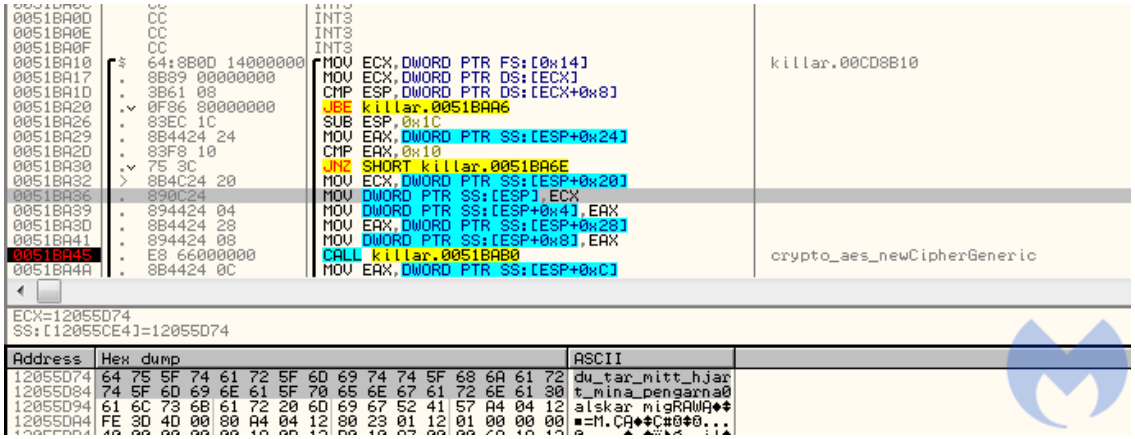
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

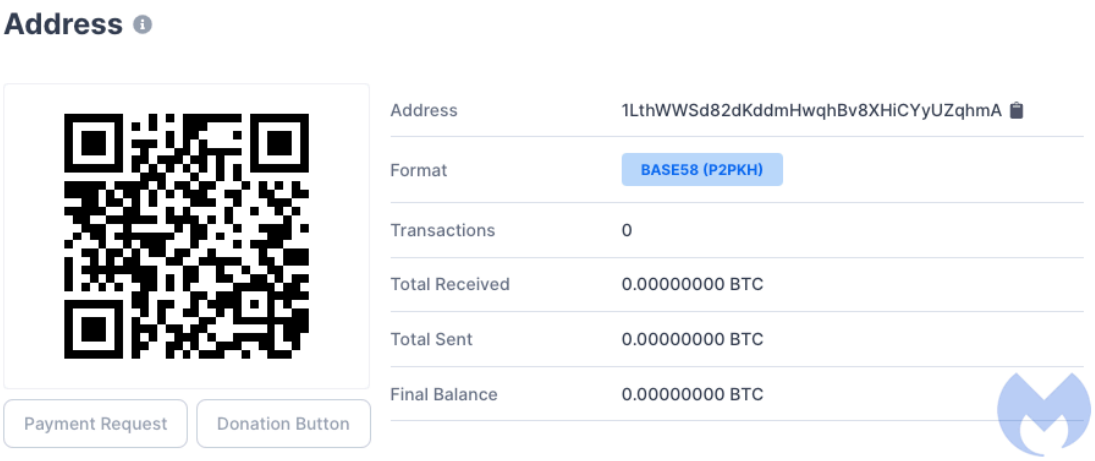
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

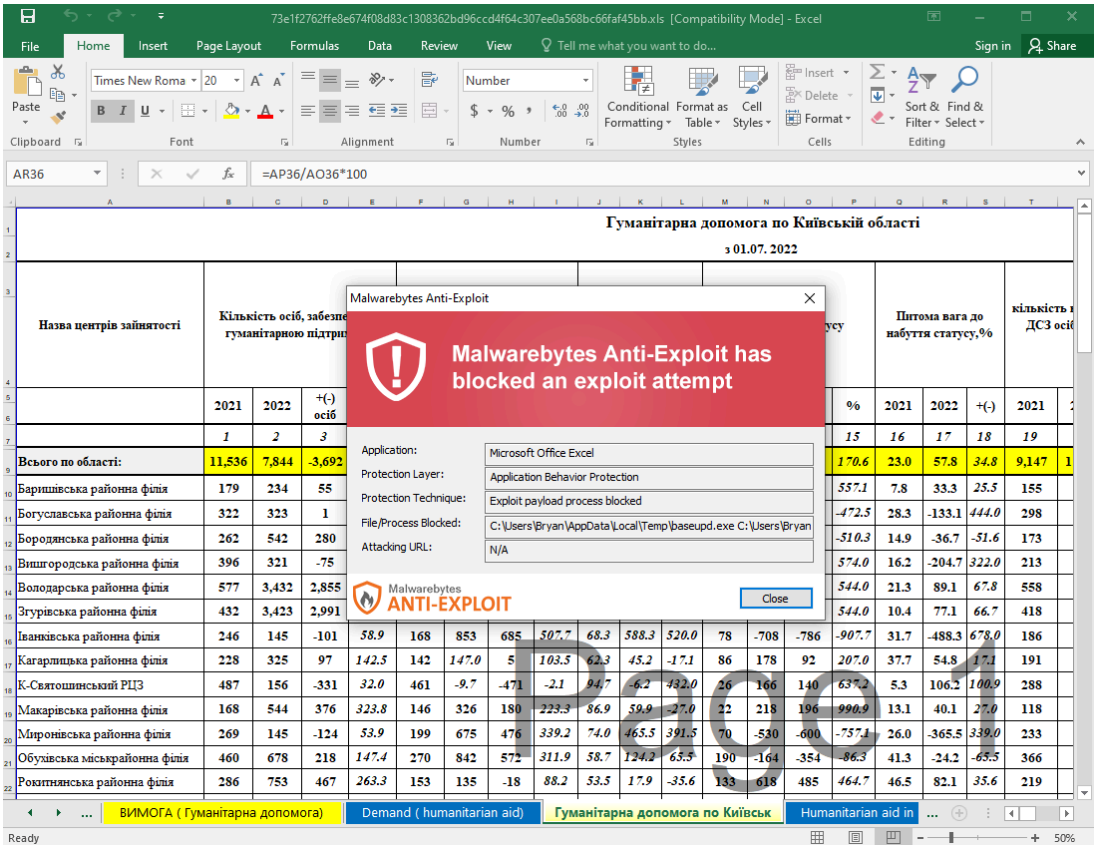
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

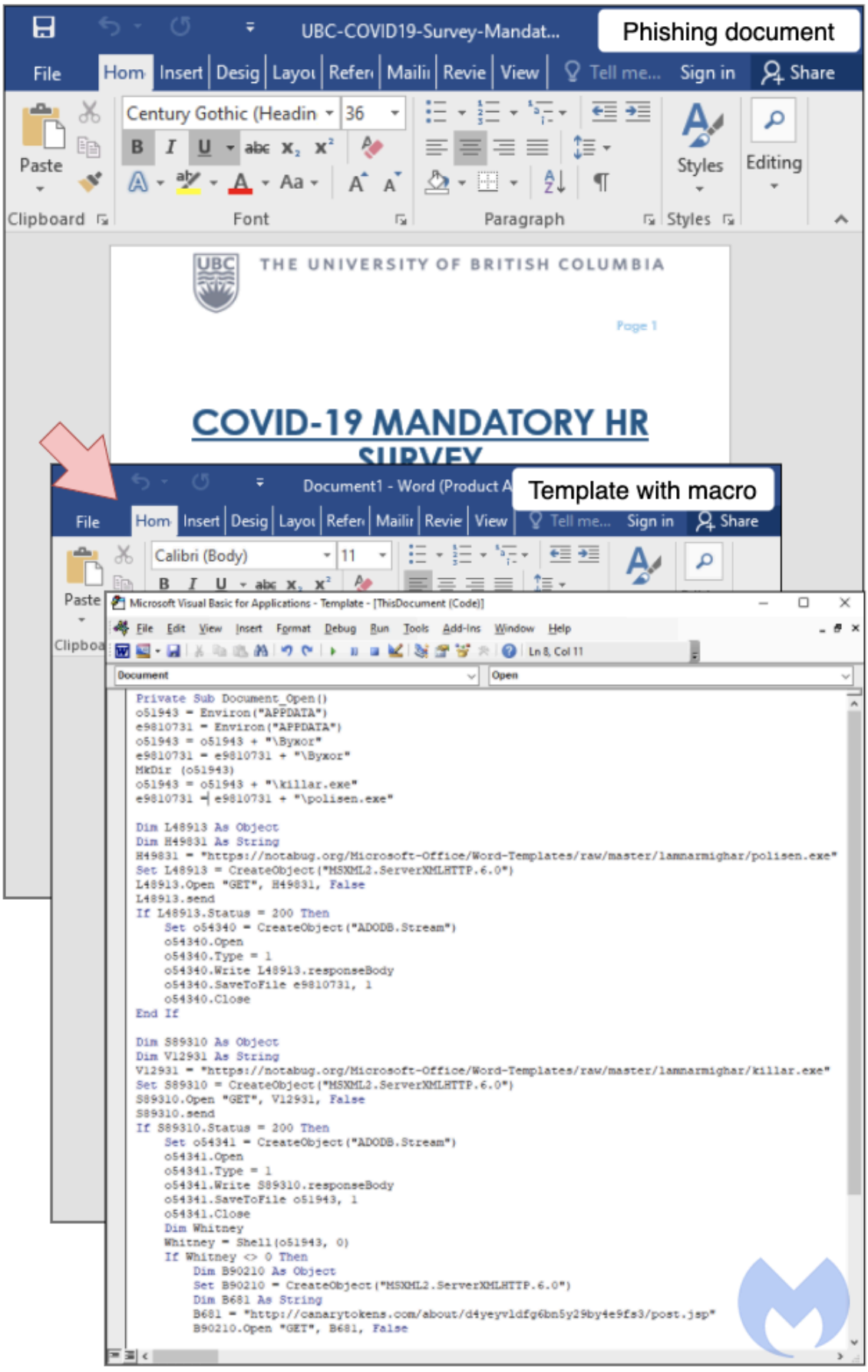
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

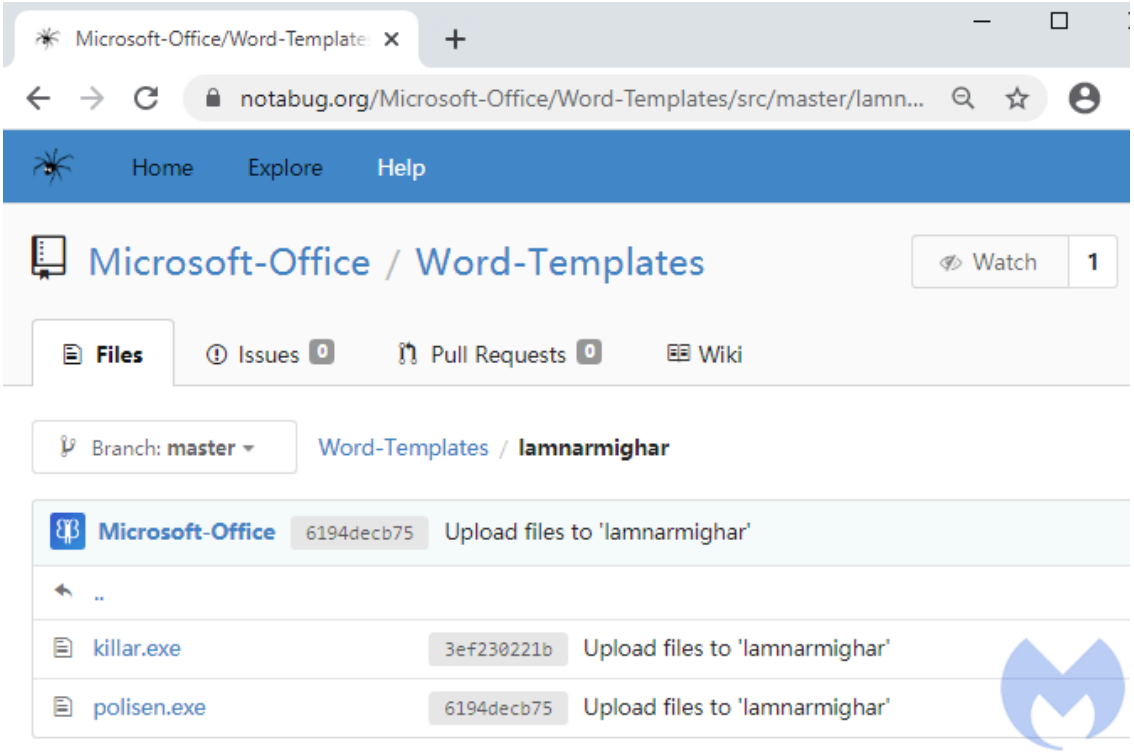
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



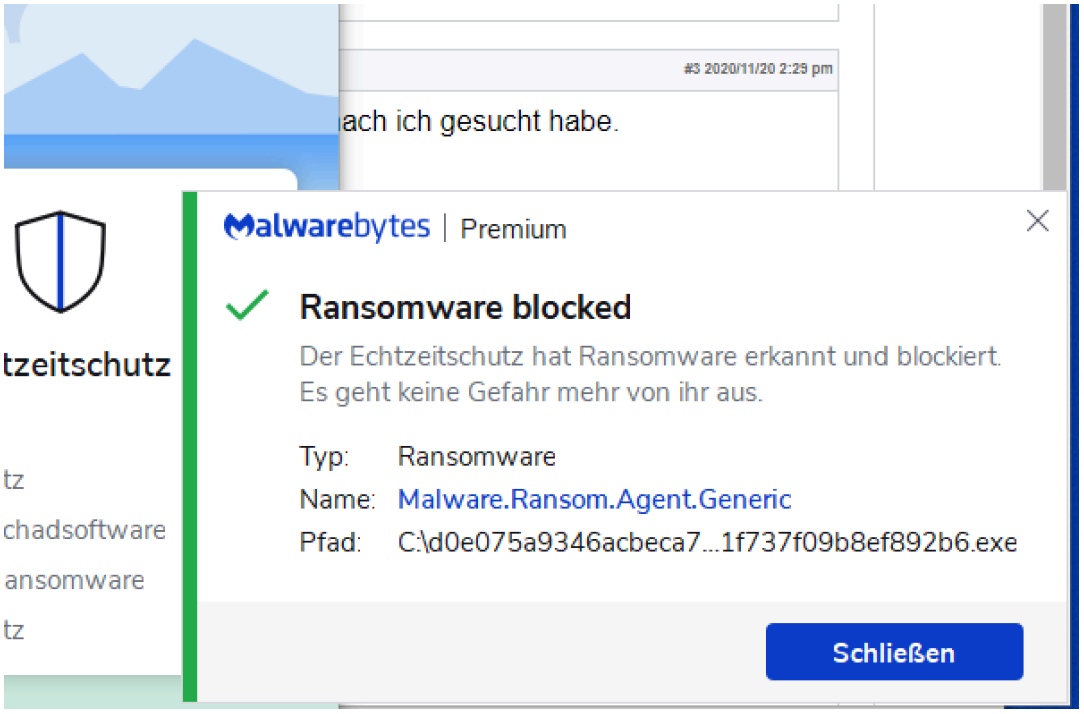
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

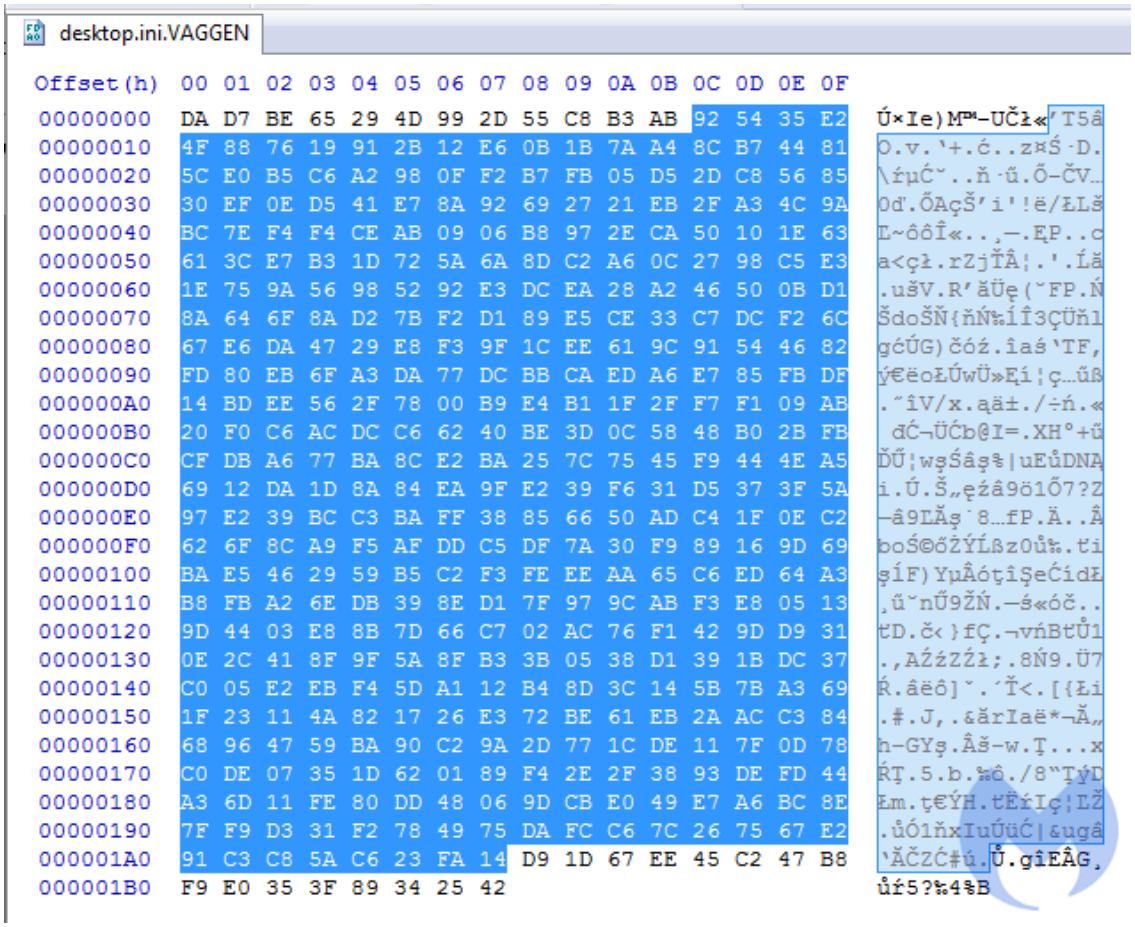
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

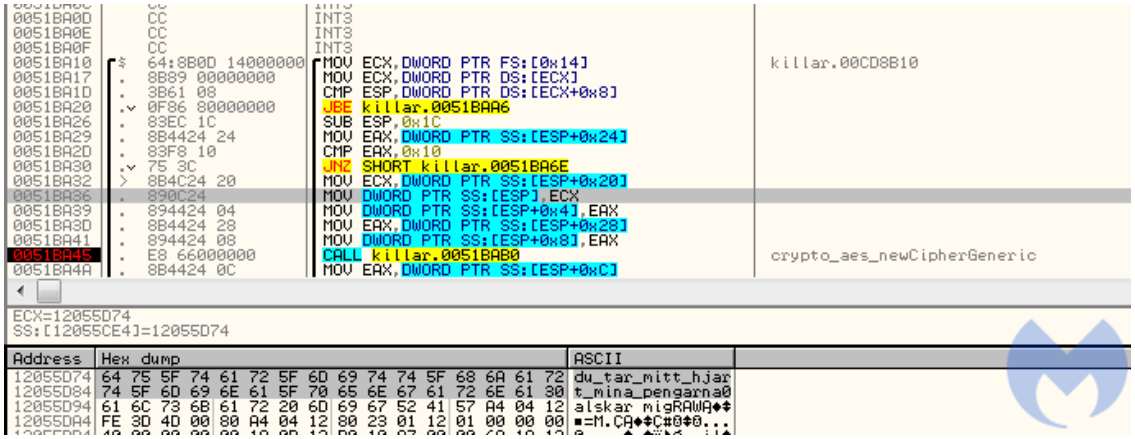


The content of the output file (with .VAGGEN extension) contains:

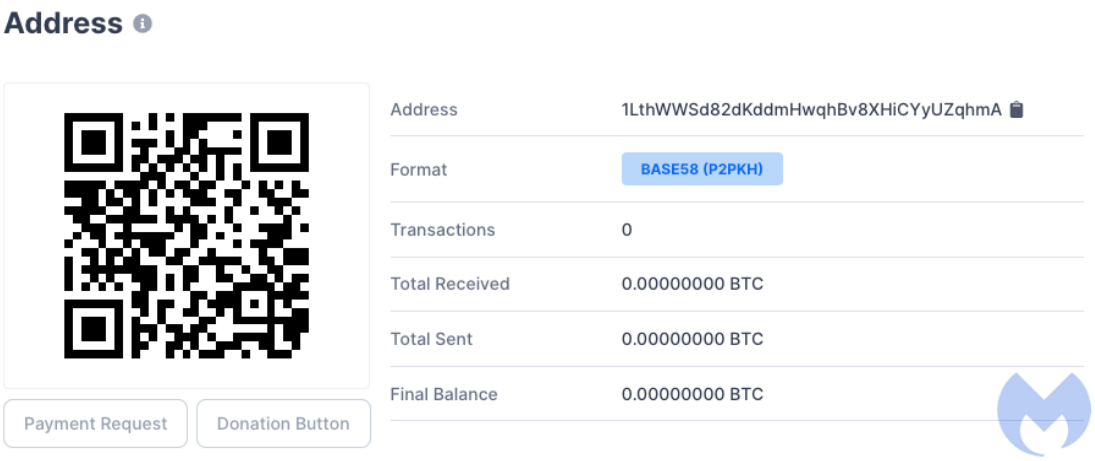
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

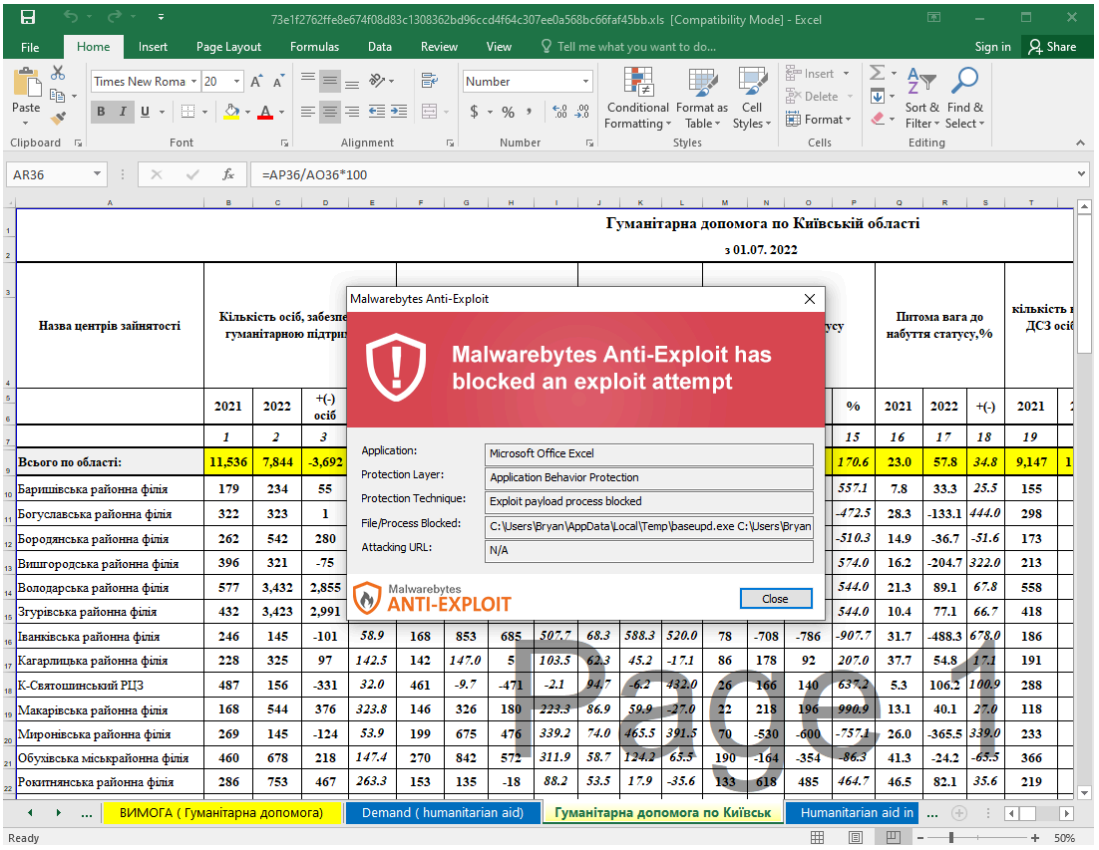
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

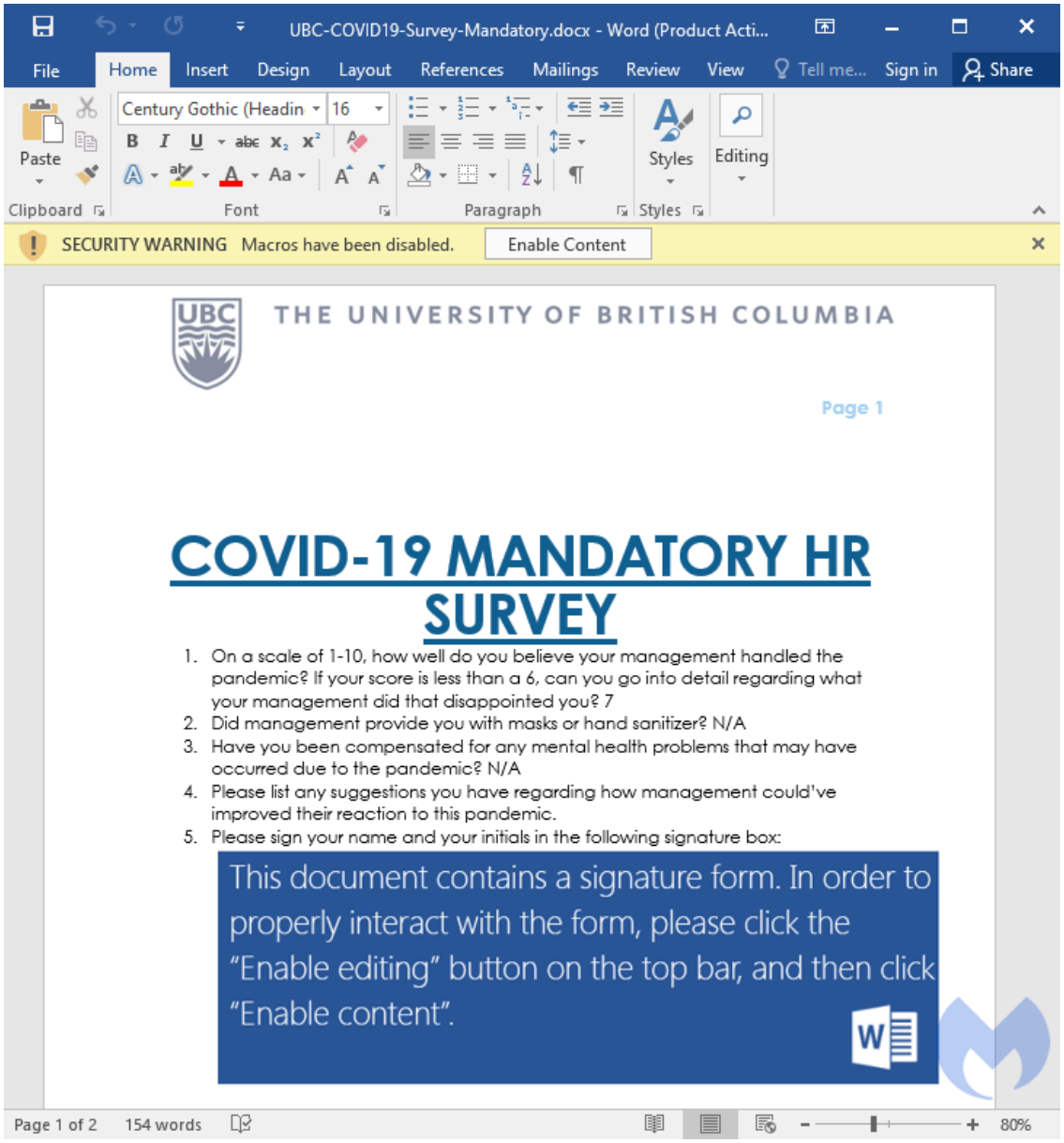
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

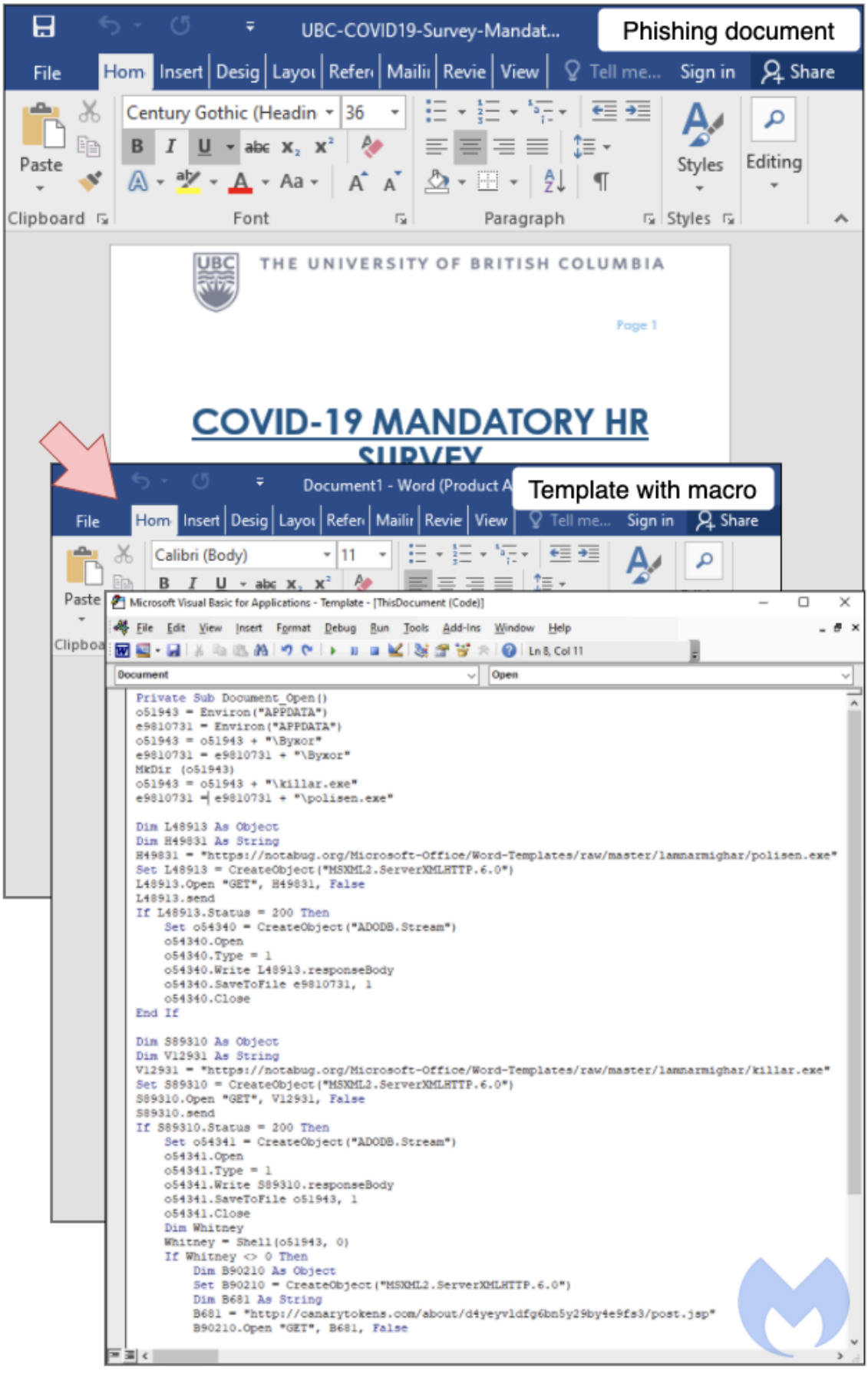
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

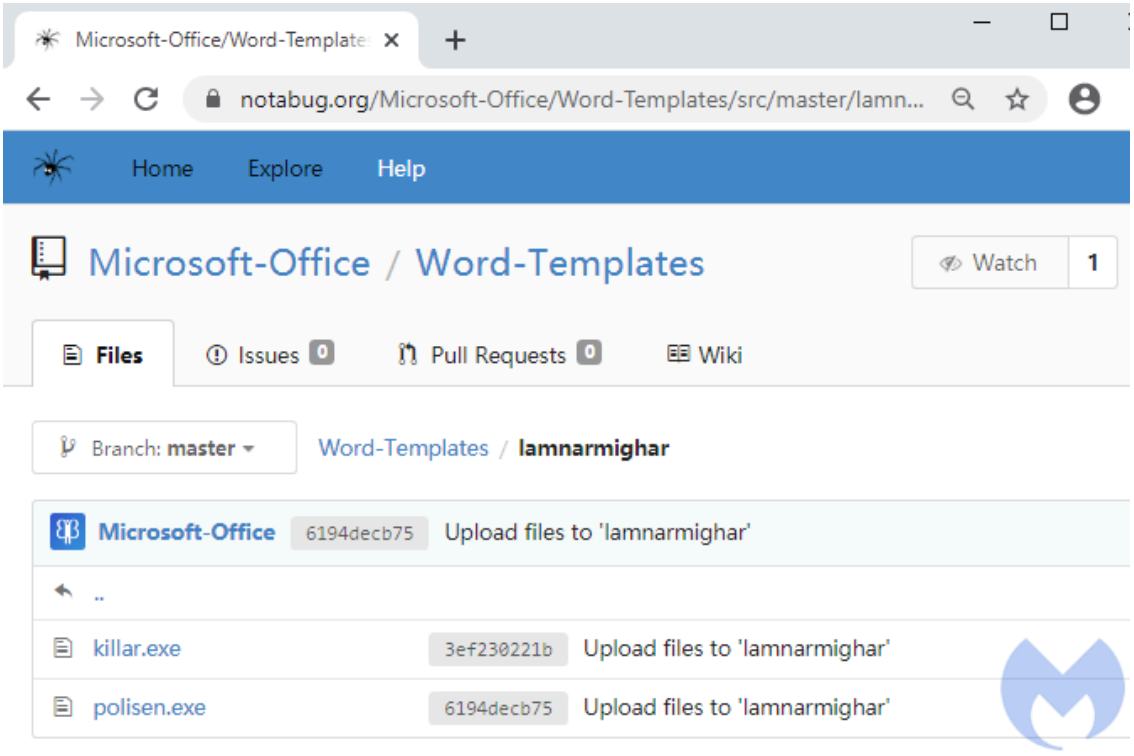
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



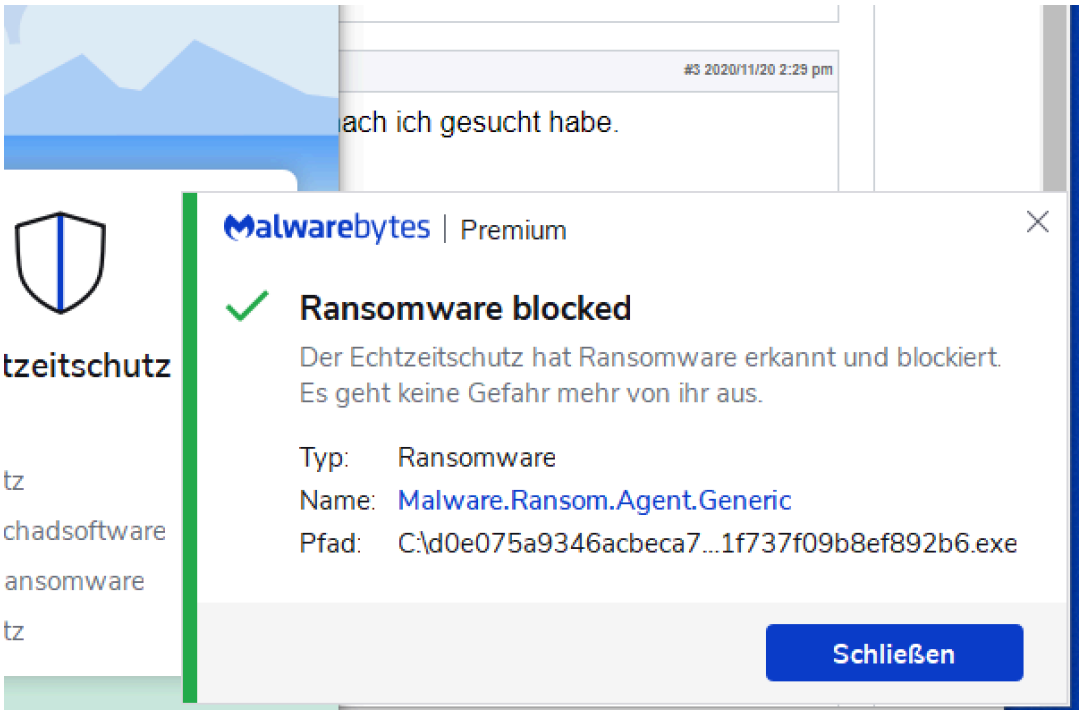
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

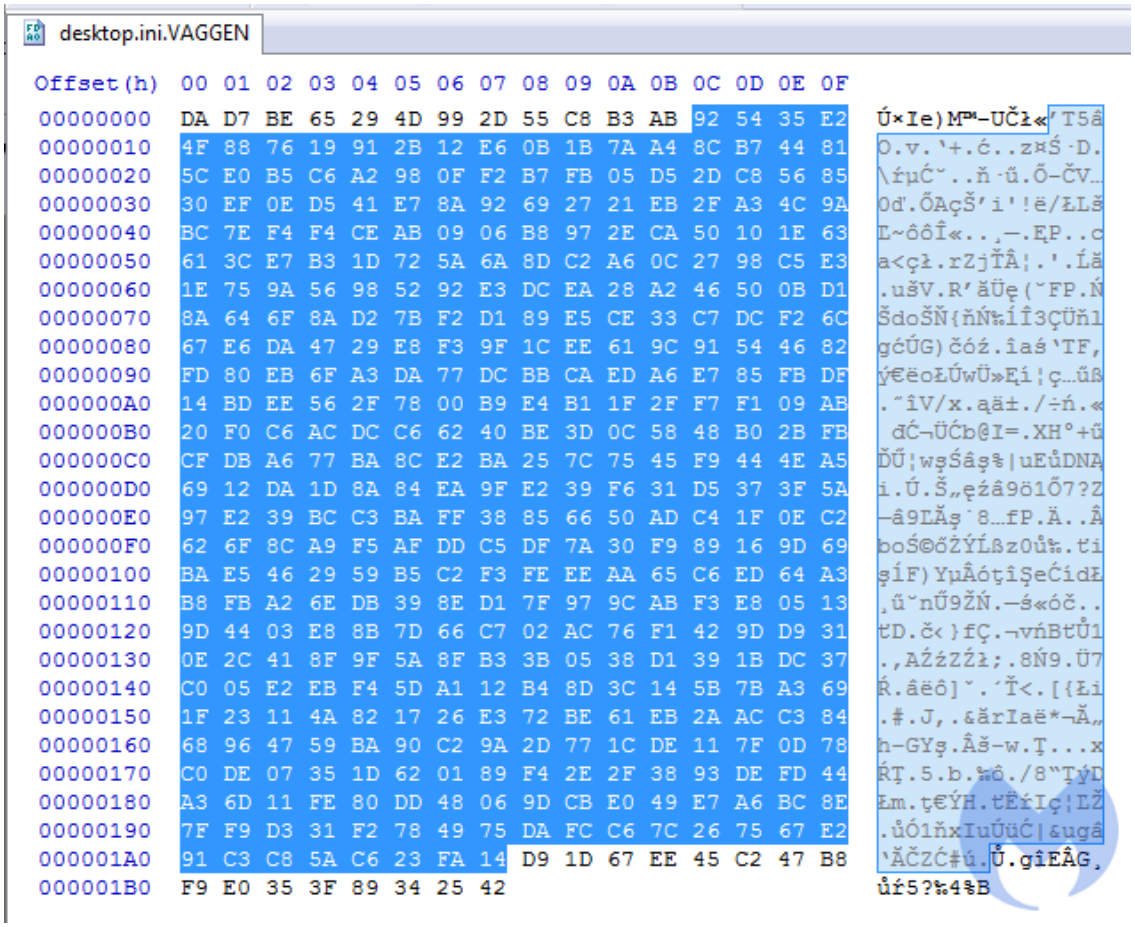
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

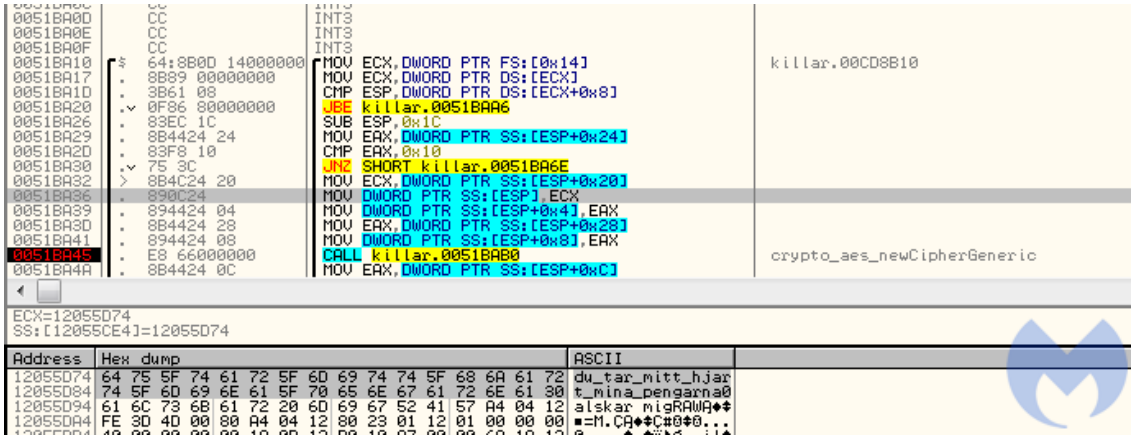


The content of the output file (with .VAGGEN extension) contains:

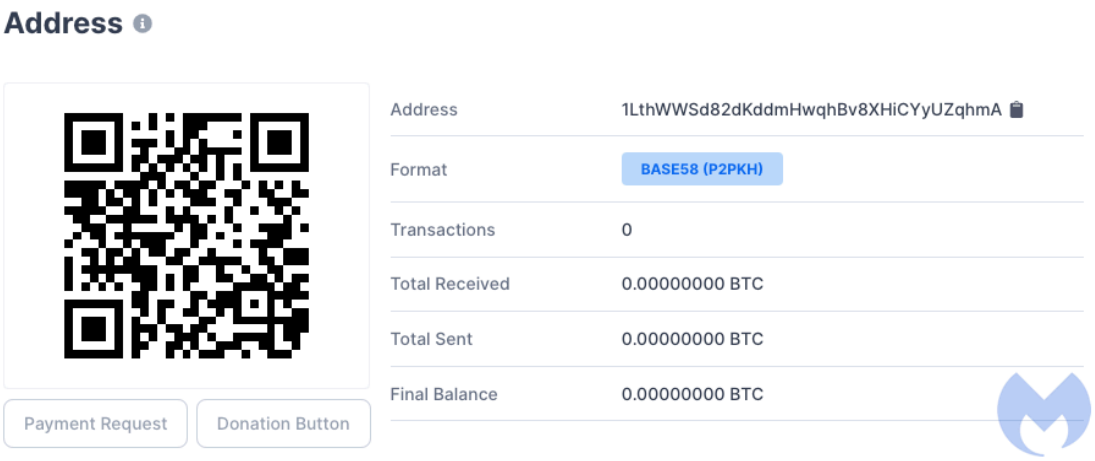
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

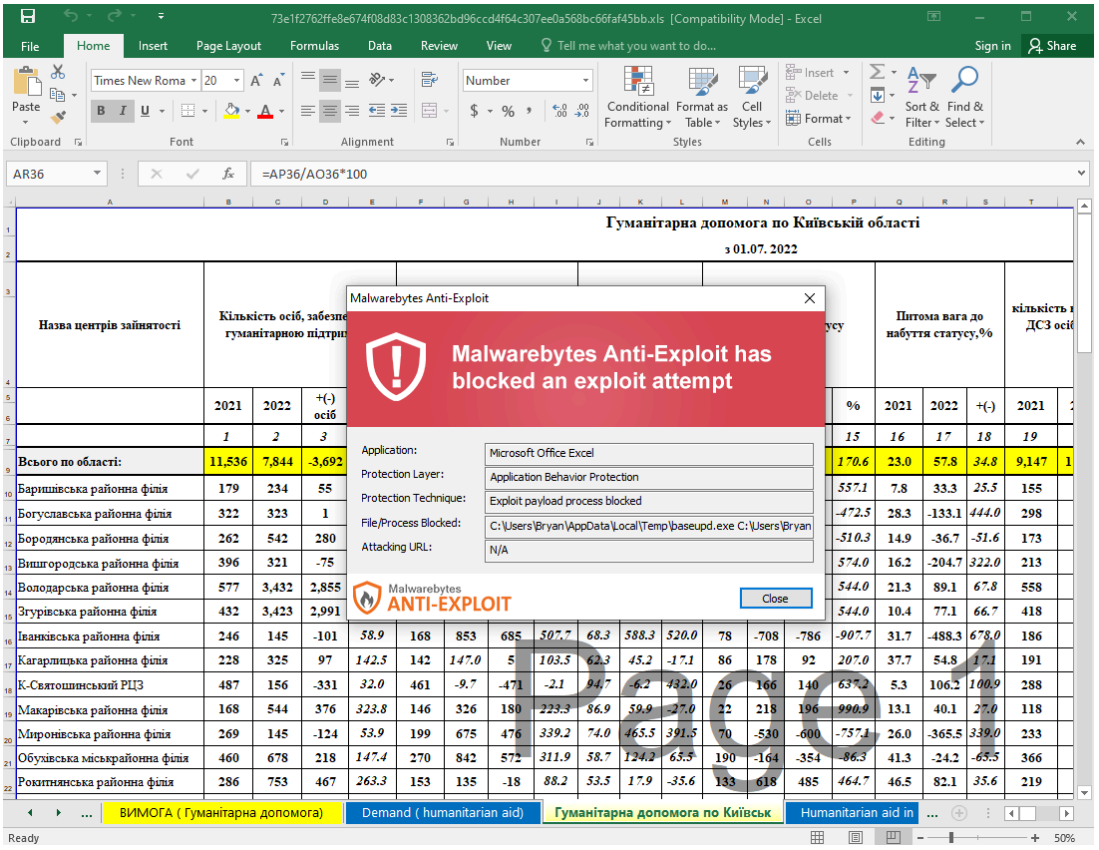
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

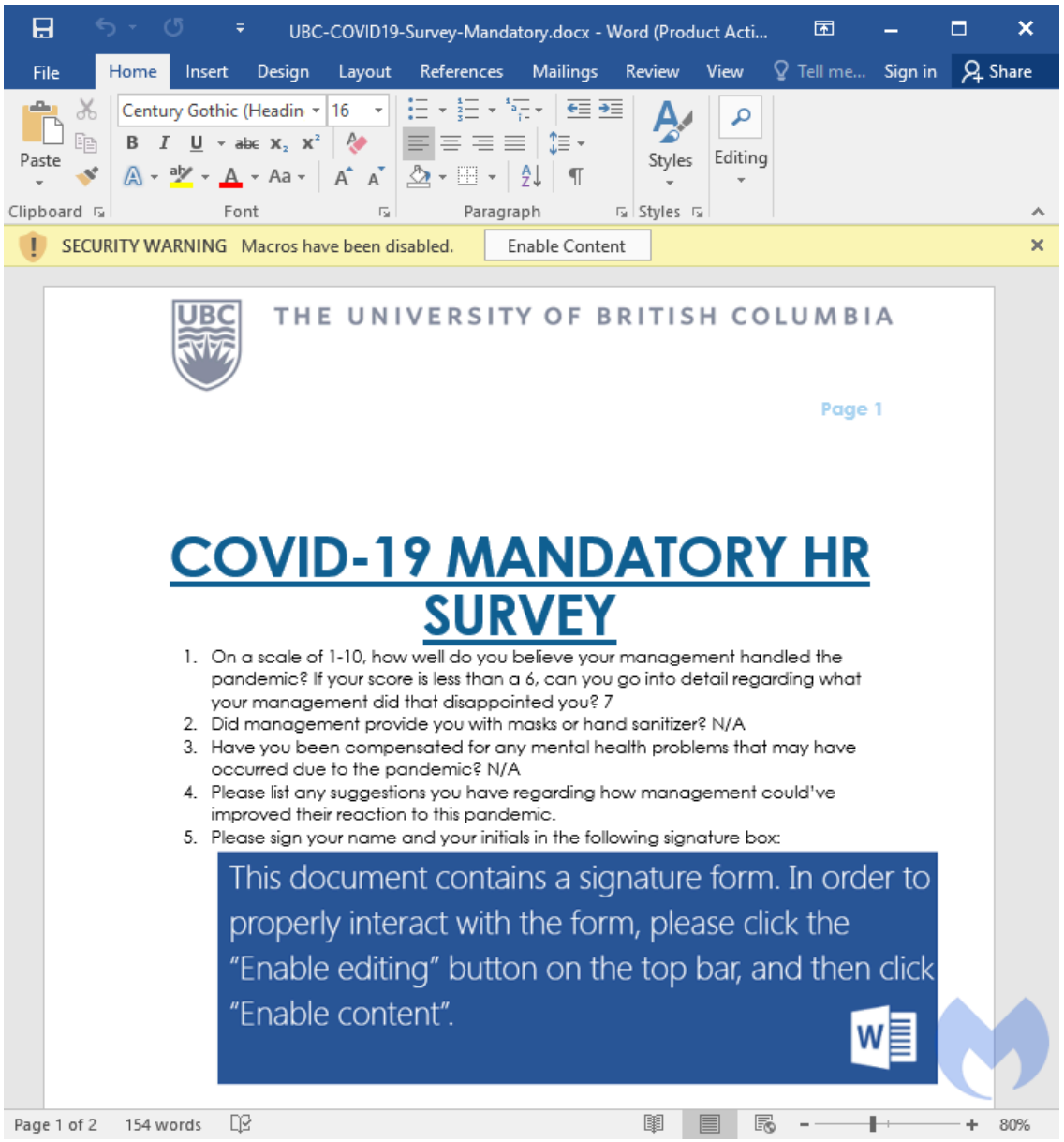
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

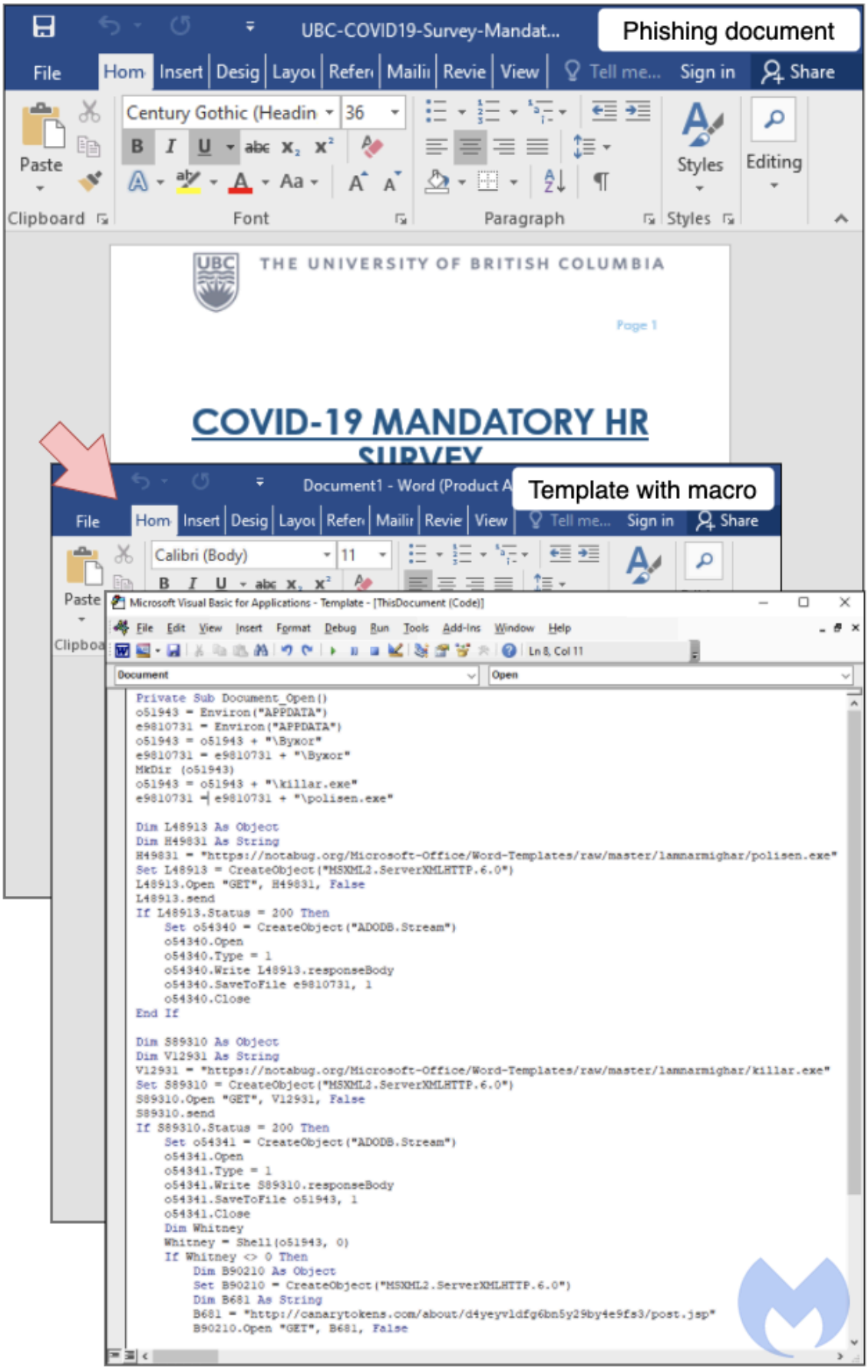
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

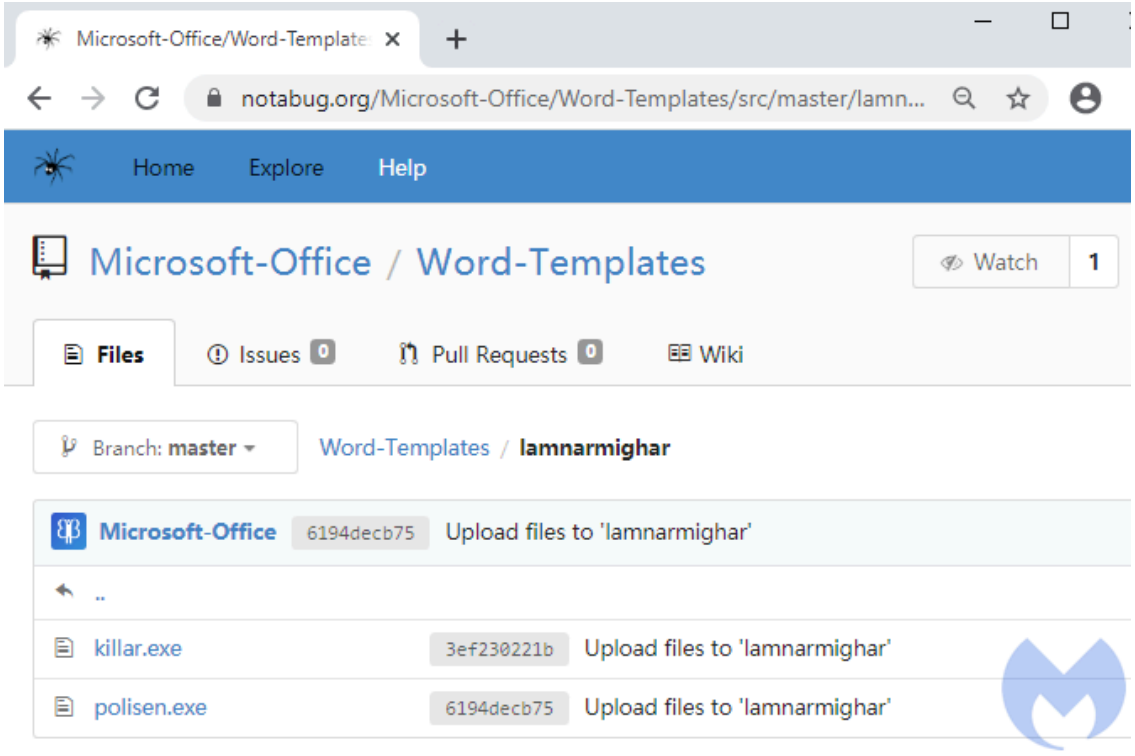
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



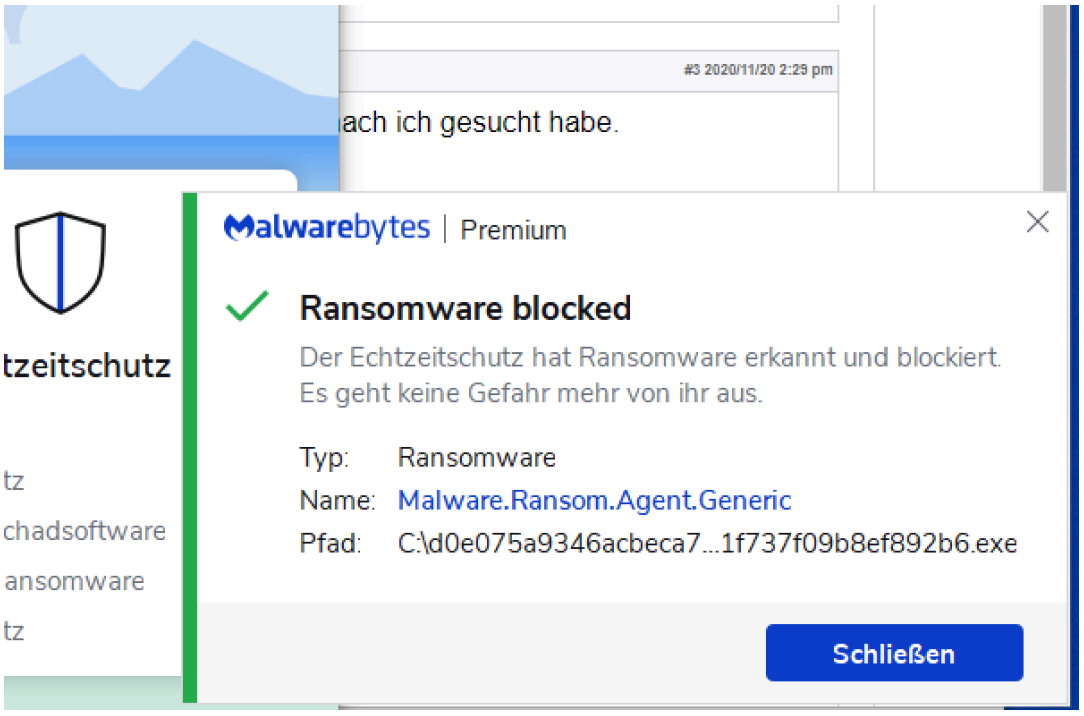
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

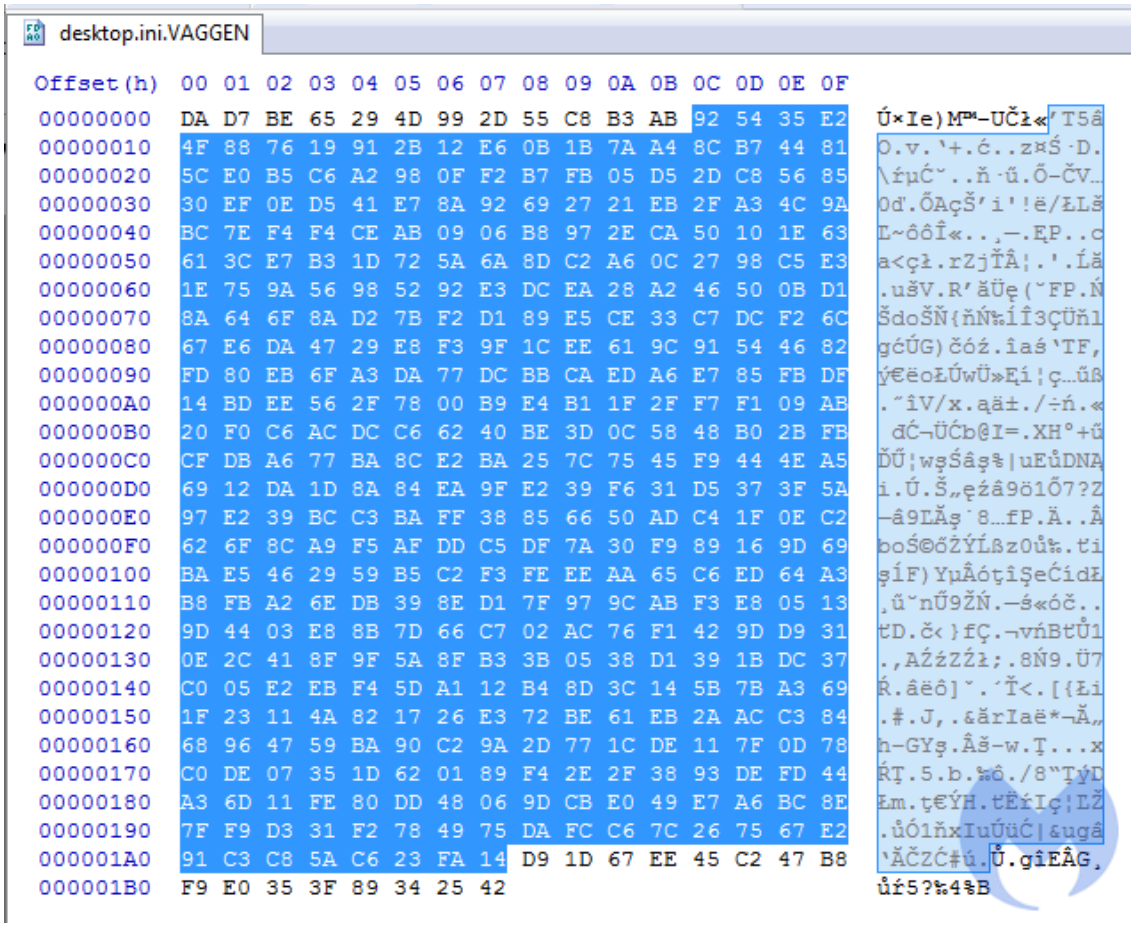
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



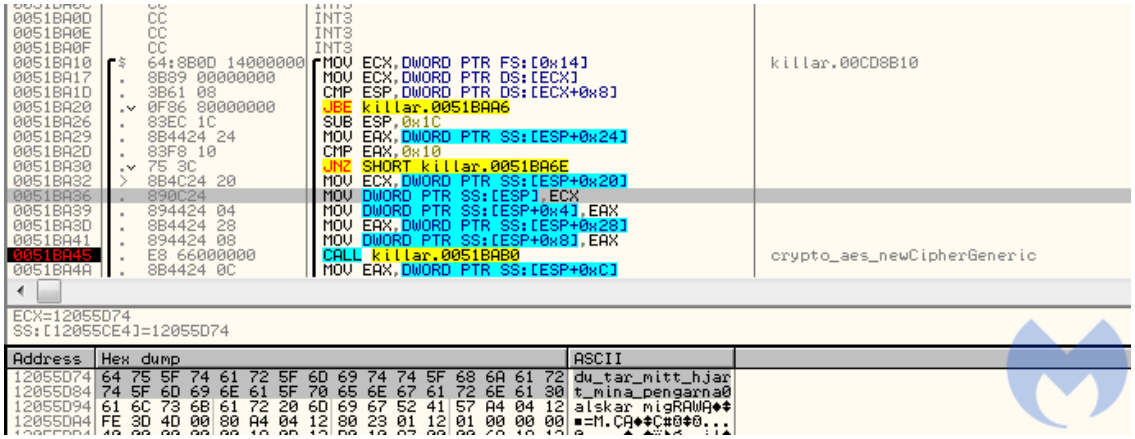
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

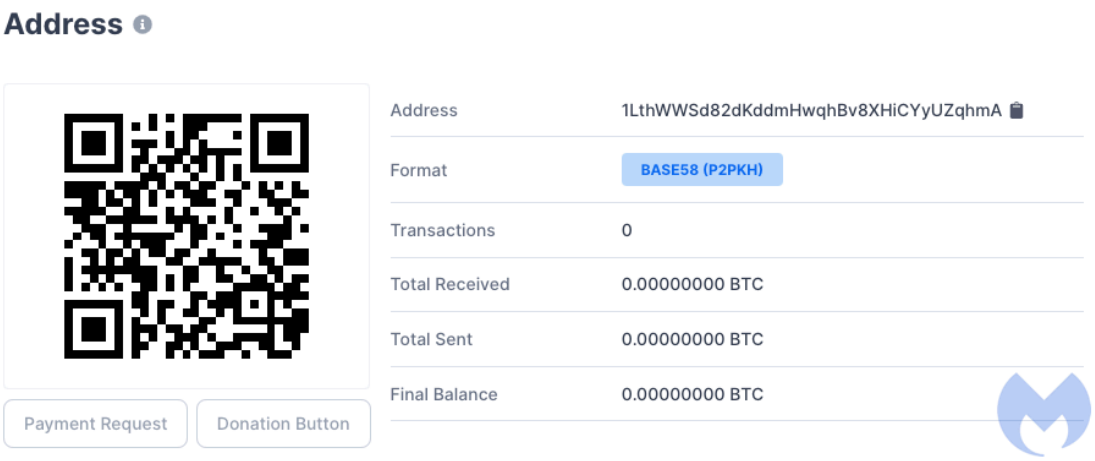
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

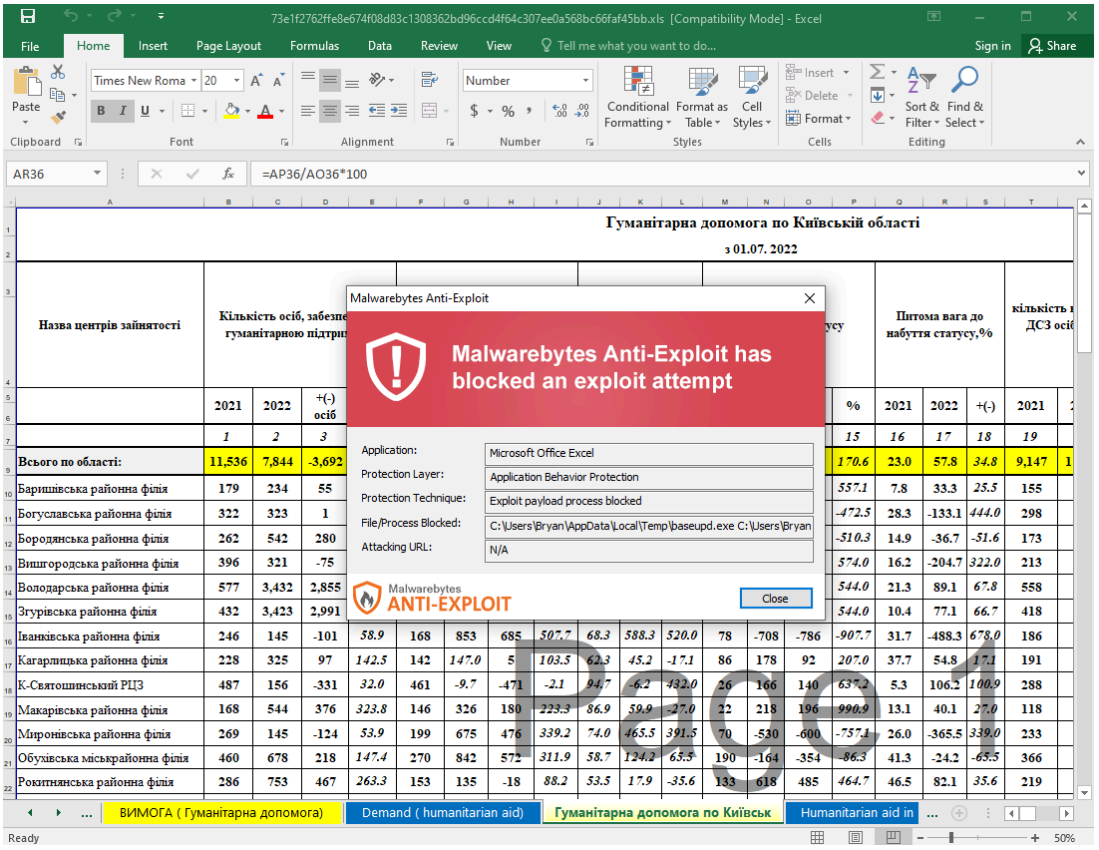
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

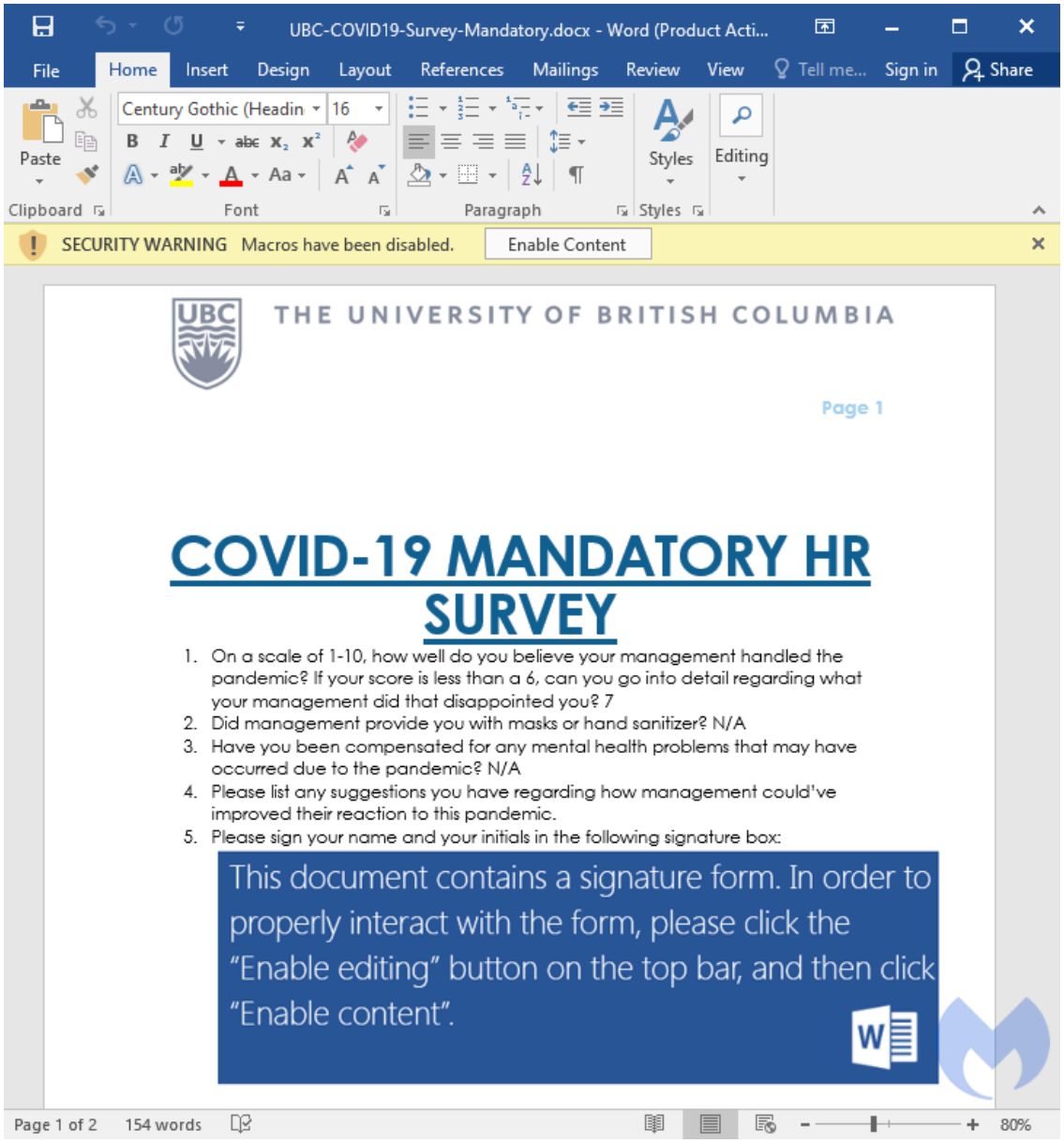
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

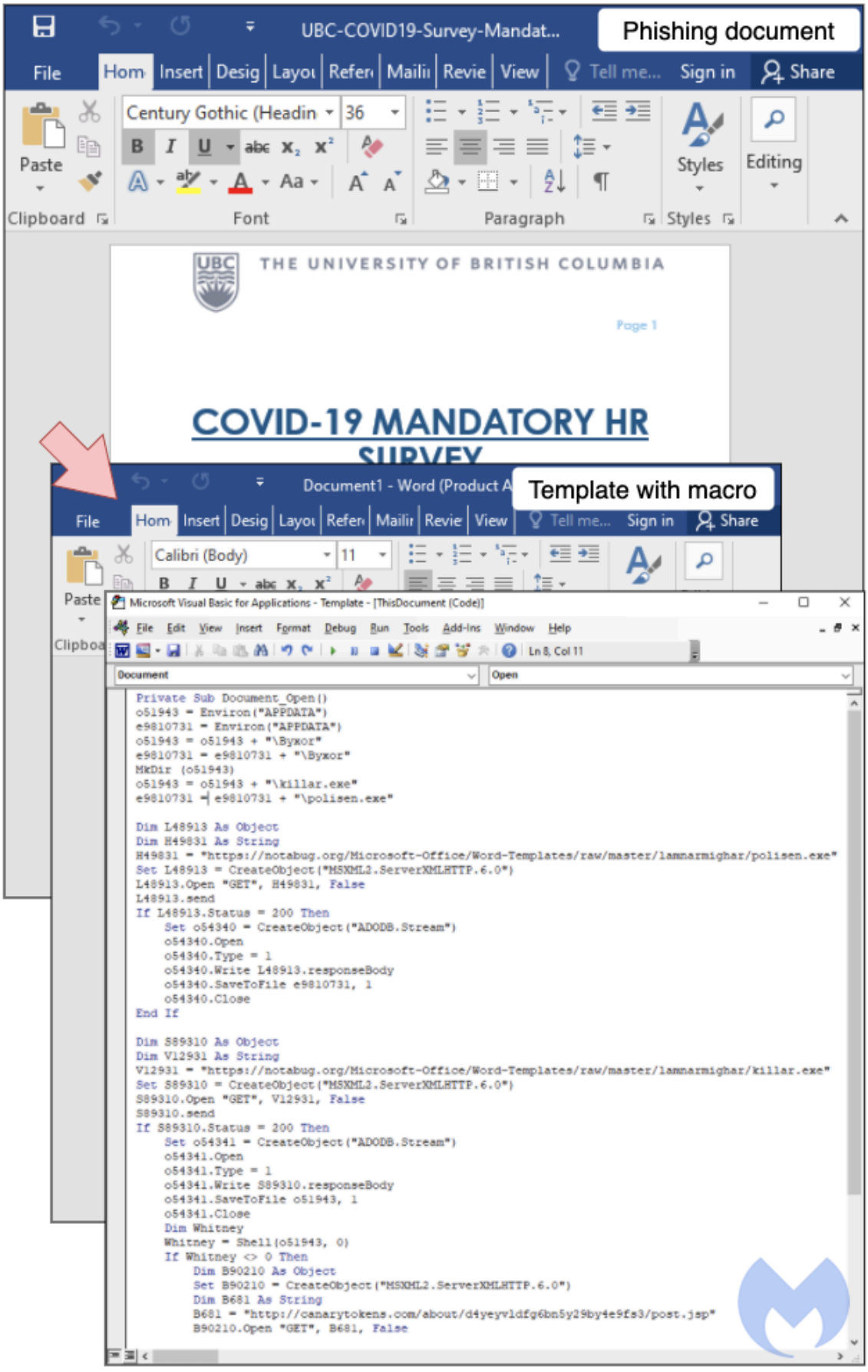
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

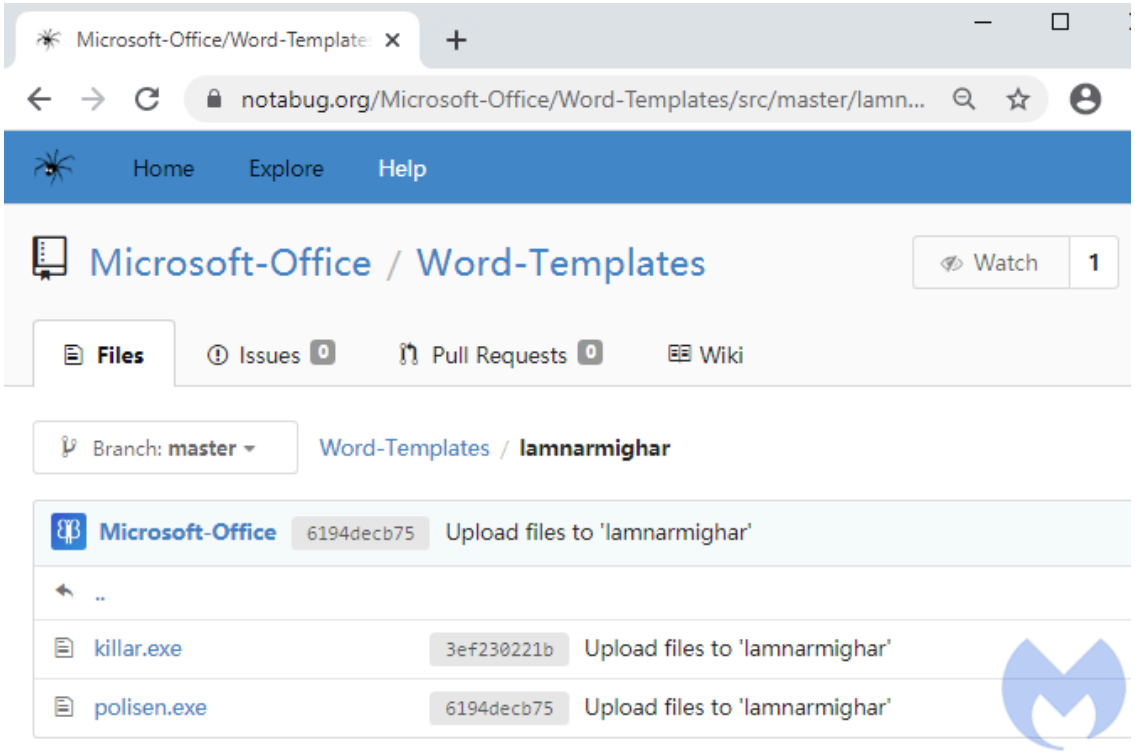
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



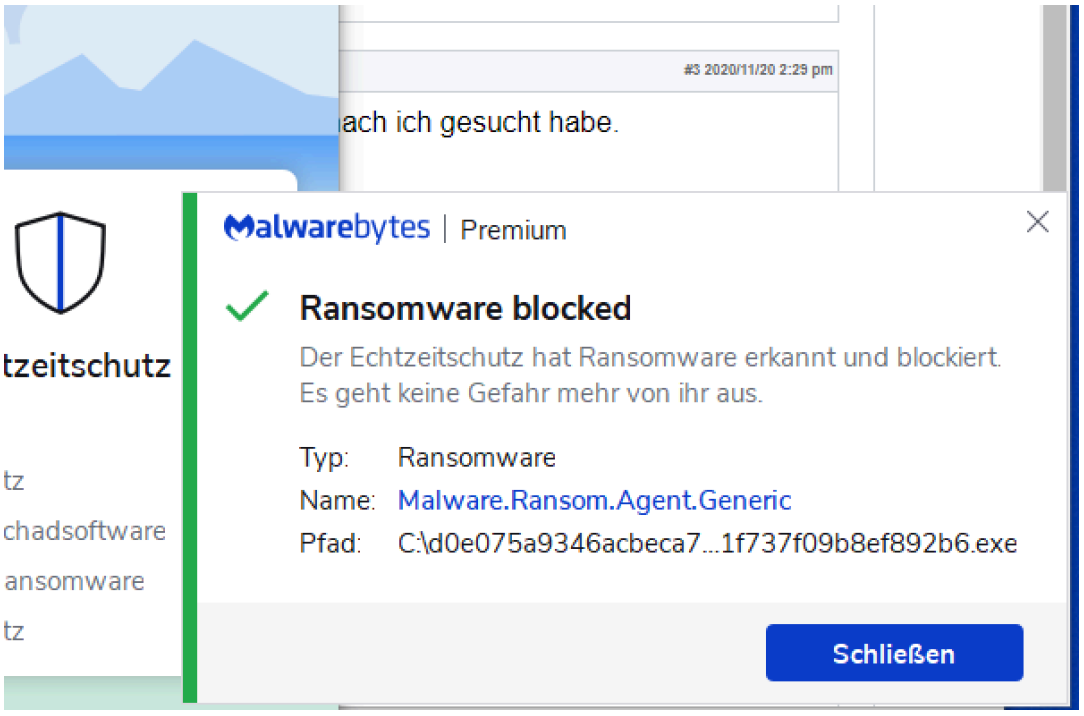
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

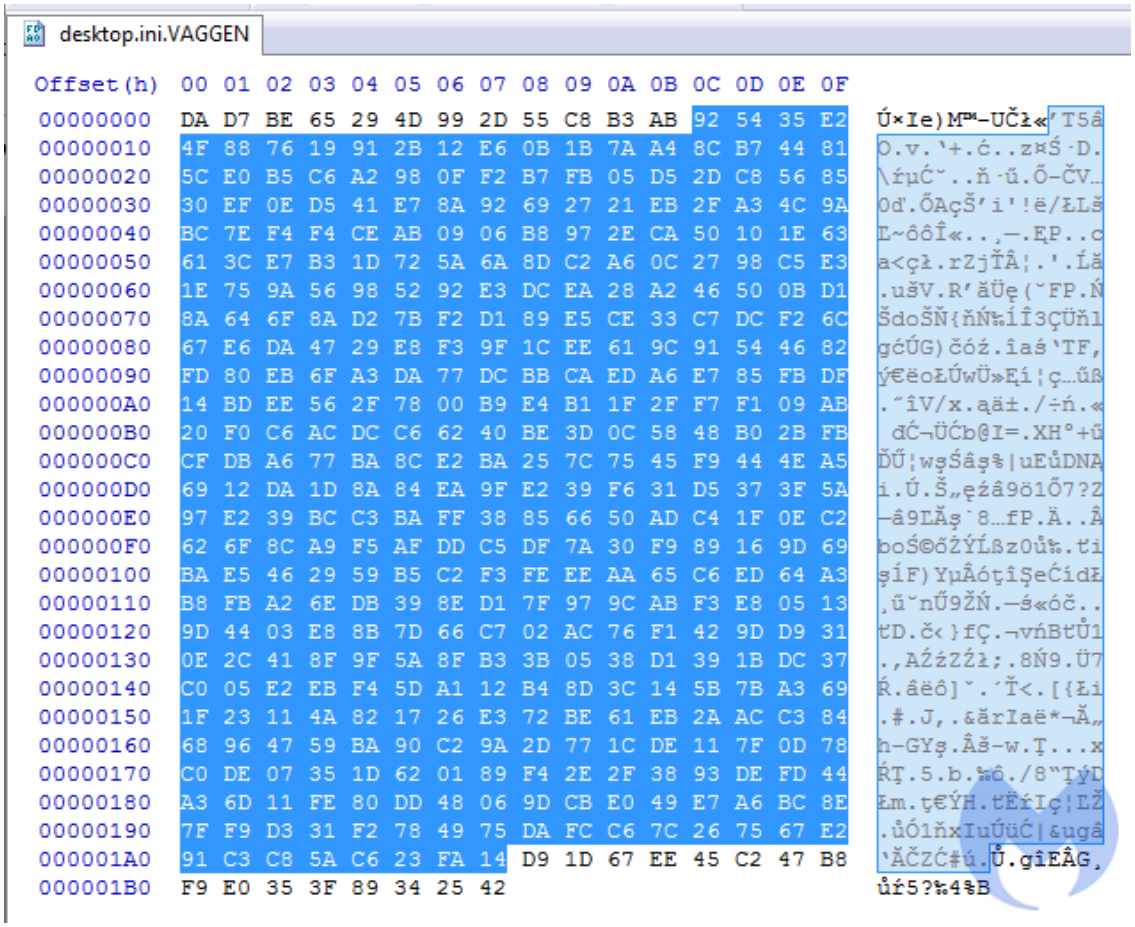
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



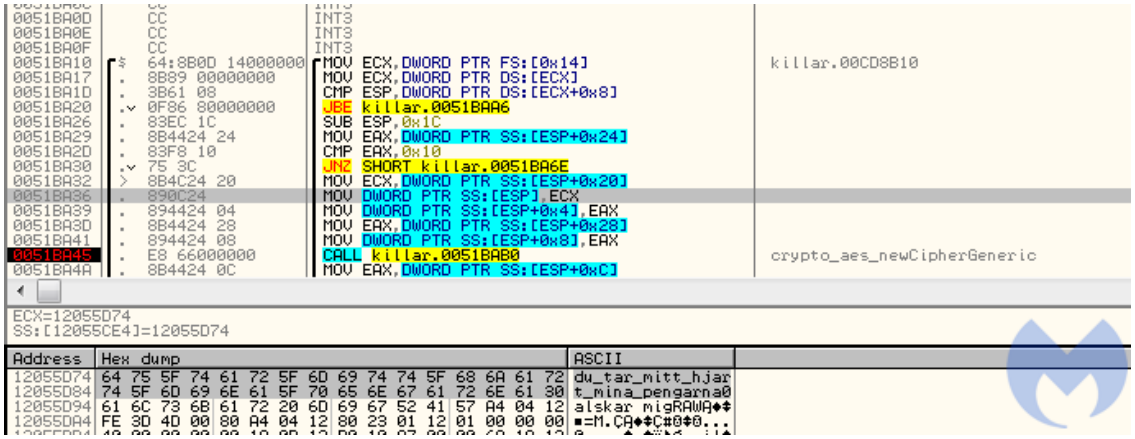
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

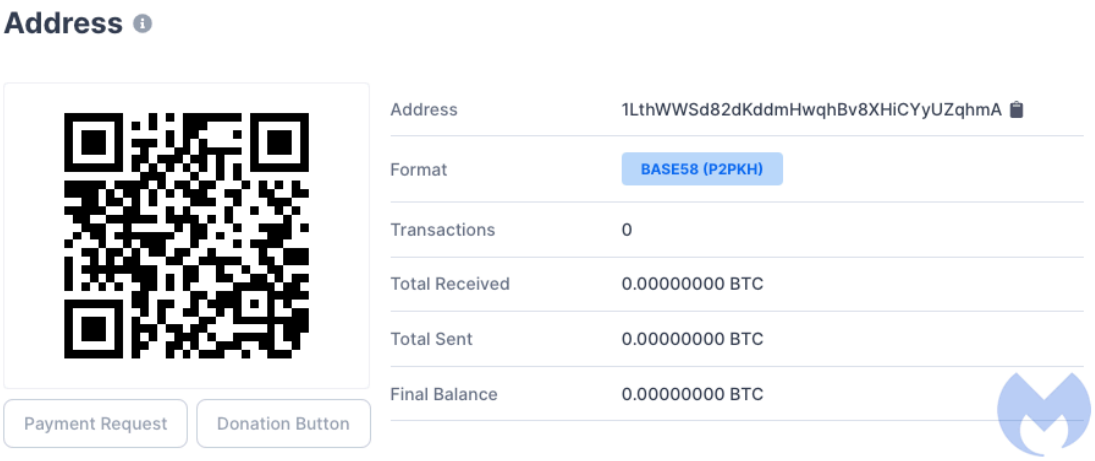
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

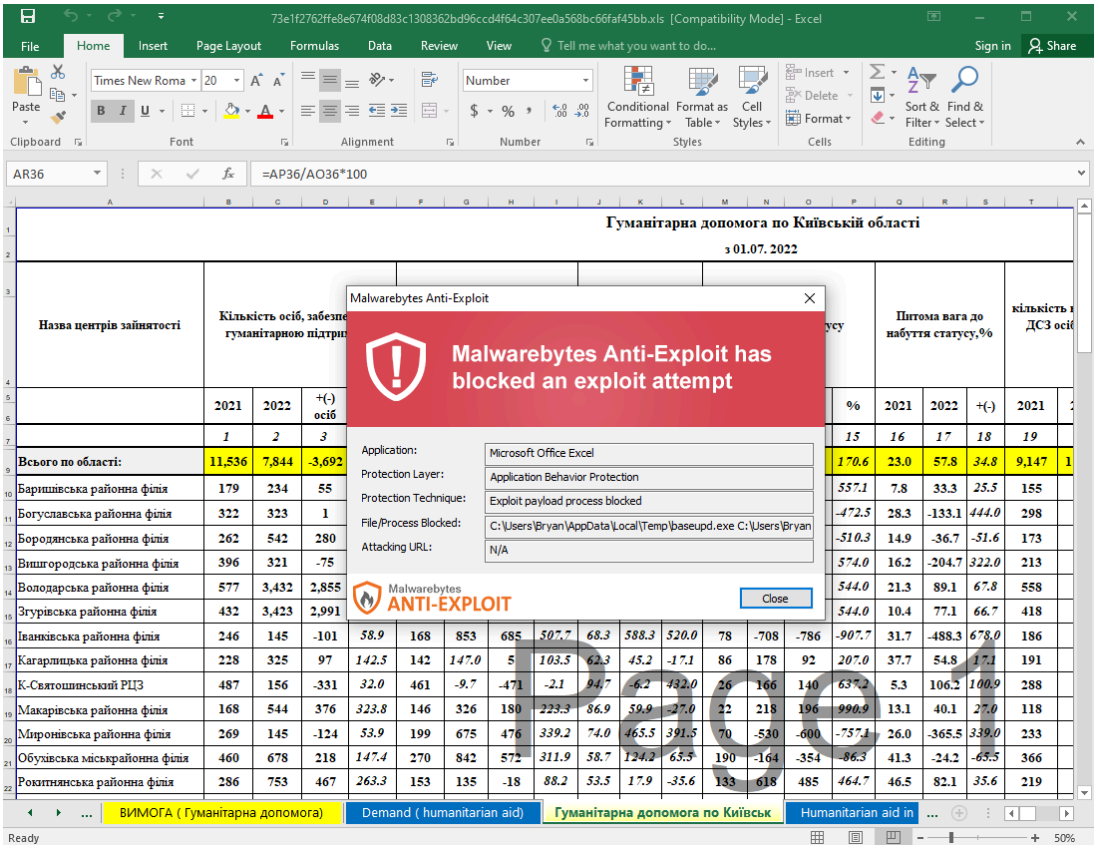
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
46  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
47    runtime_morestack_noctxt();
48  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
49  ((void (*)(void))loc_45CDA0)();
50  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
51  v38 = v4;
52  v30 = v5;
53  v31 = v6;
54  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
55  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
56  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
57  v32 = v10;
58  v11 = *(void (__cdecl **)(int))(v10 + 16);
59  v39 = v12;
60  v11(v12);
61  v29 = v13;
62  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
63  v40 = v15;
64  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
65  v17 = v27;
66  if ( v27 )
67  {
68    if ( v27 )
69      v17 = *(_DWORD *)(v27 + 4);
70    v43 = v17;
71    v44 = v28;
72    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
73  }
```

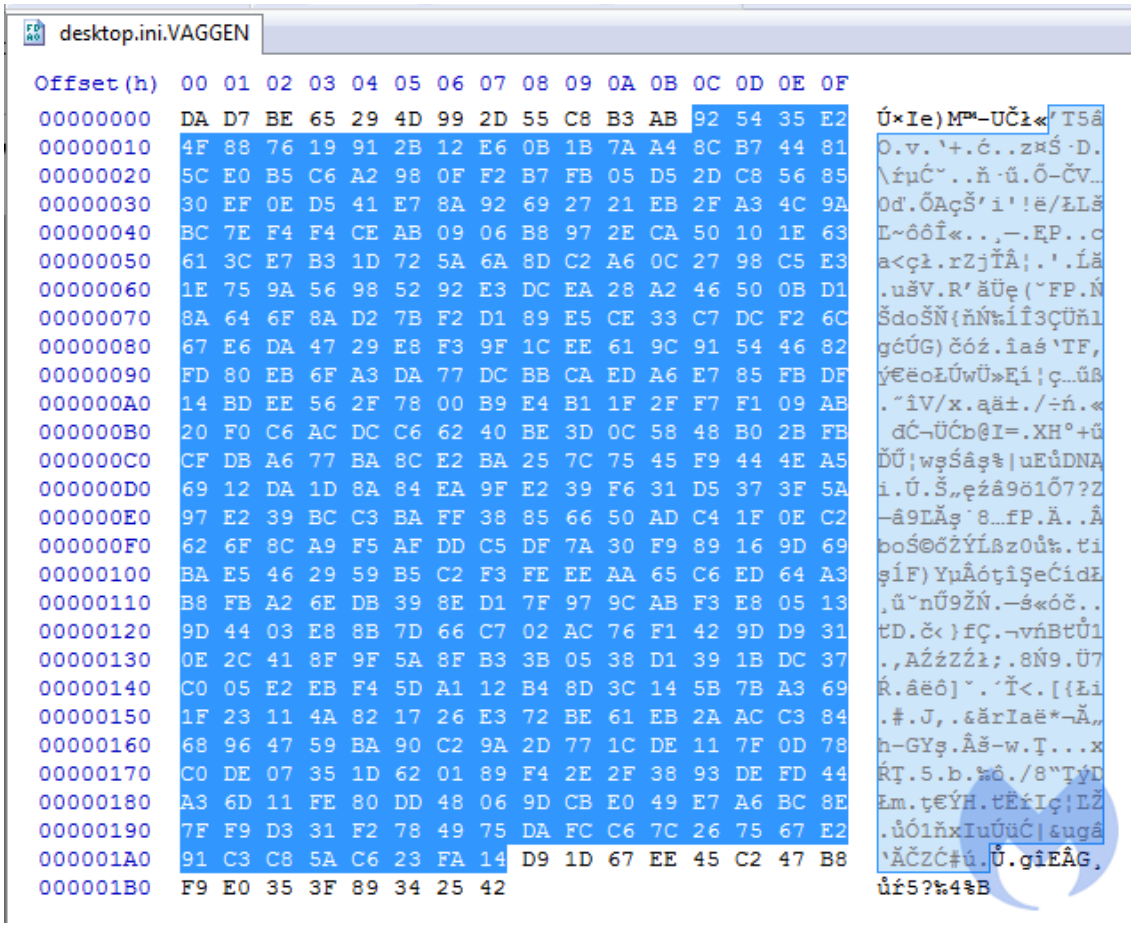


The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

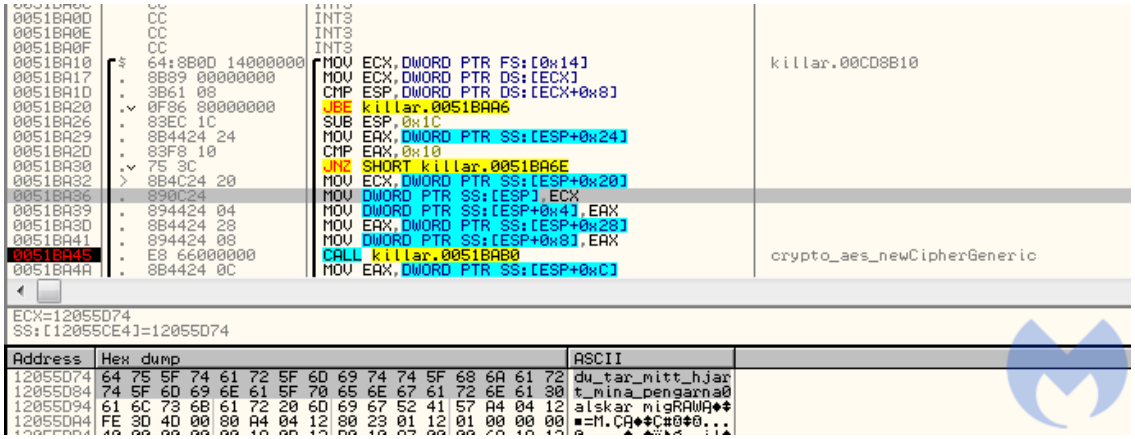


The content of the output file (with .VAGGEN extension) contains:

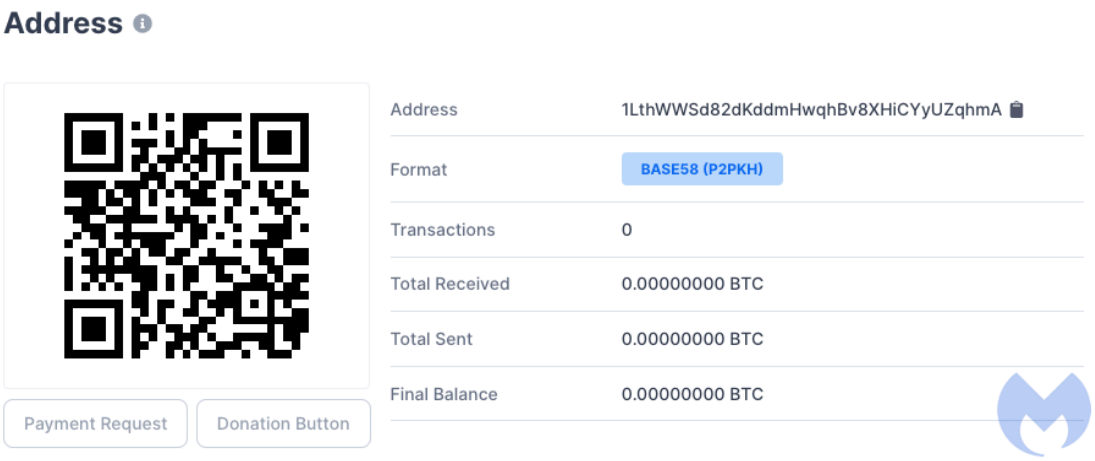
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

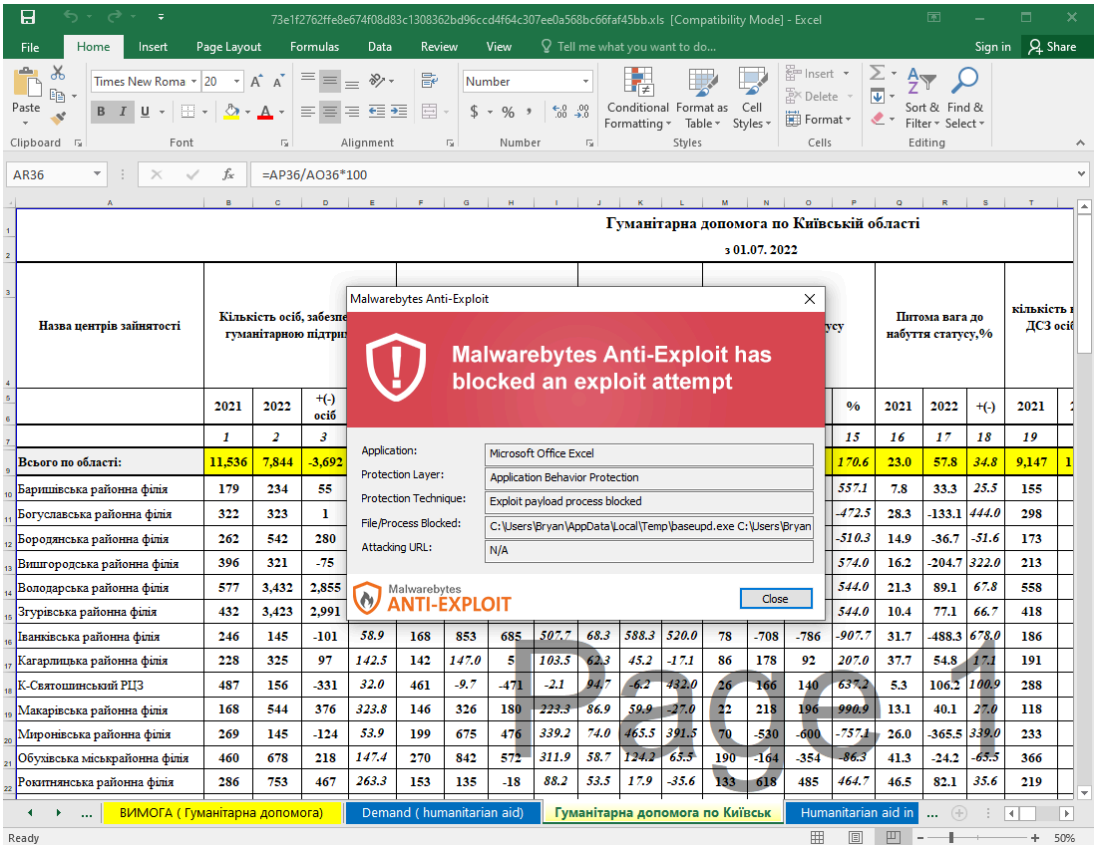
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

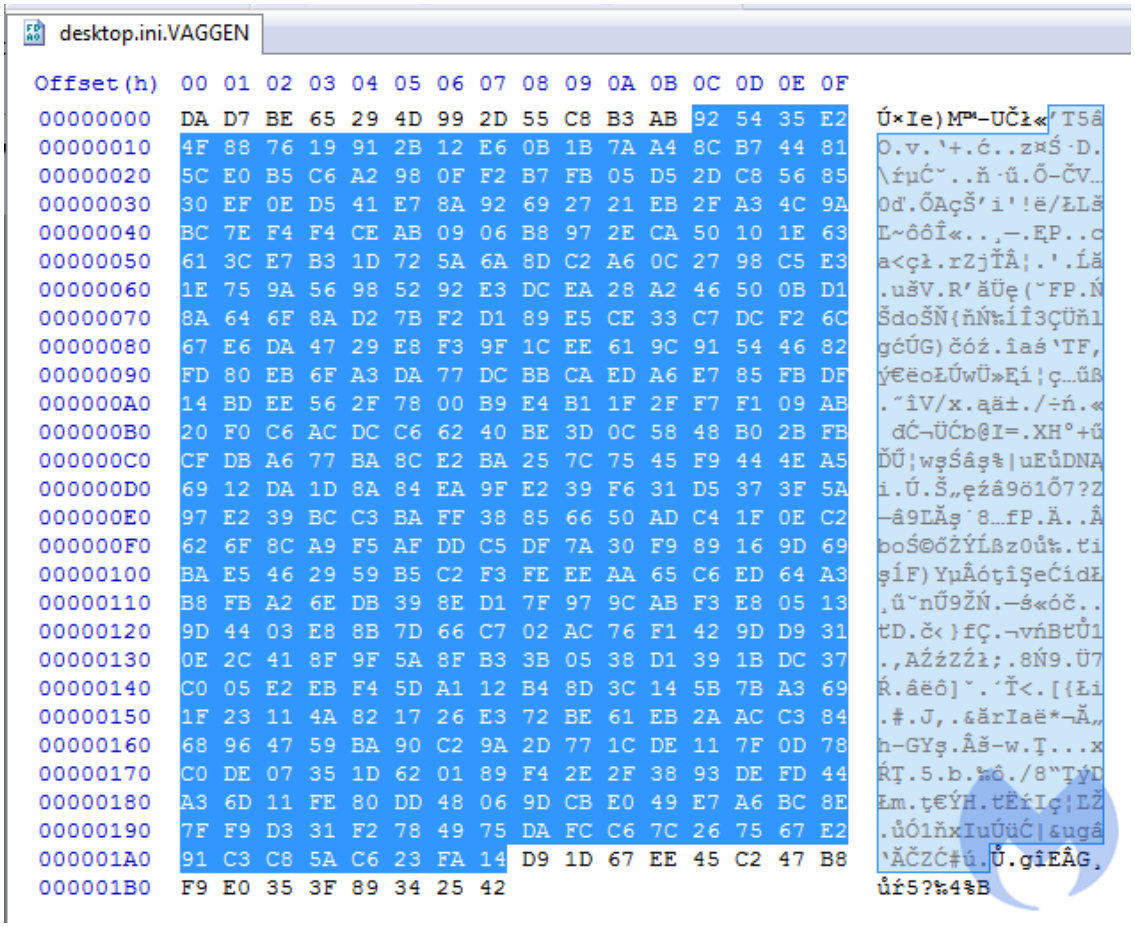
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

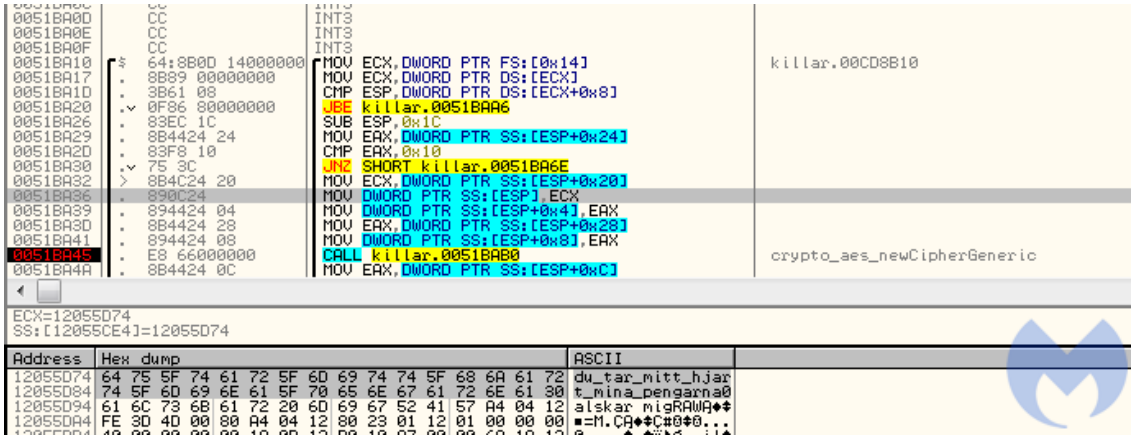


The content of the output file (with .VAGGEN extension) contains:

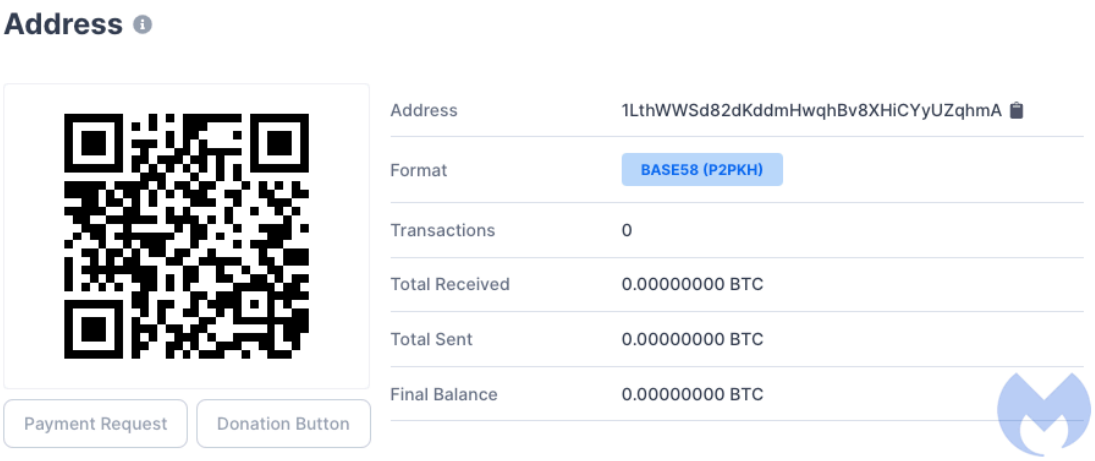
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

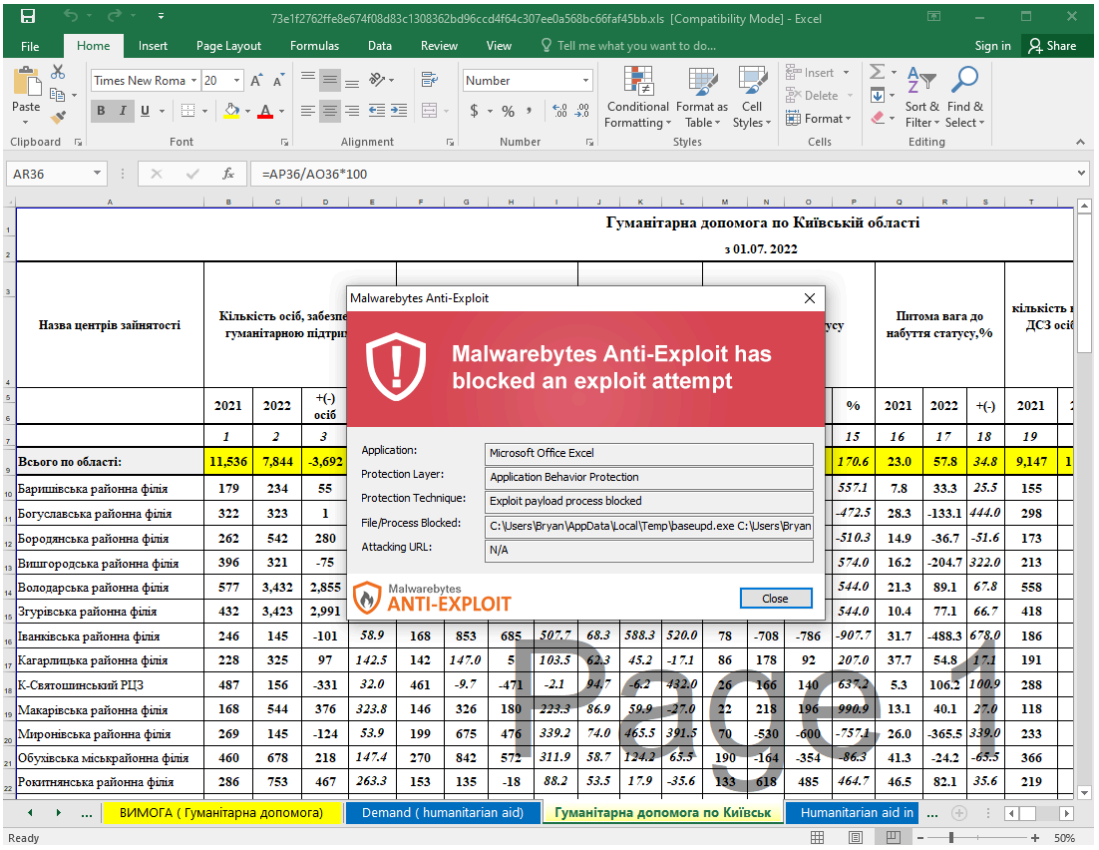
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

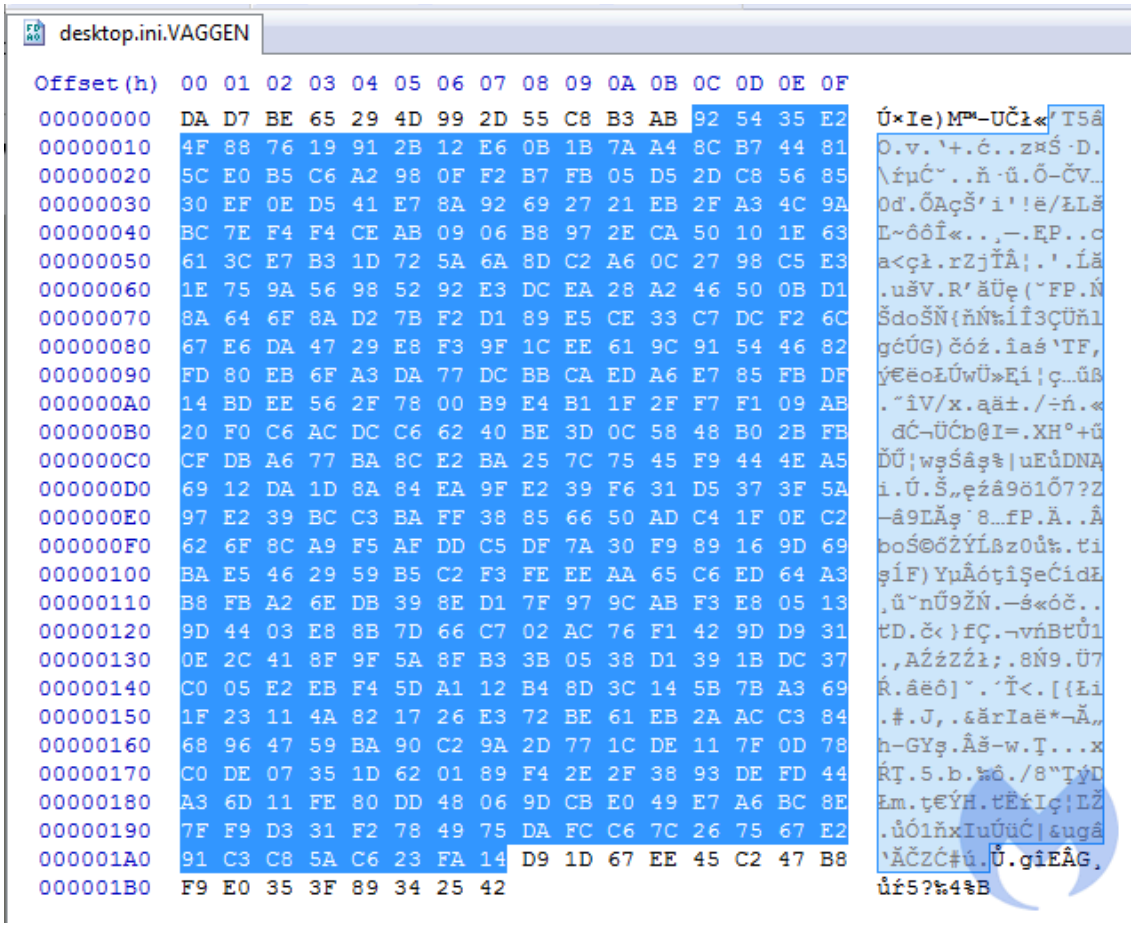
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



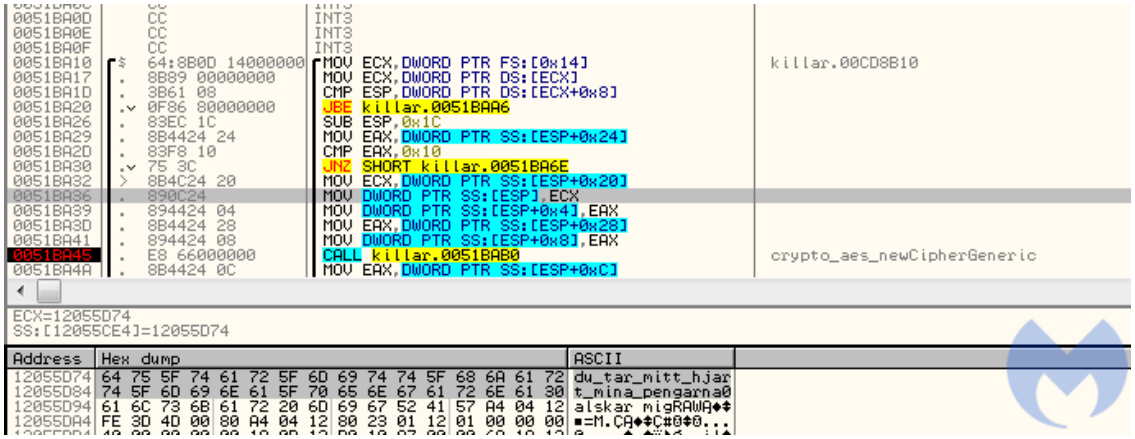
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

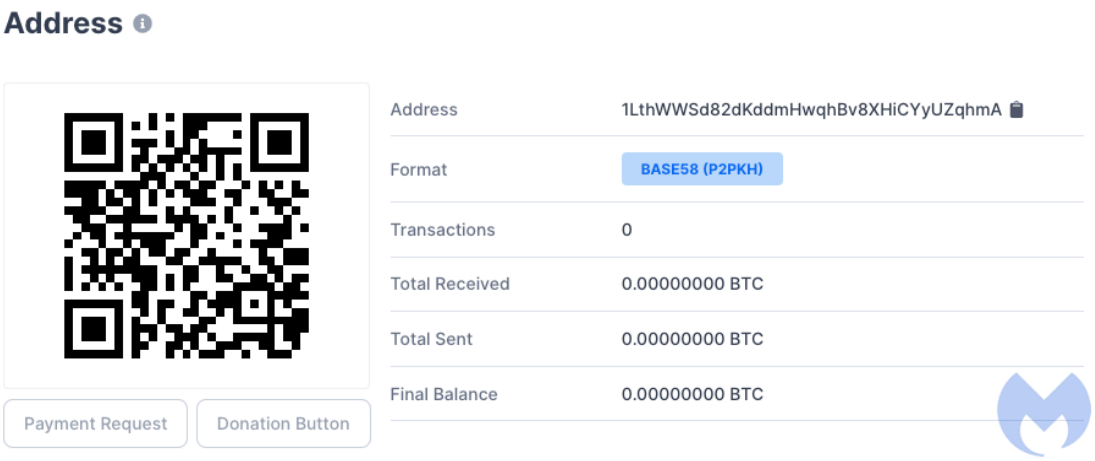
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

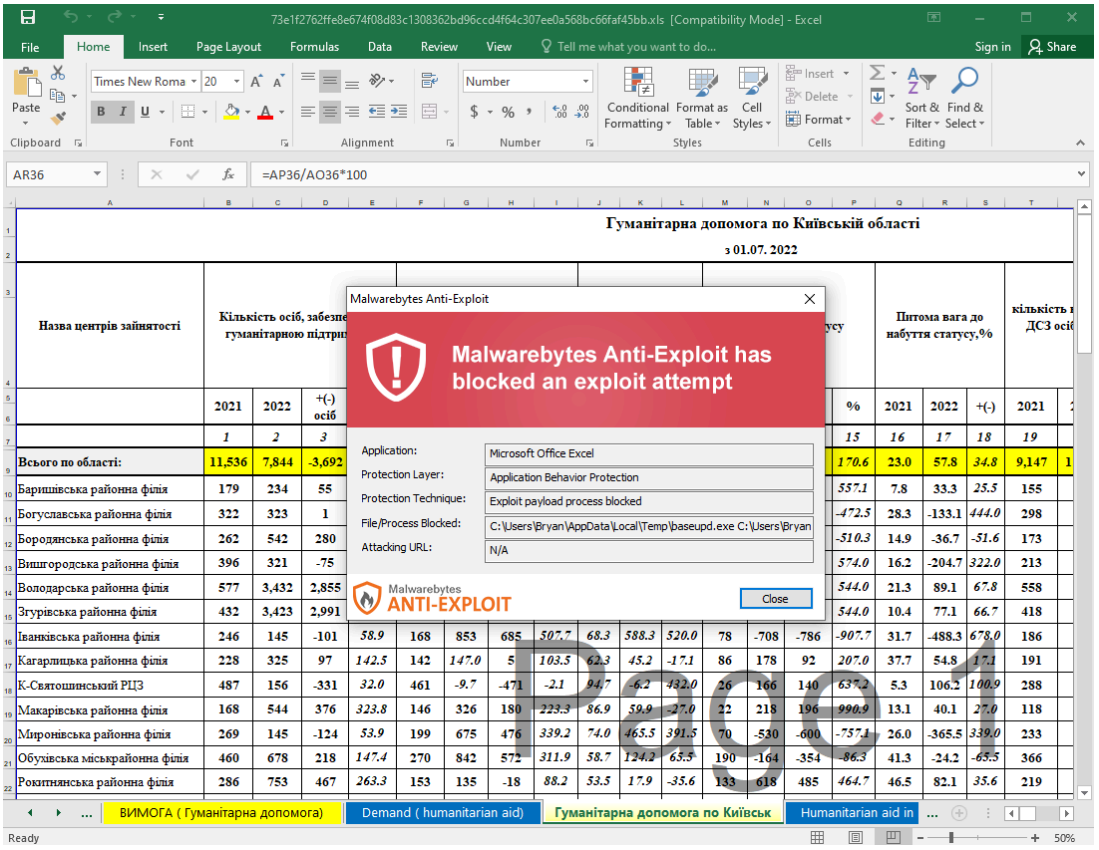
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

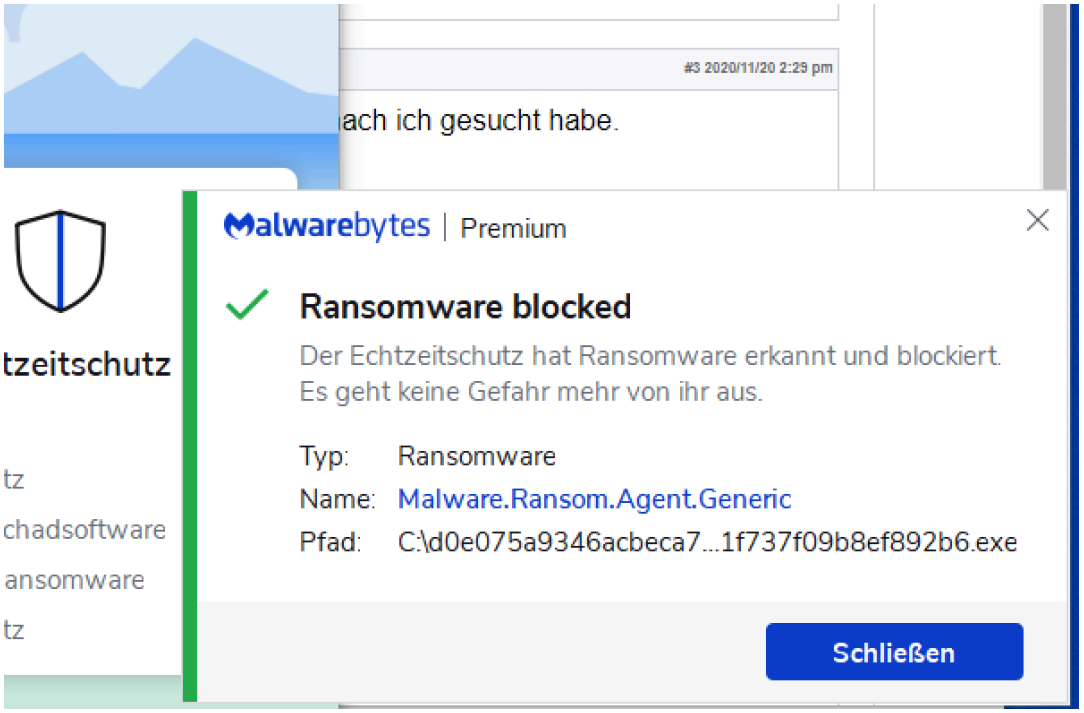
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

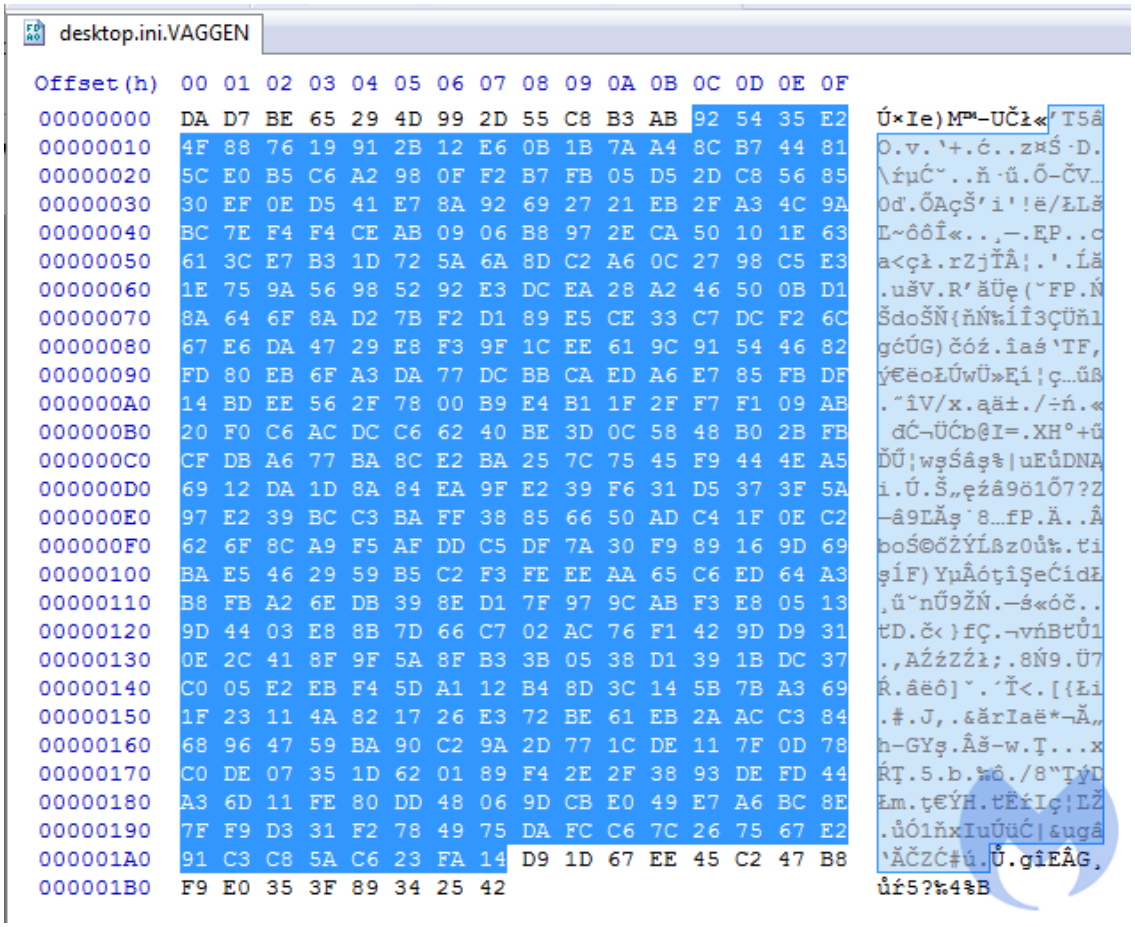
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



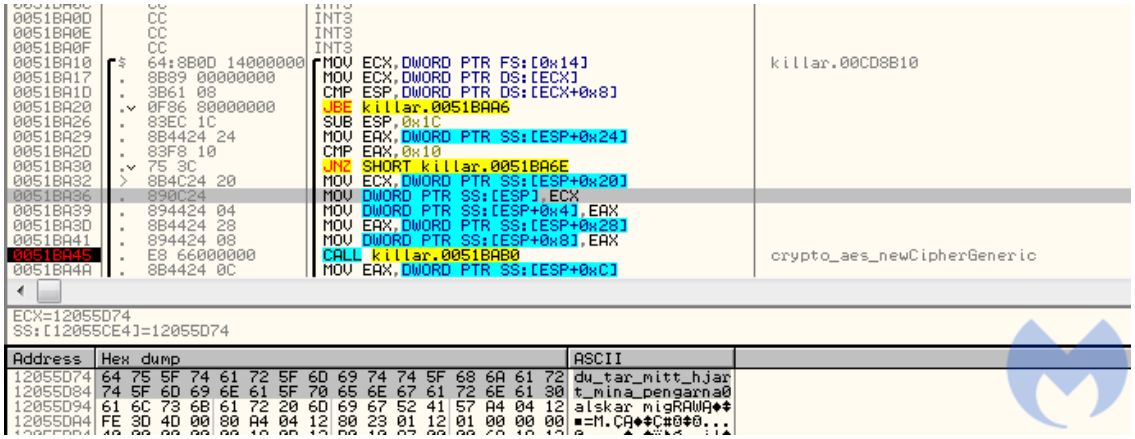
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

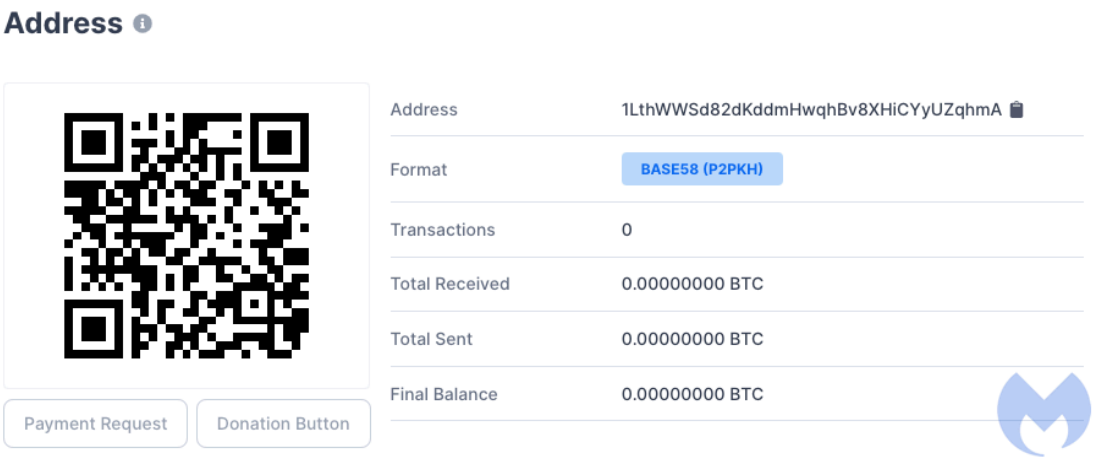
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

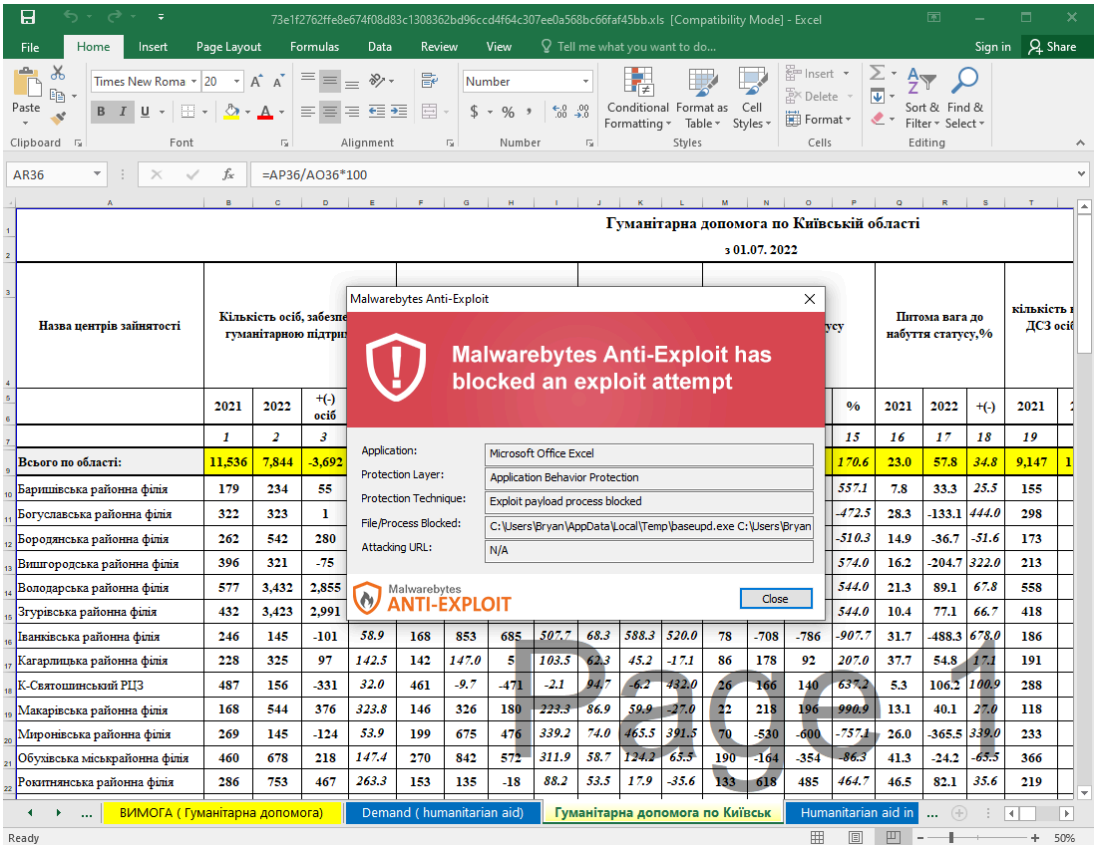
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

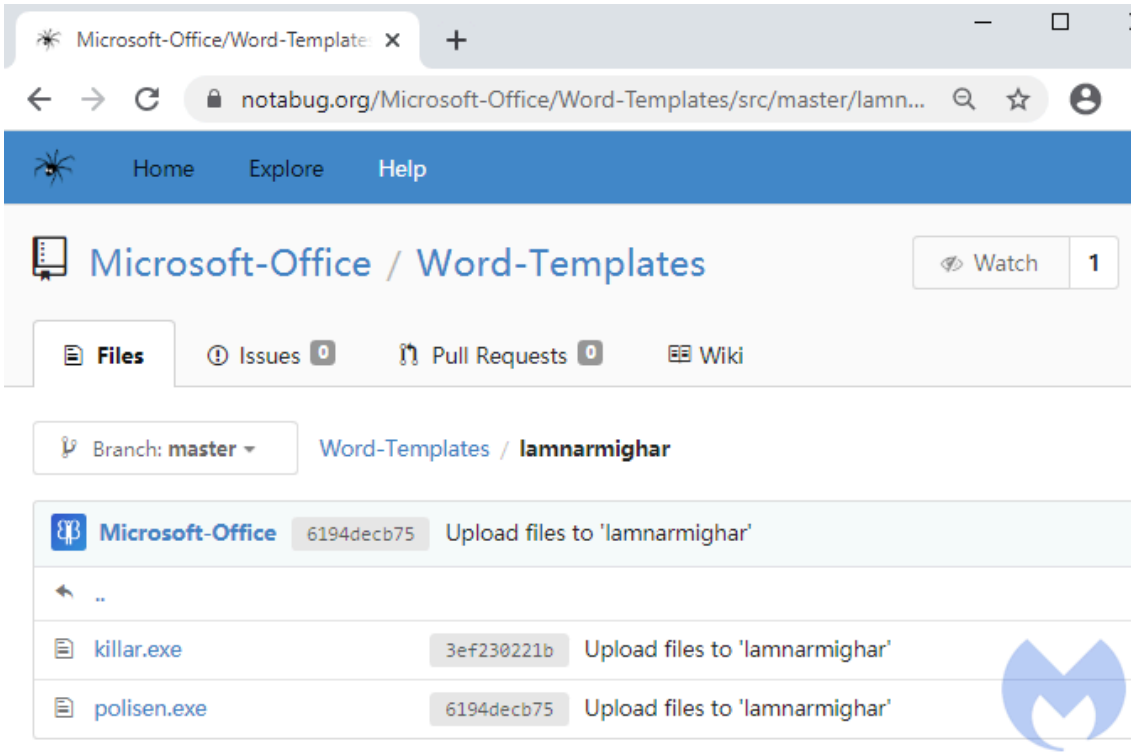
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



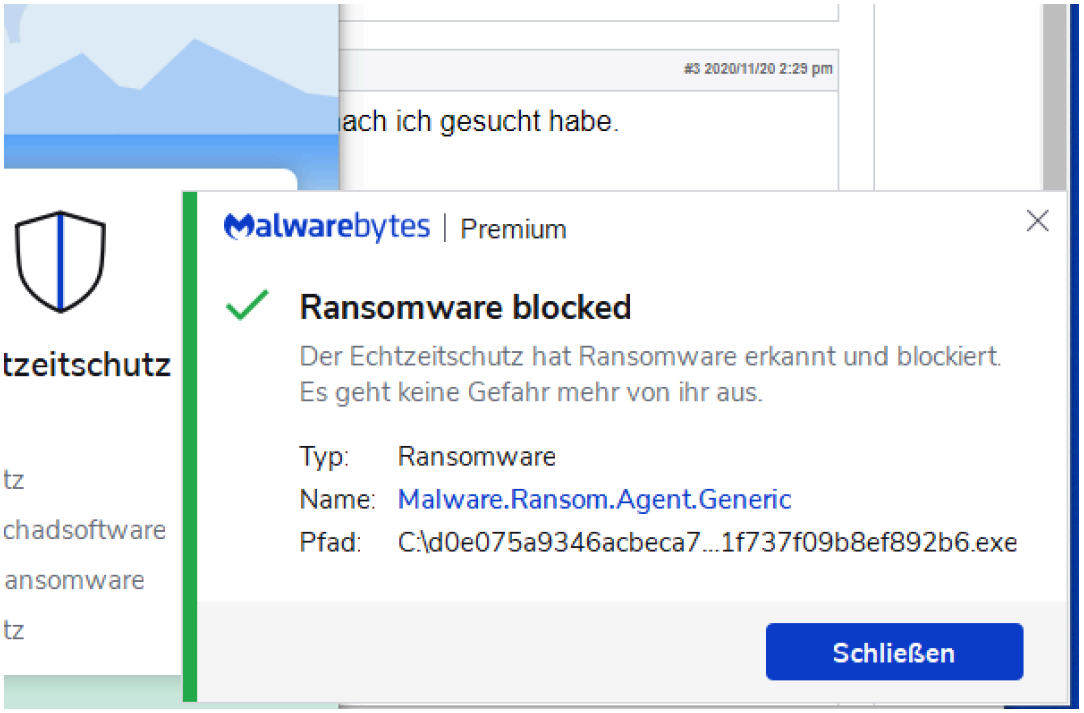
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

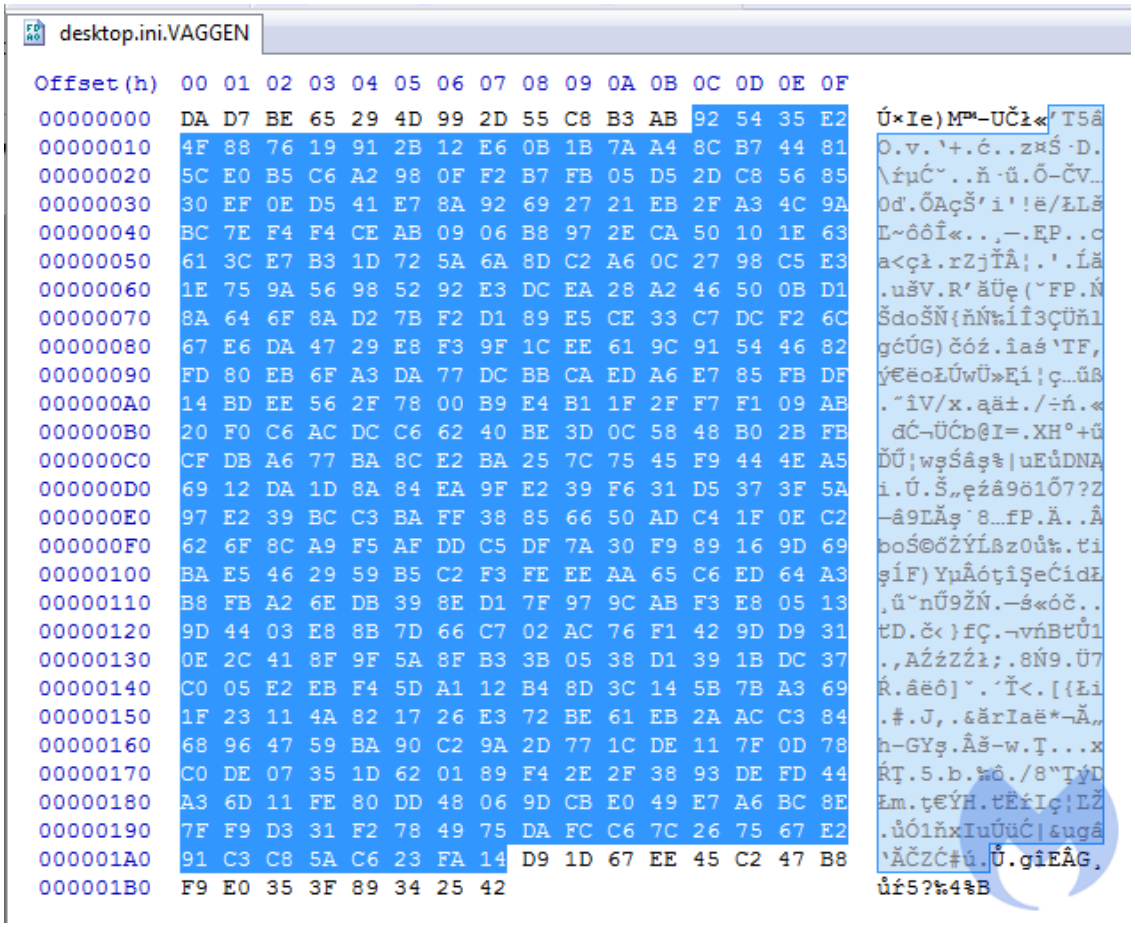
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

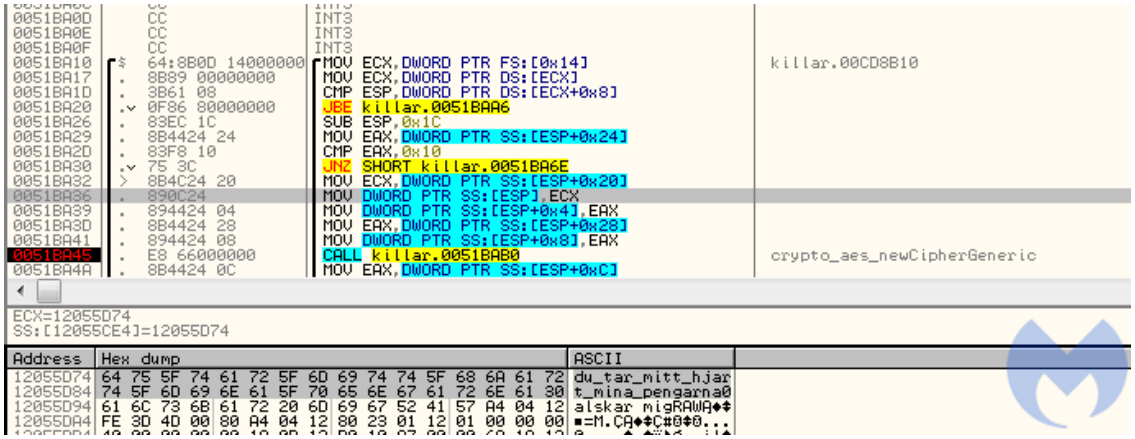


The content of the output file (with .VAGGEN extension) contains:

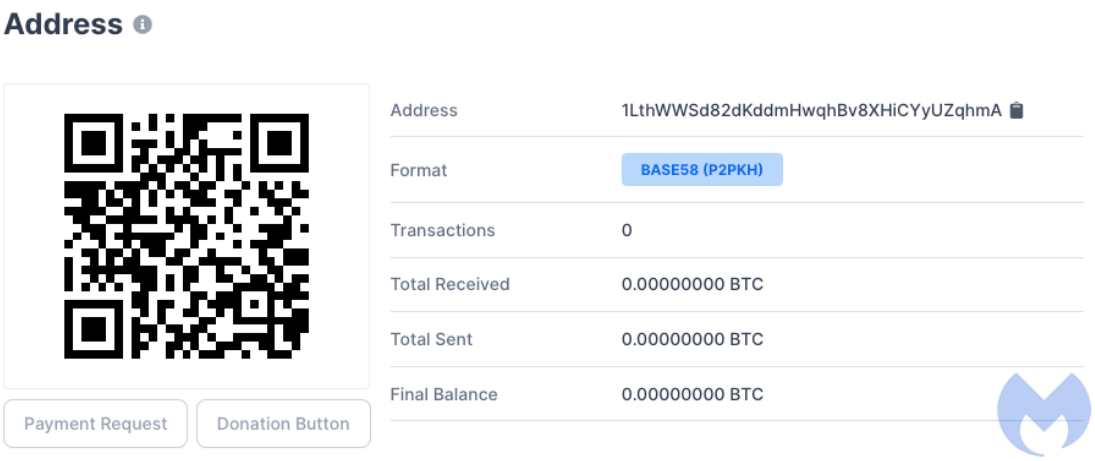
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

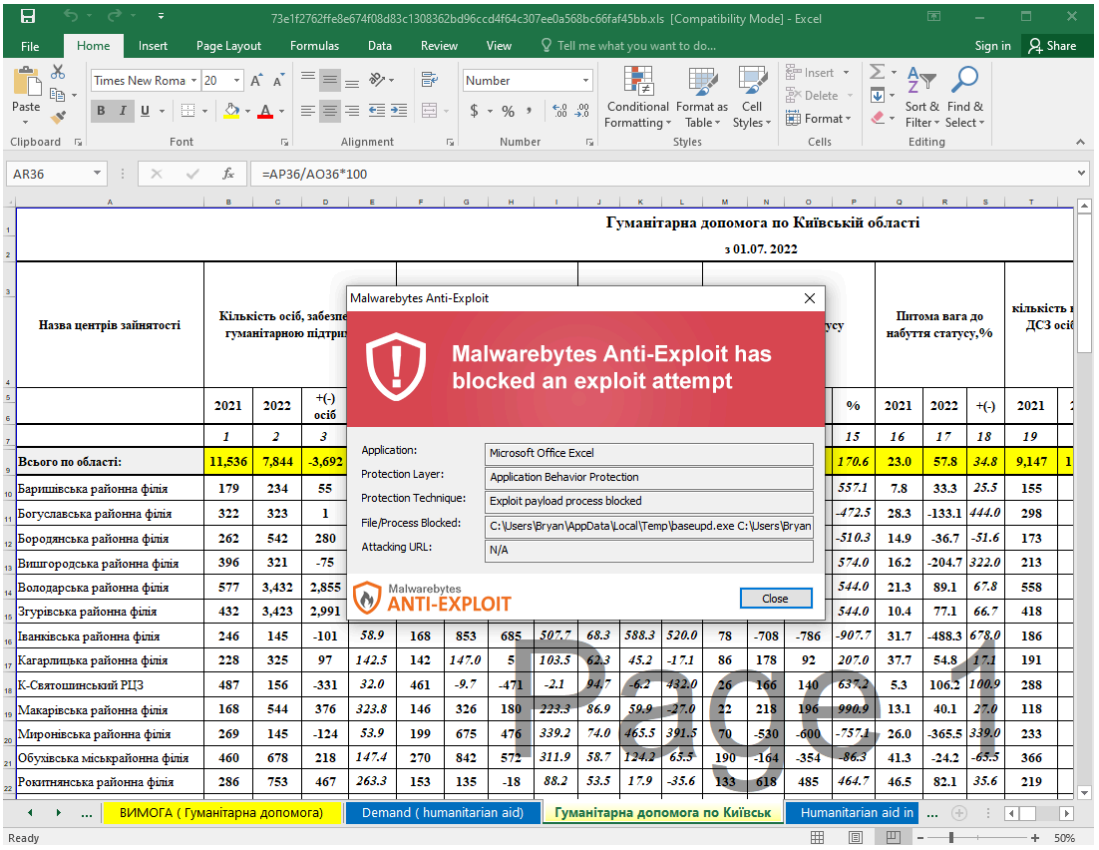
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

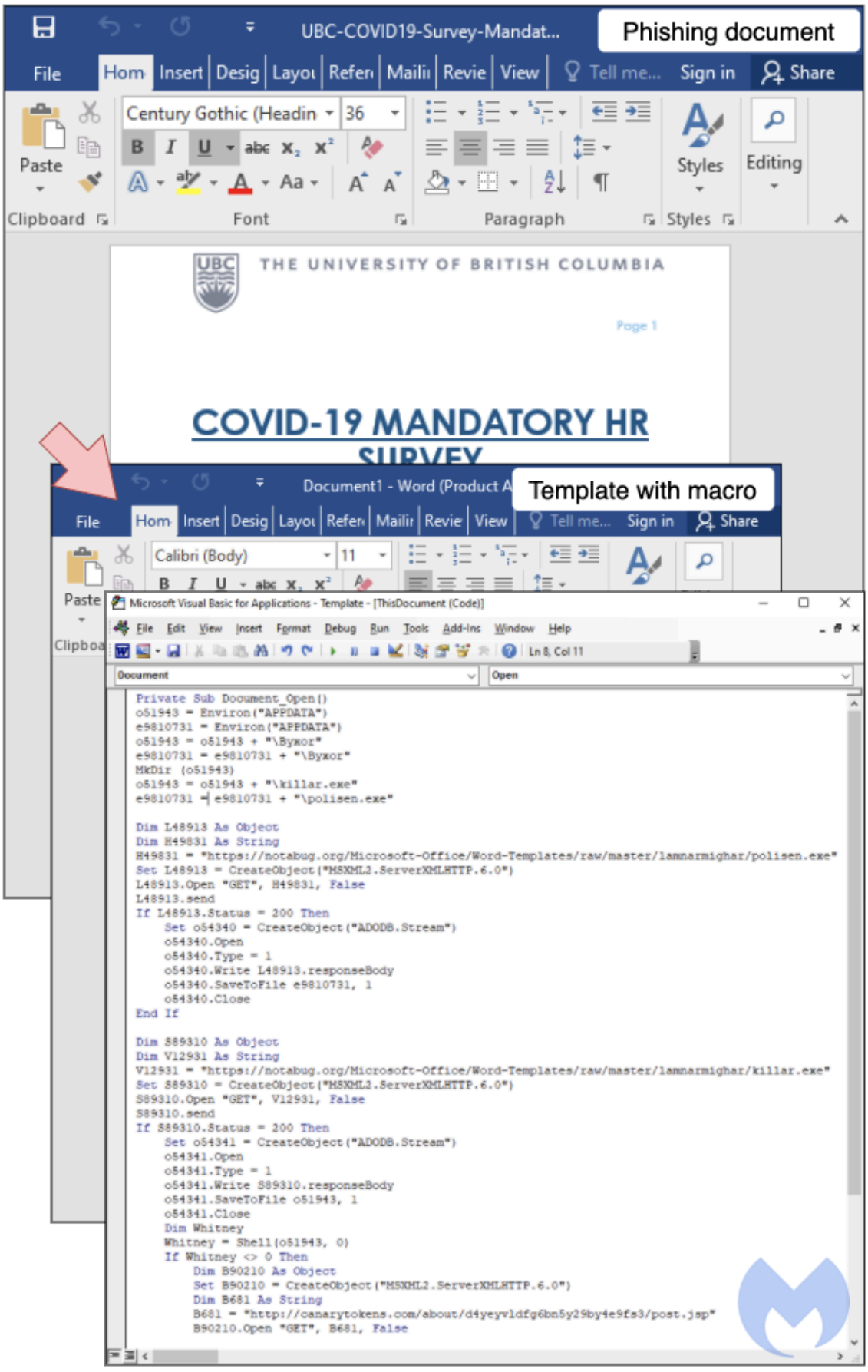
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

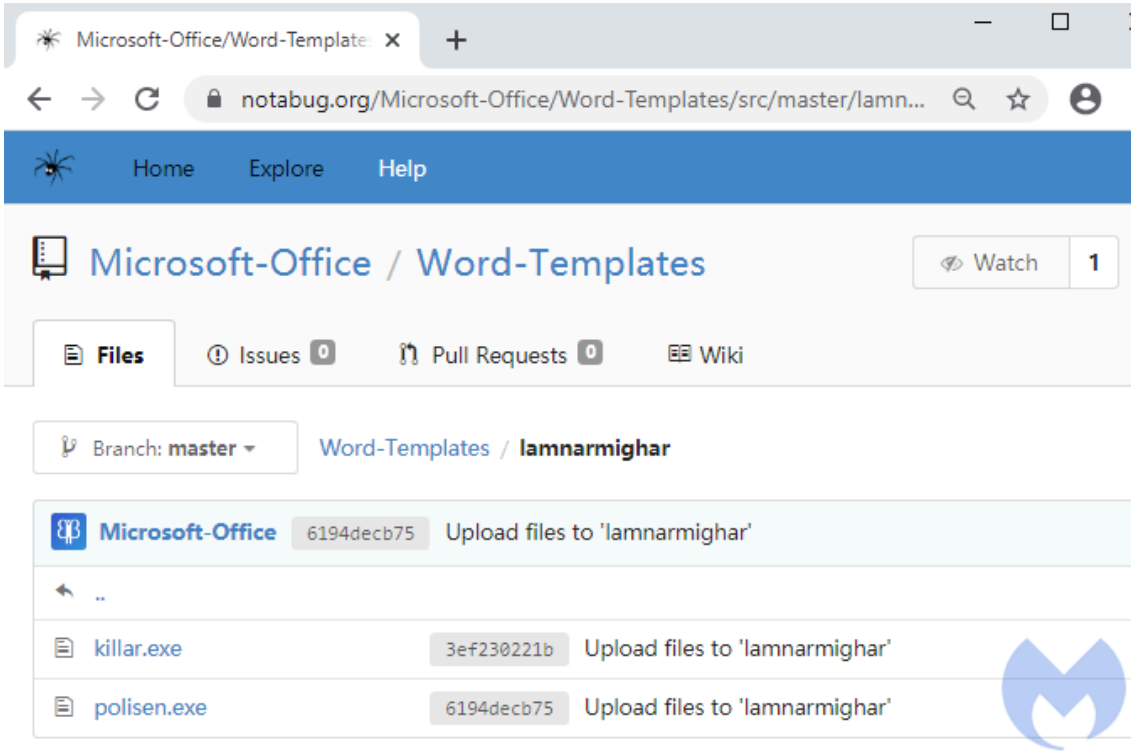
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



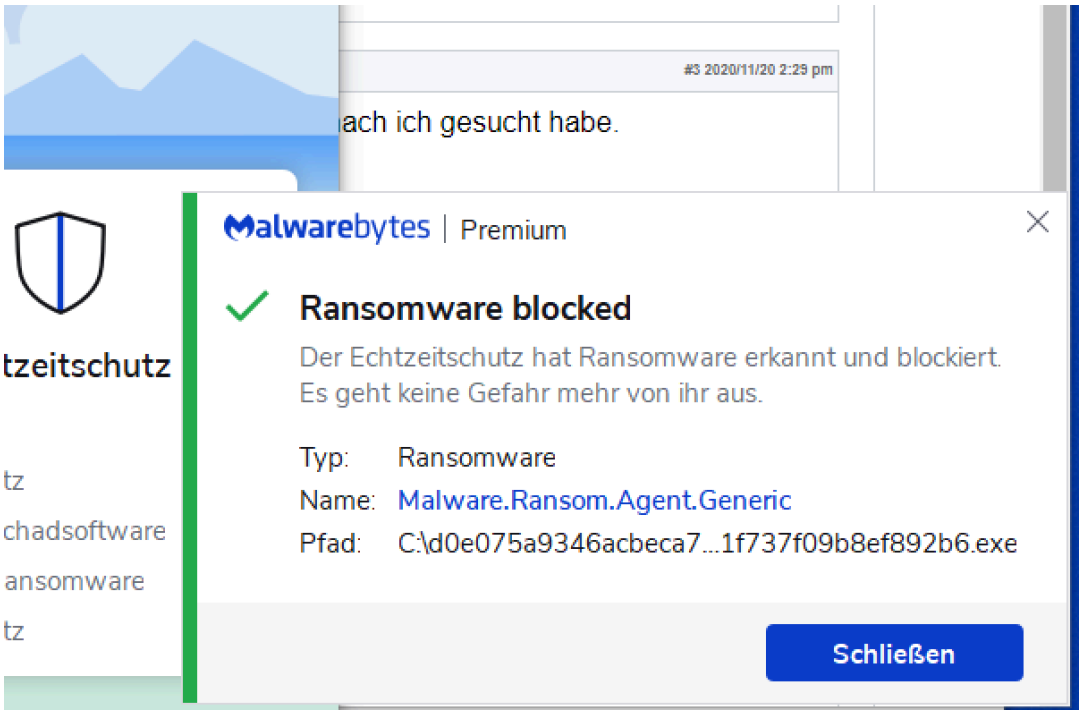
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

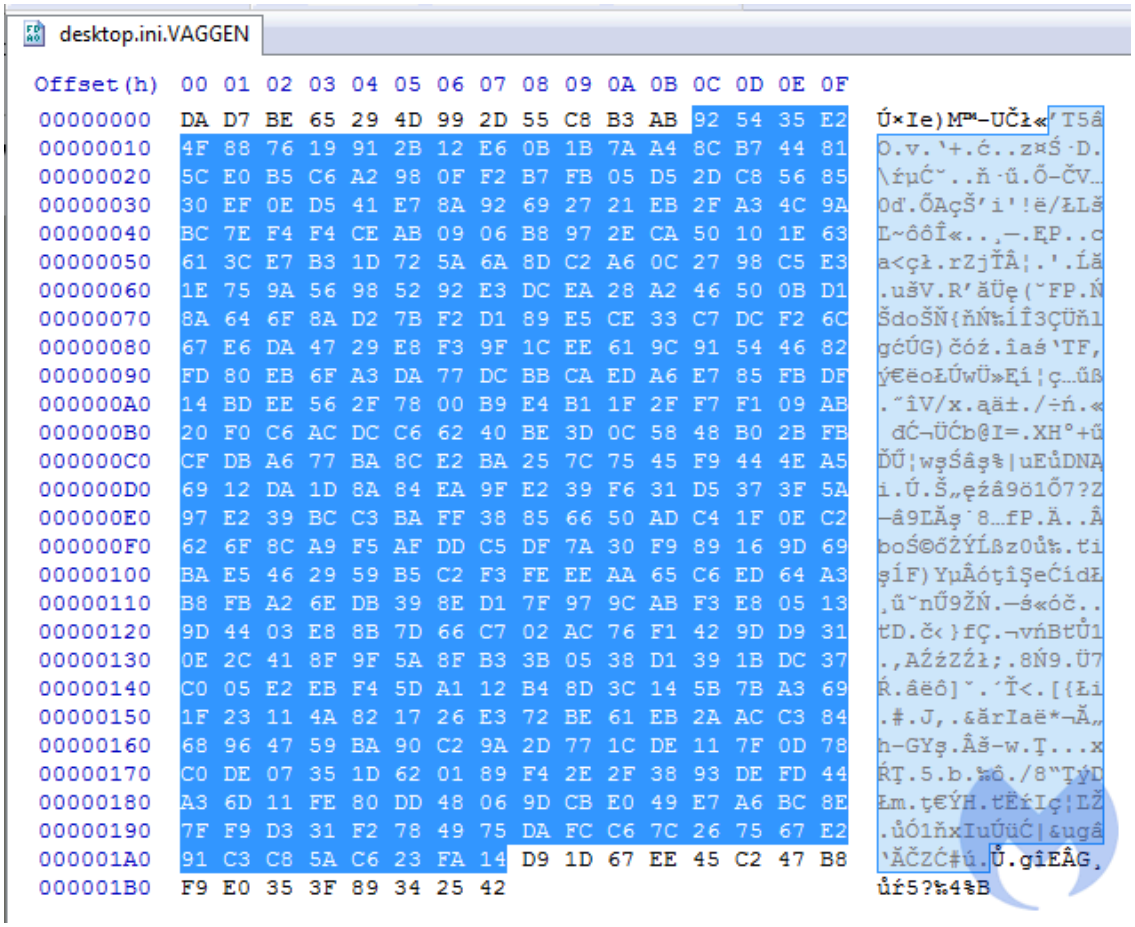
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

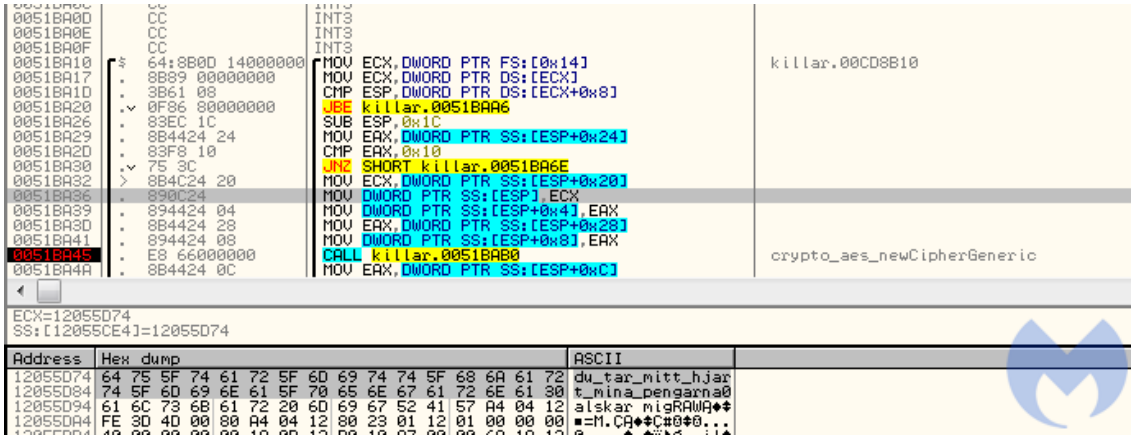


The content of the output file (with .VAGGEN extension) contains:

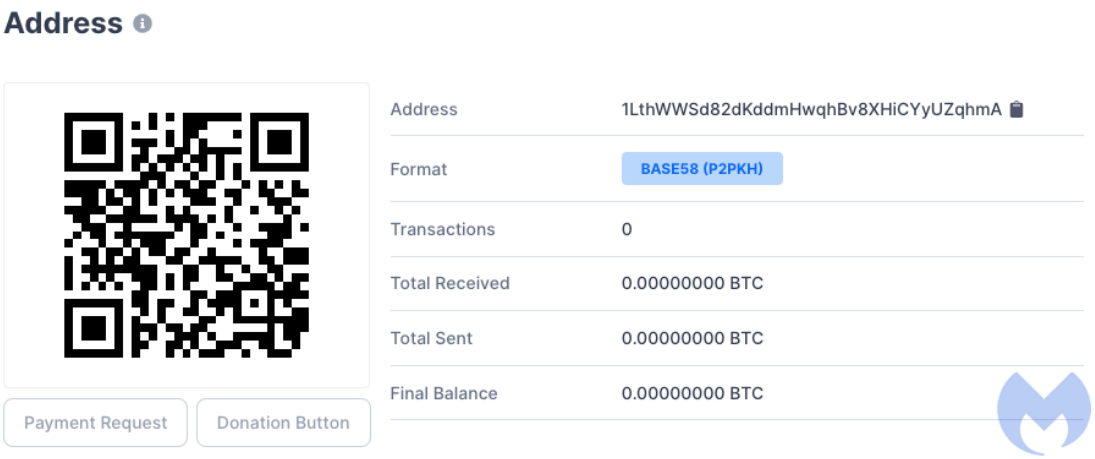
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

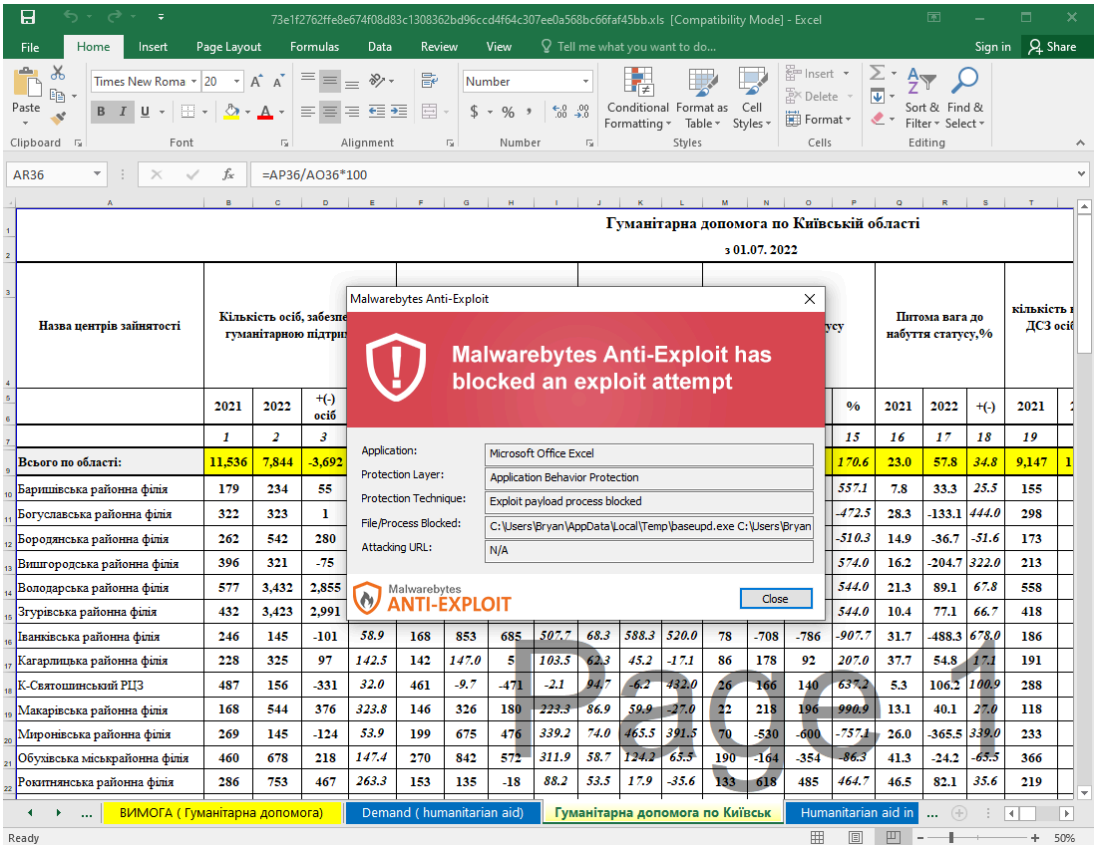
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

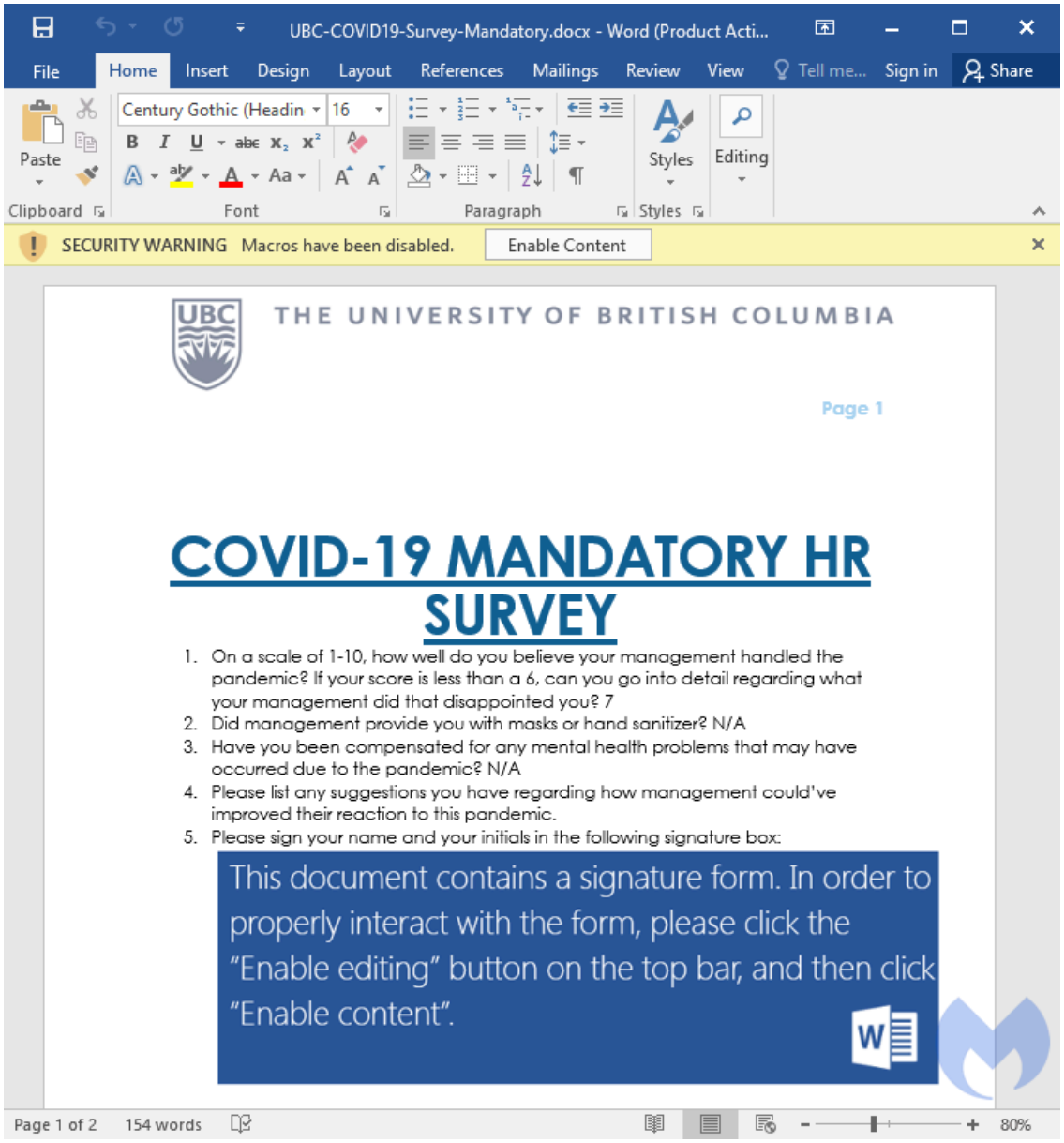
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

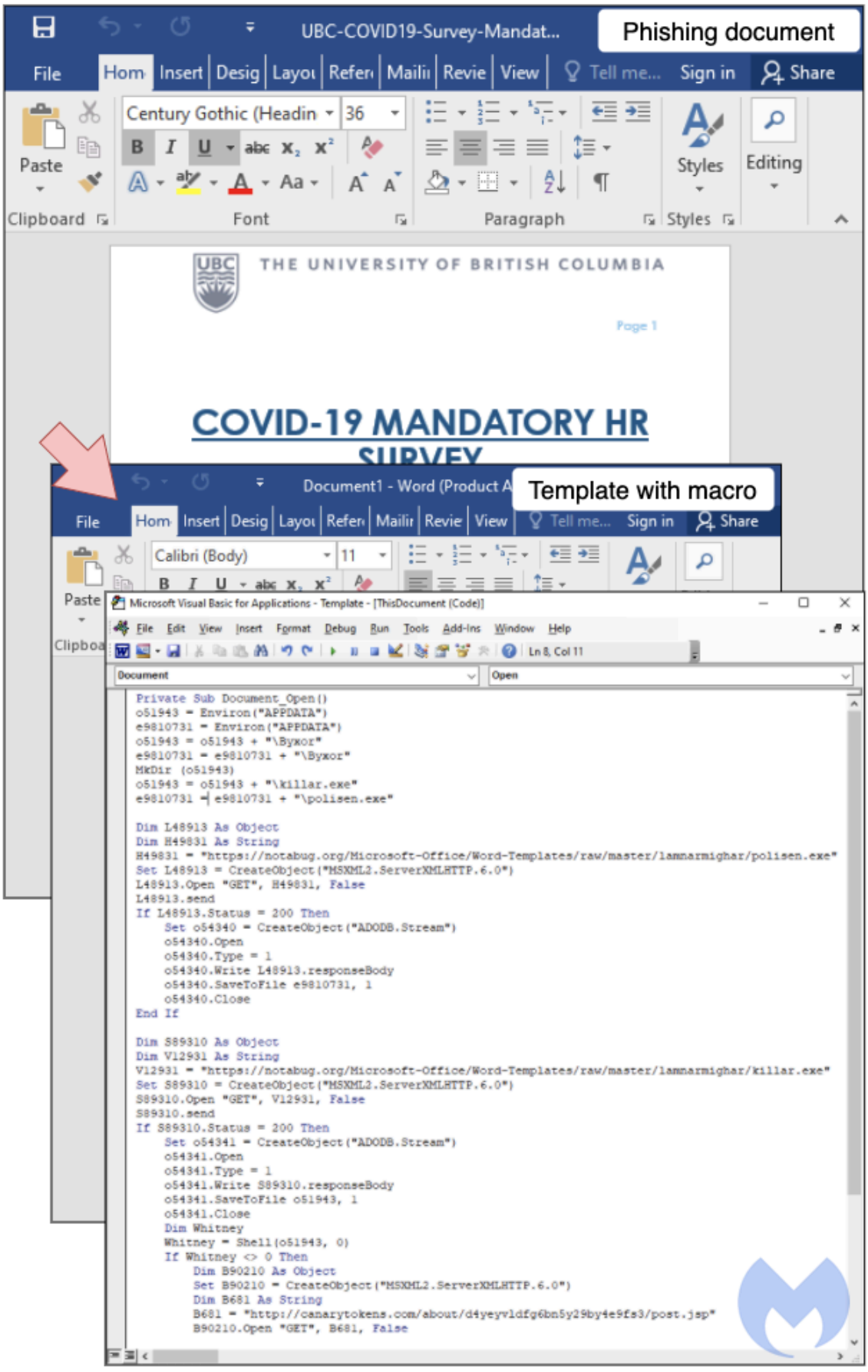
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

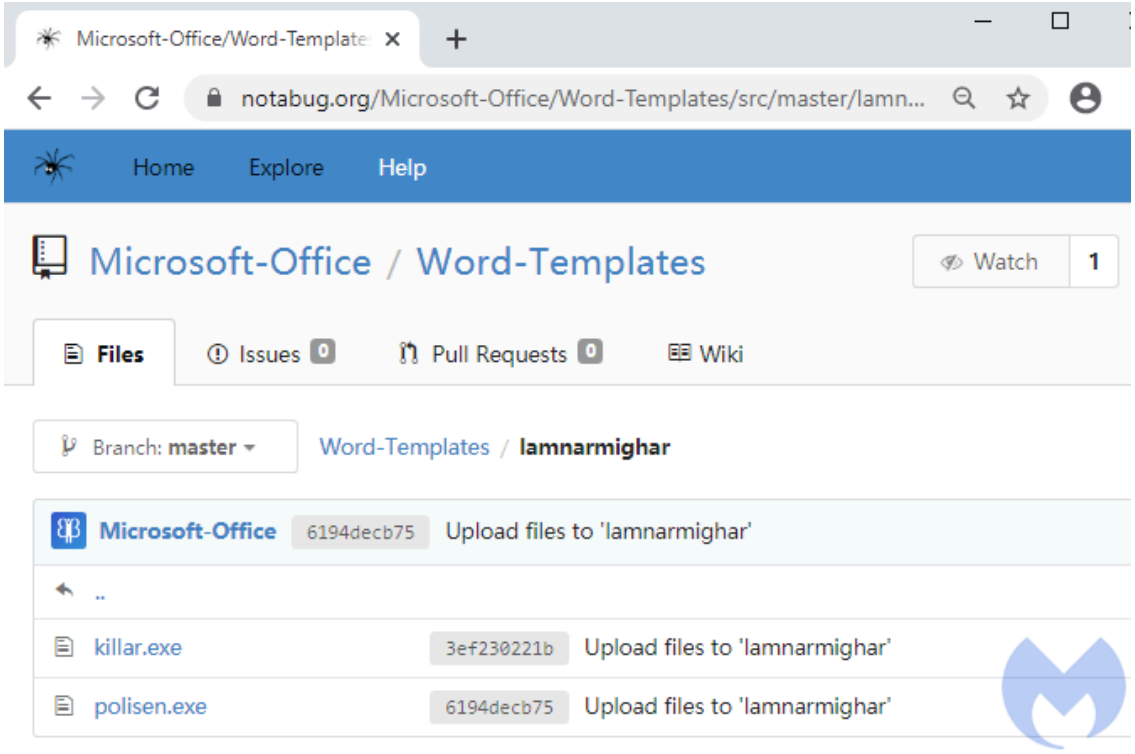
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



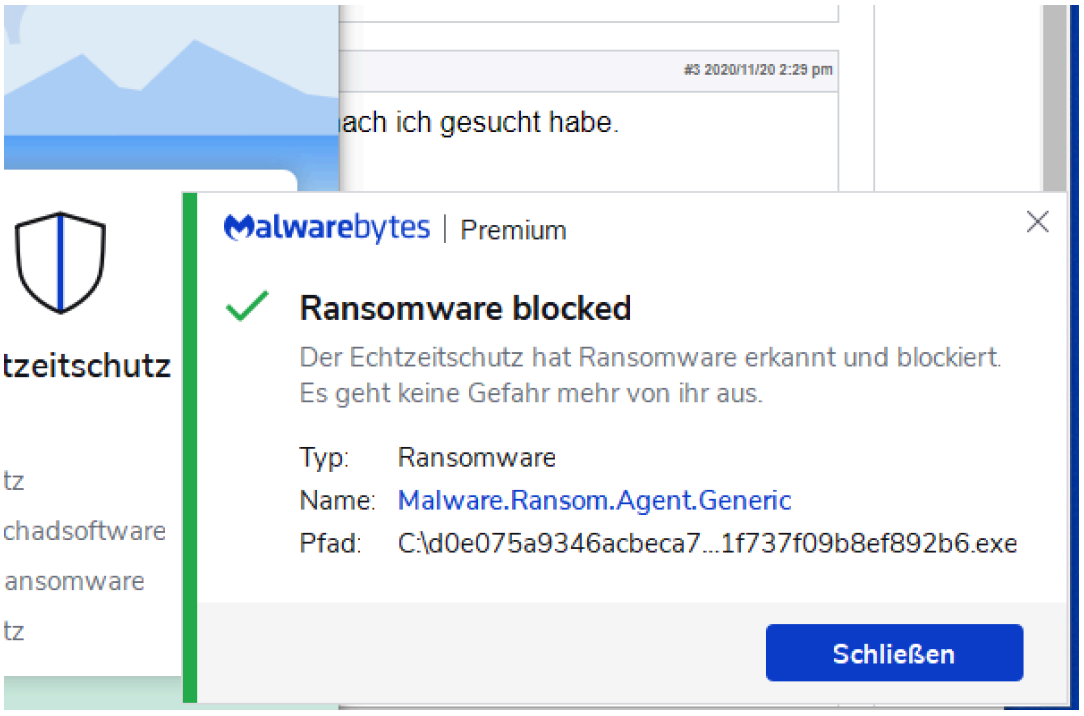
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

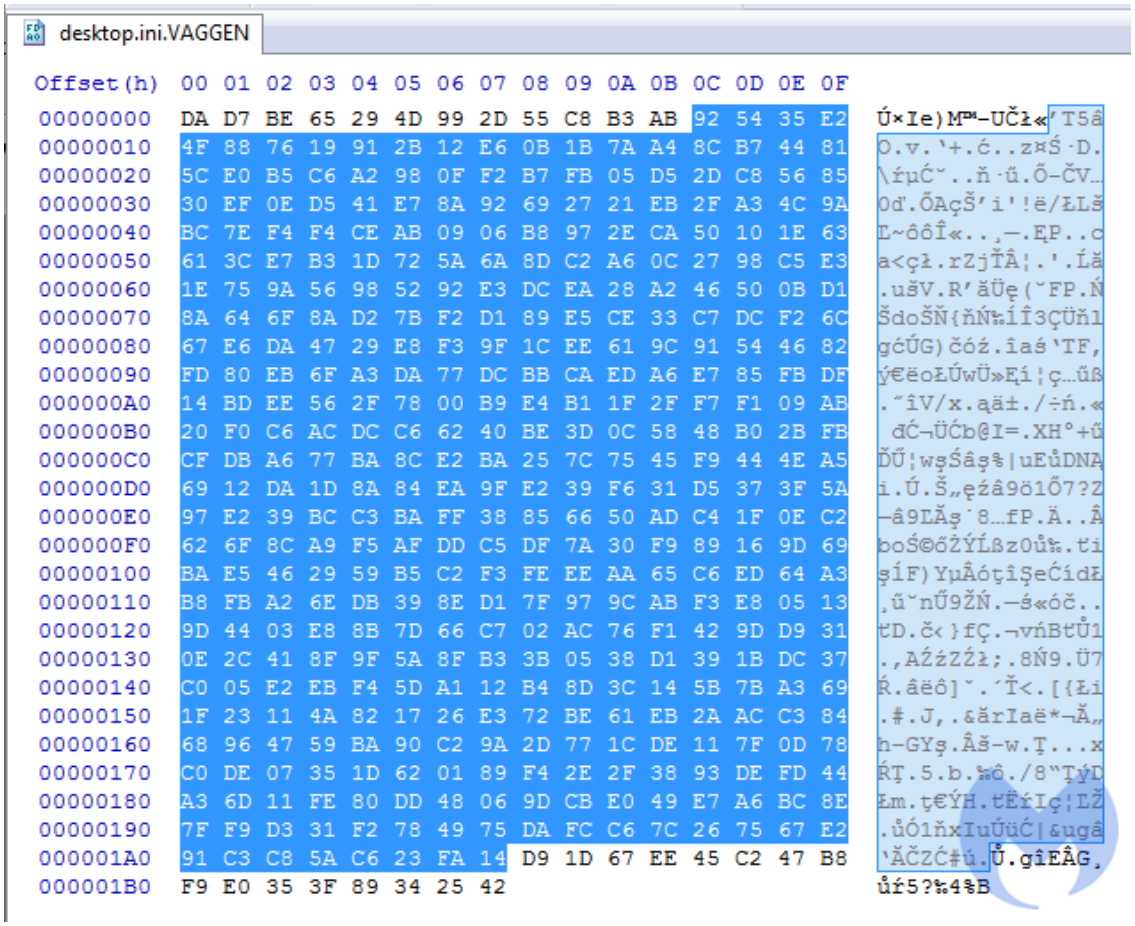
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

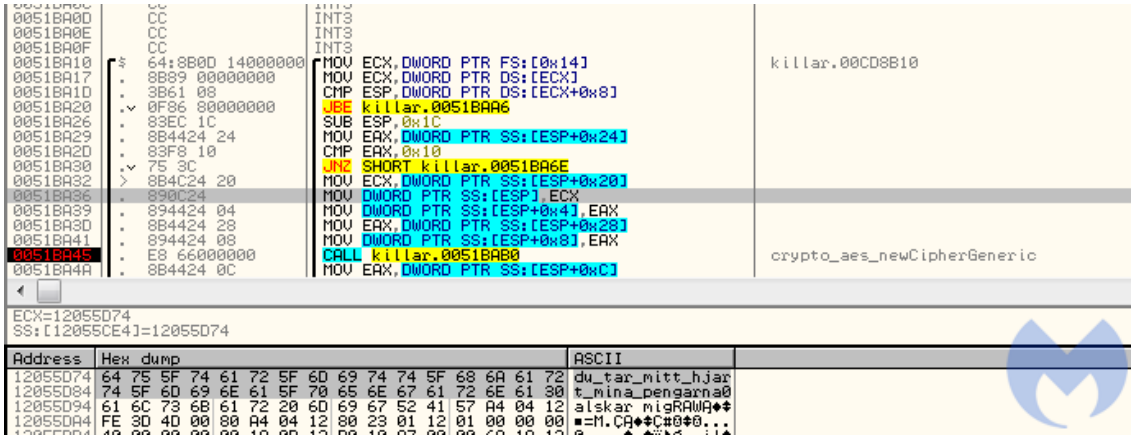


The content of the output file (with .VAGGEN extension) contains:

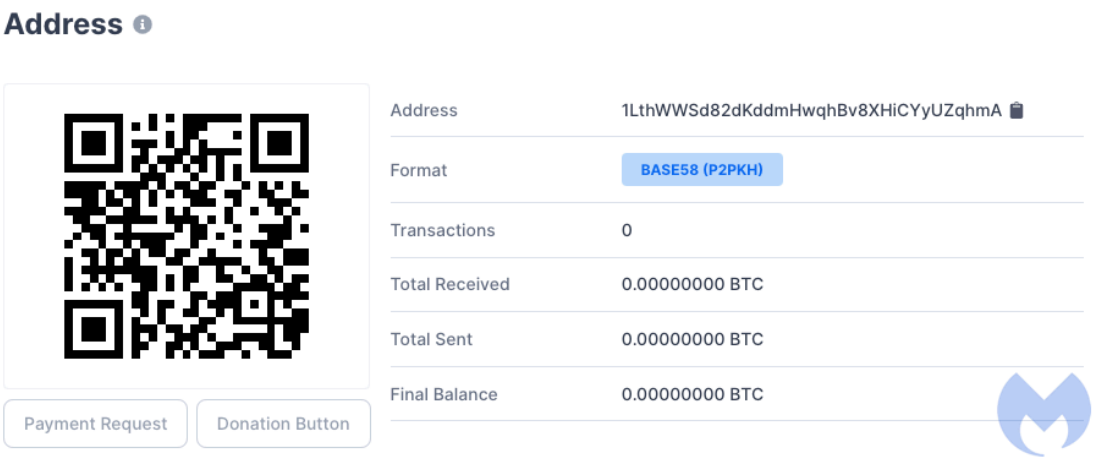
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

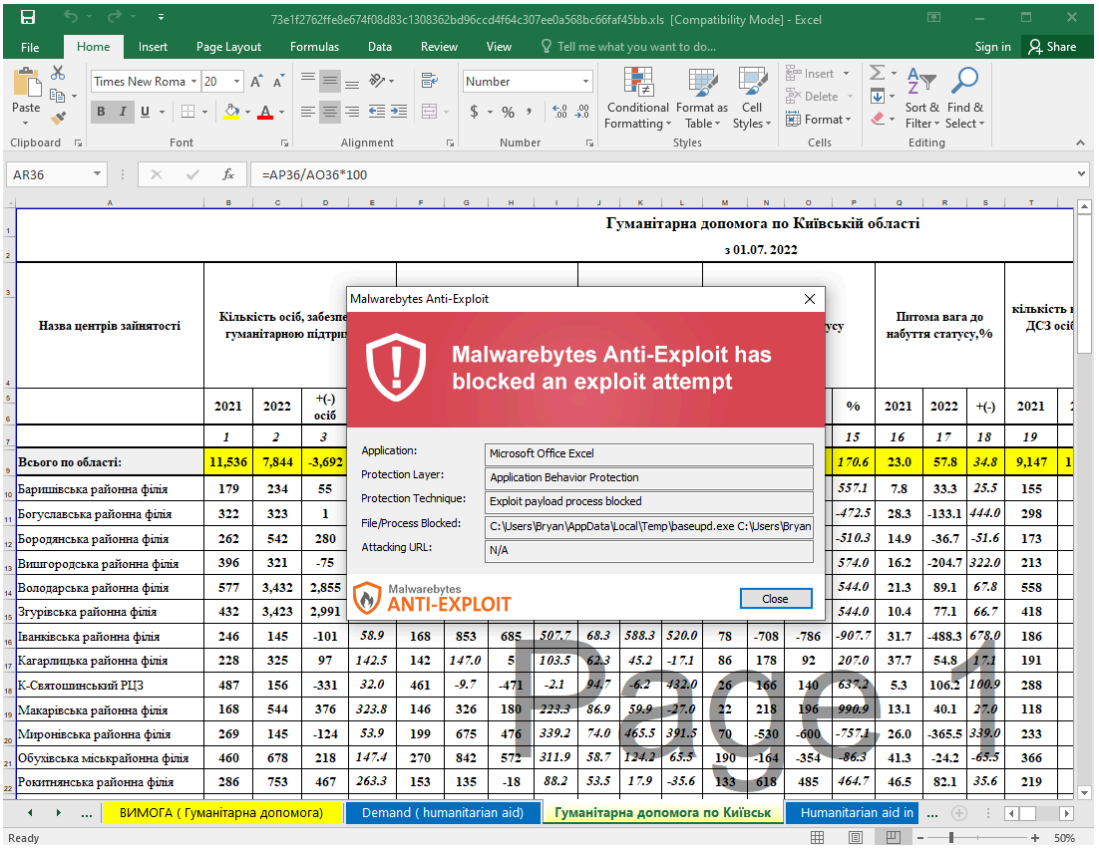
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

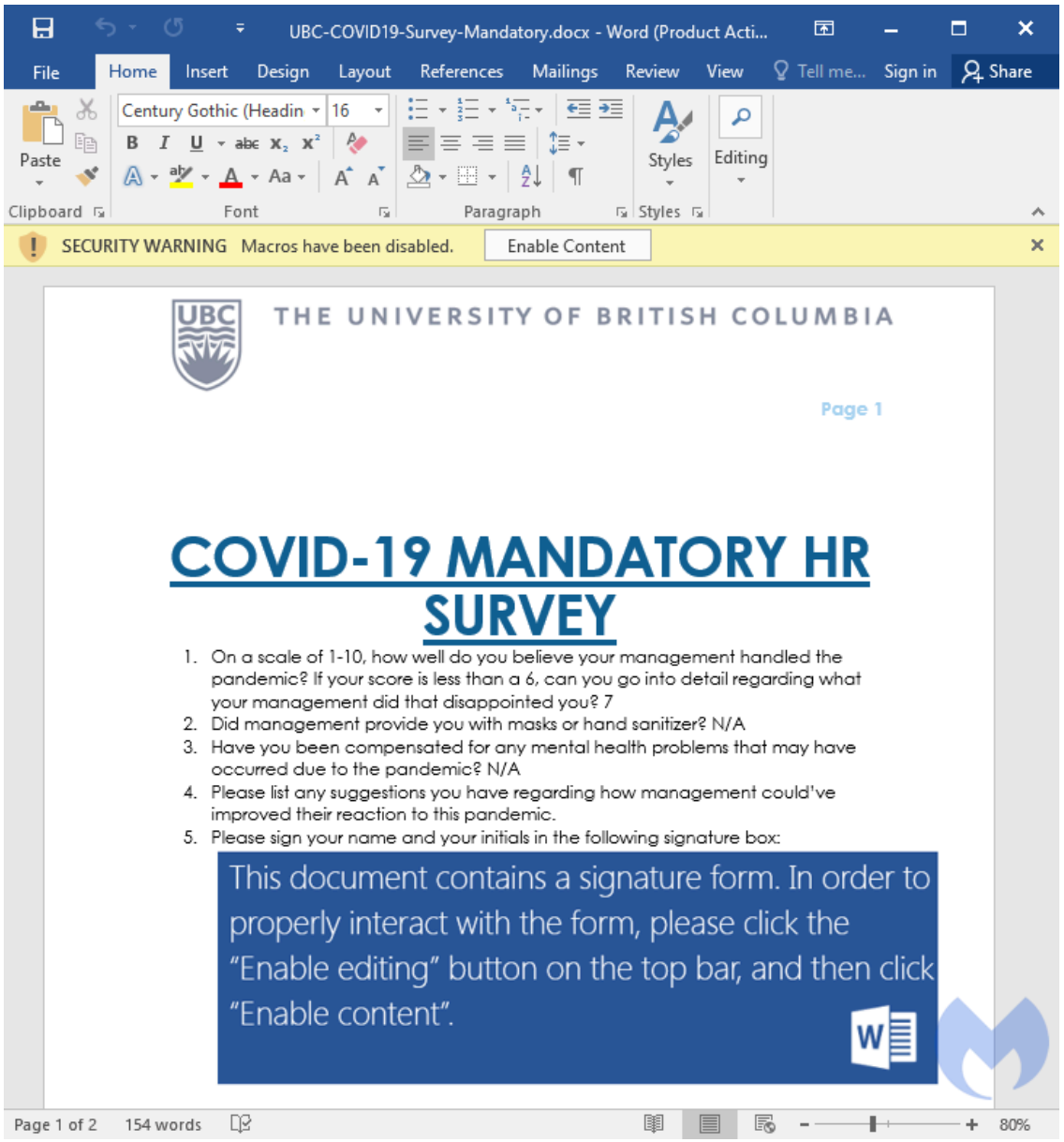
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

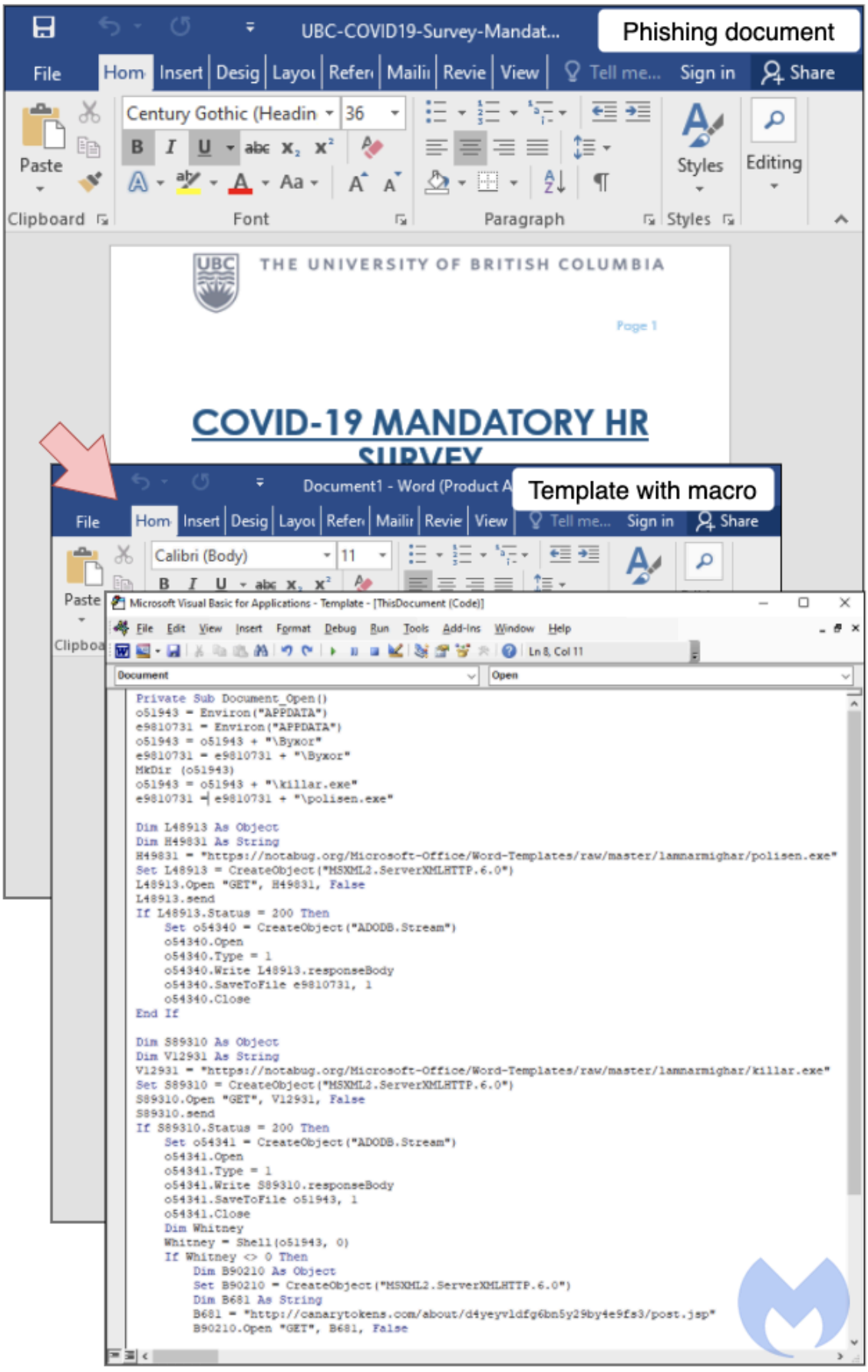
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

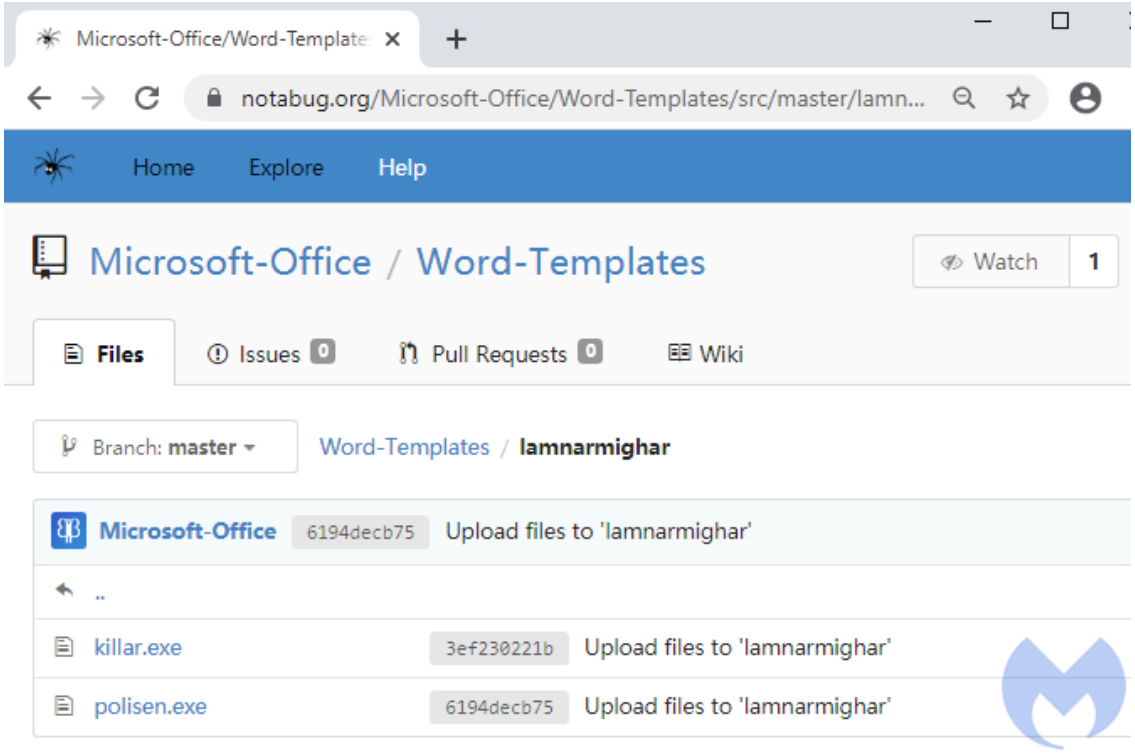
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



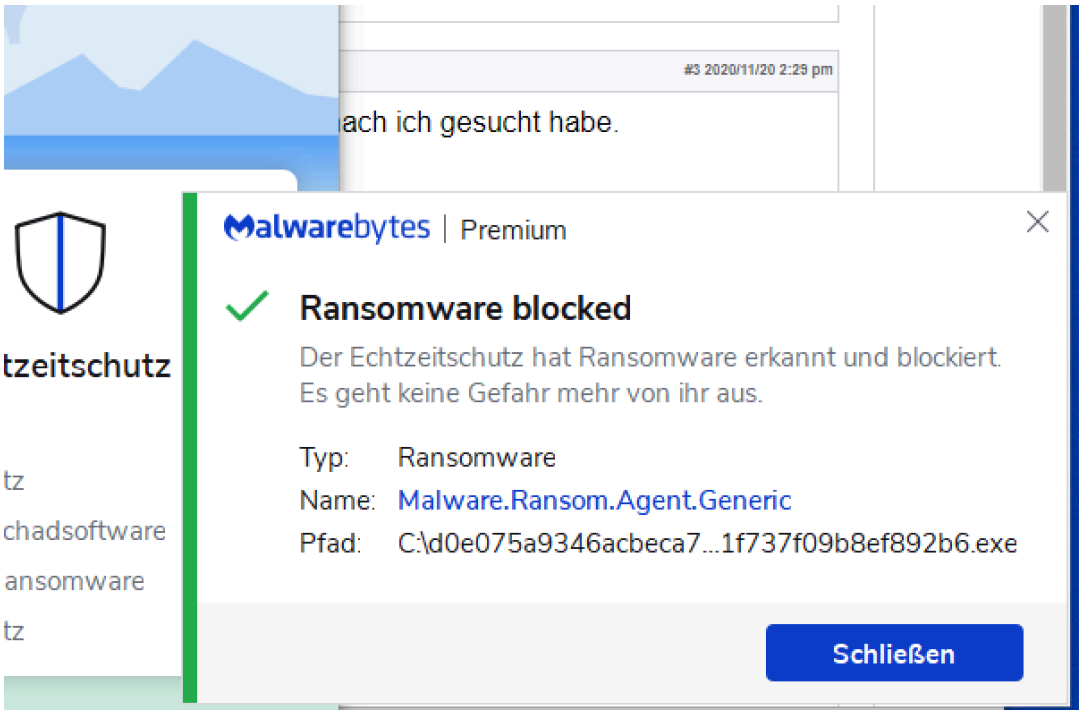
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

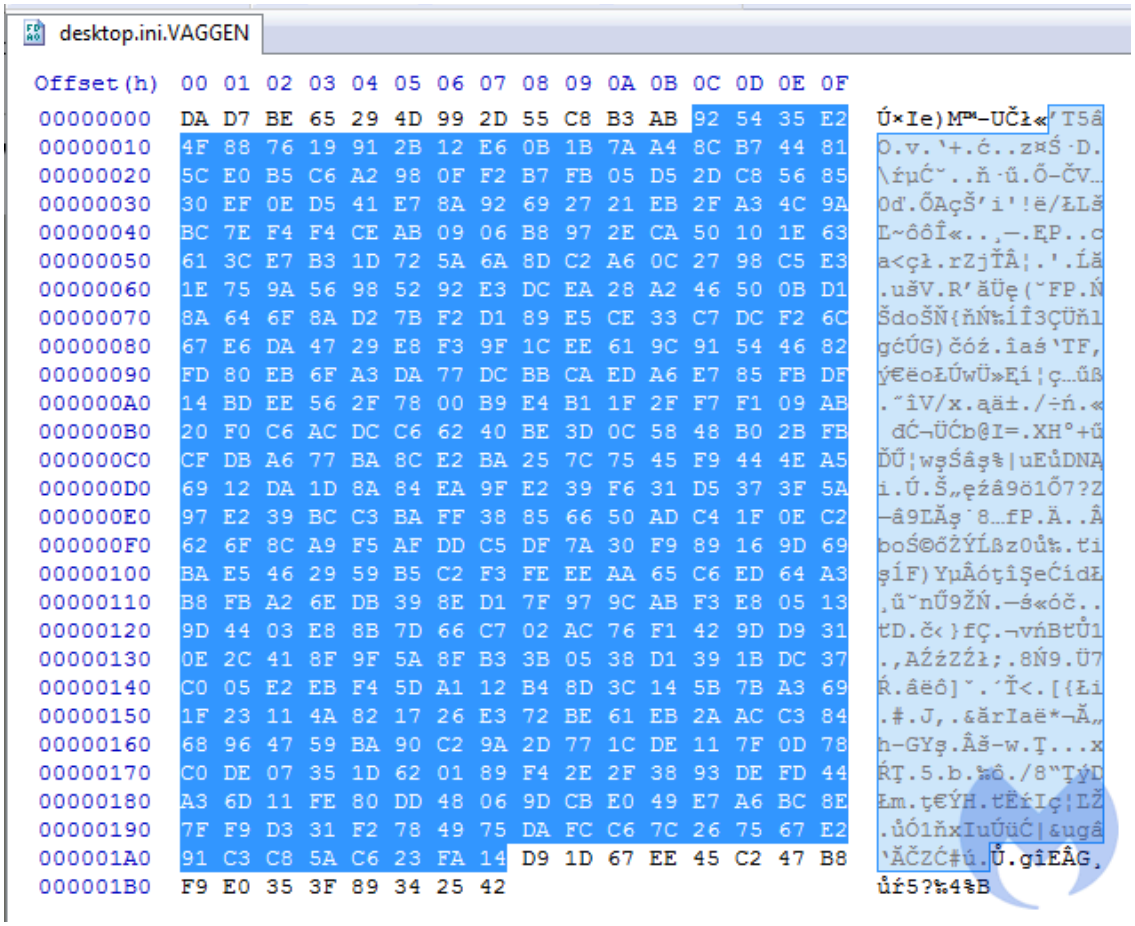
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

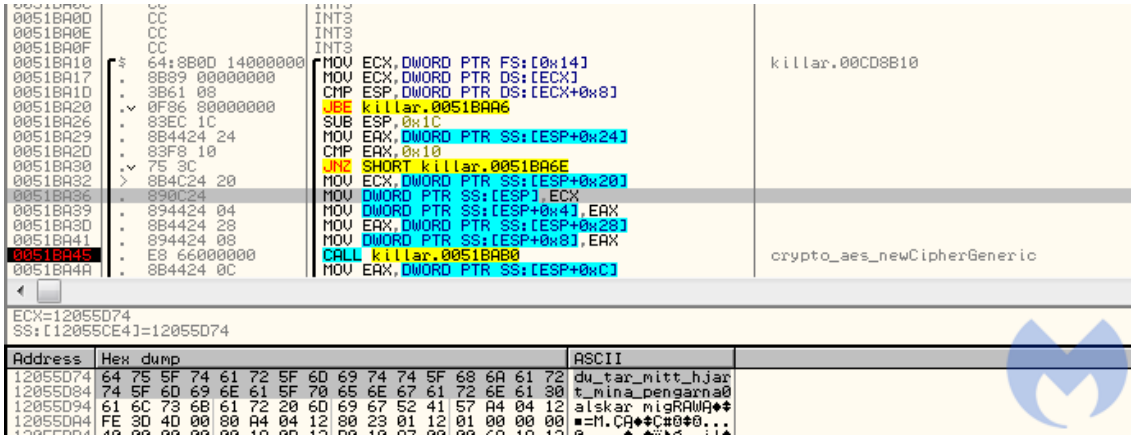


The content of the output file (with .VAGGEN extension) contains:

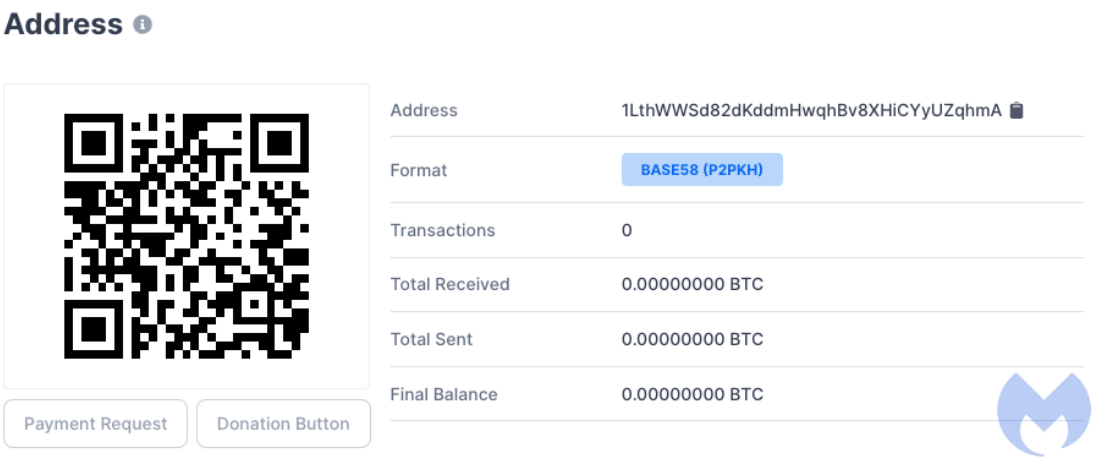
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

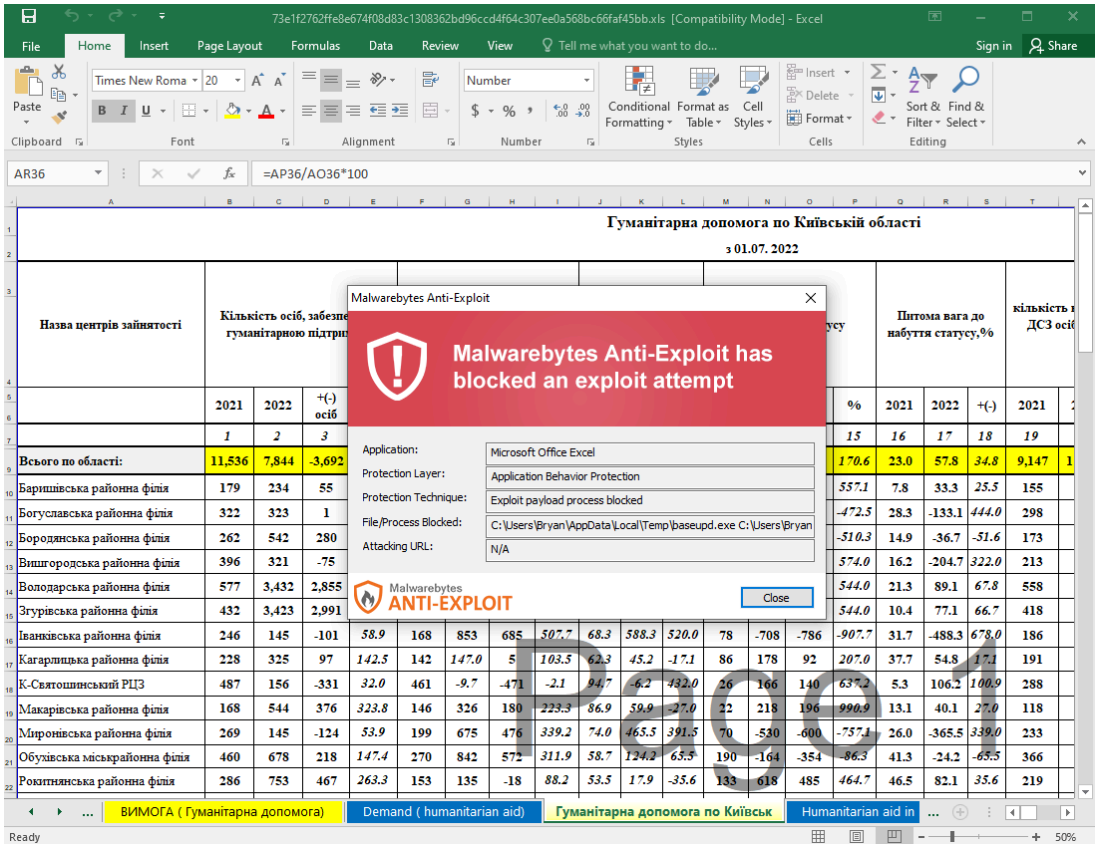
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

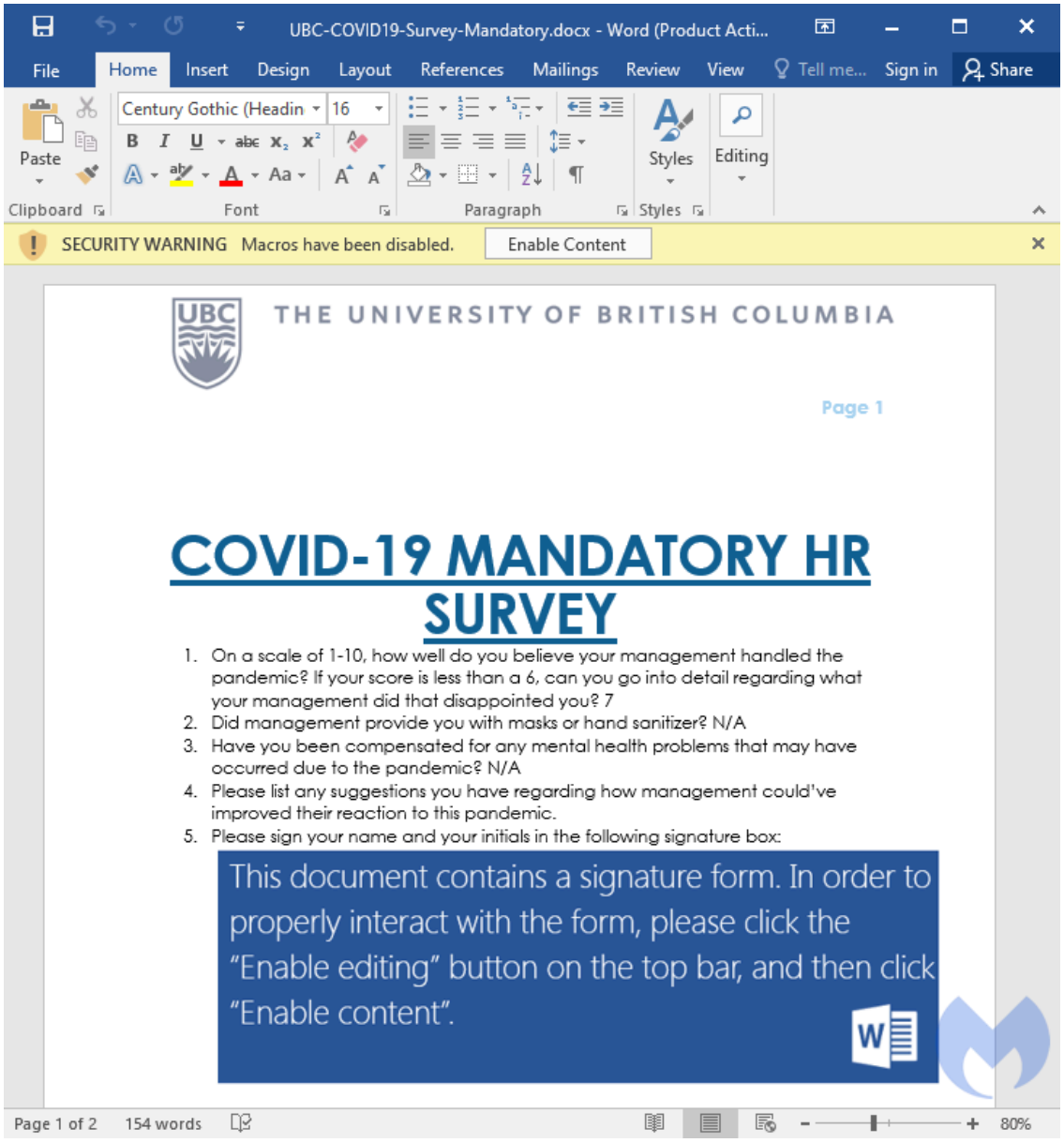
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

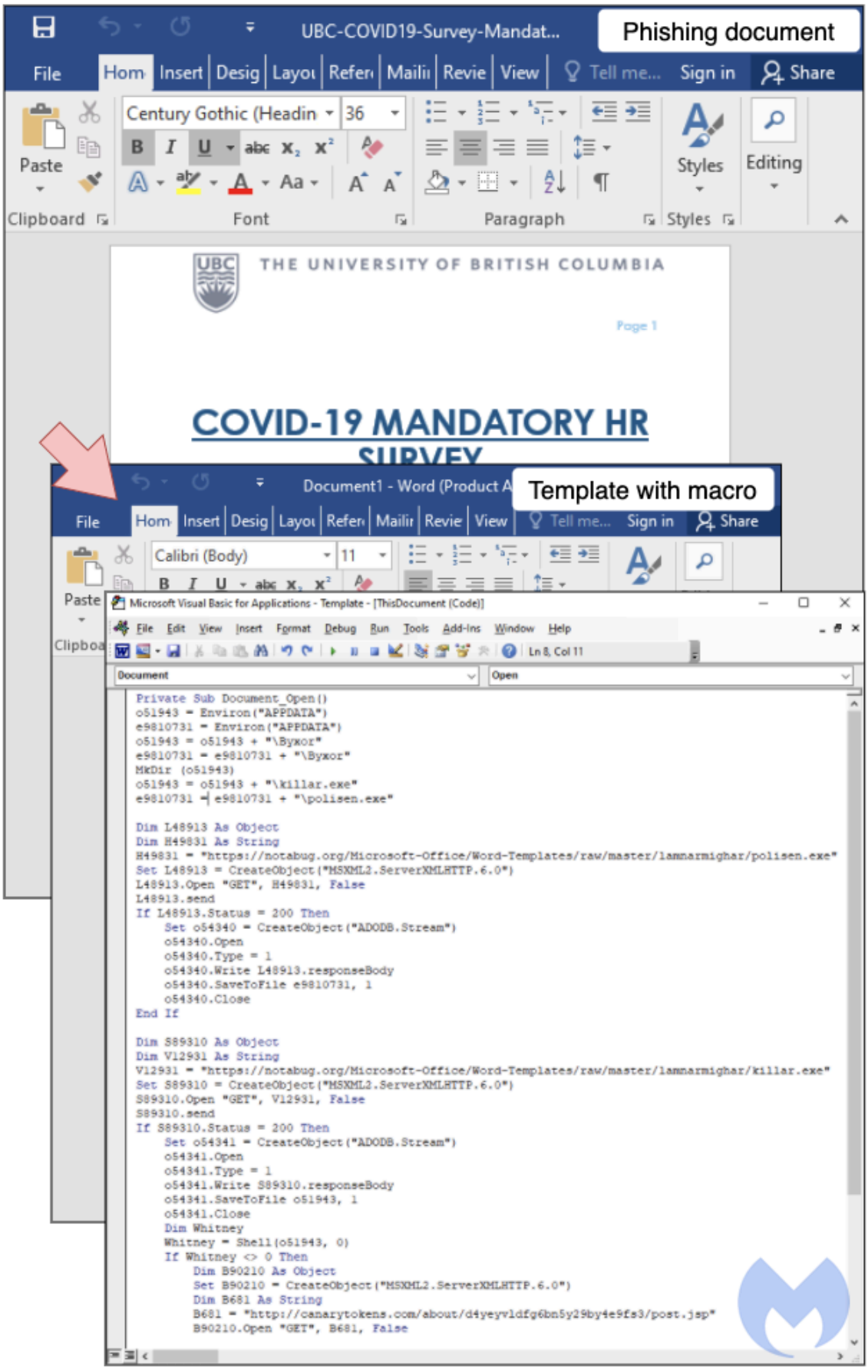
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

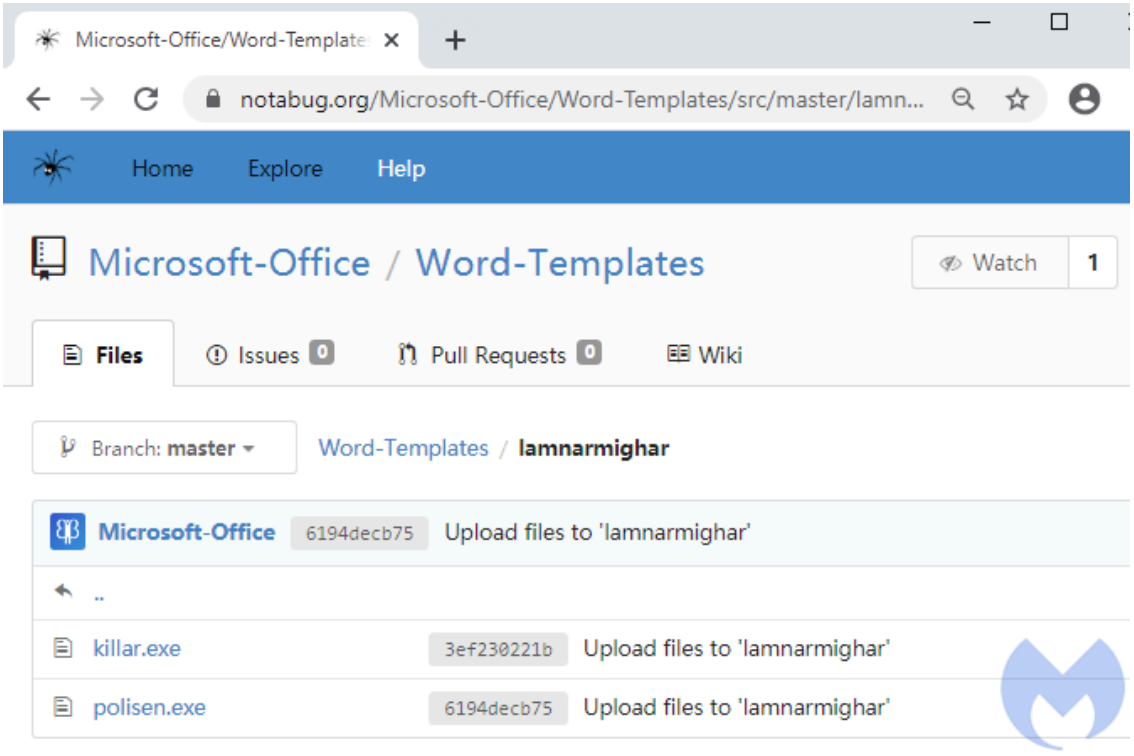
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



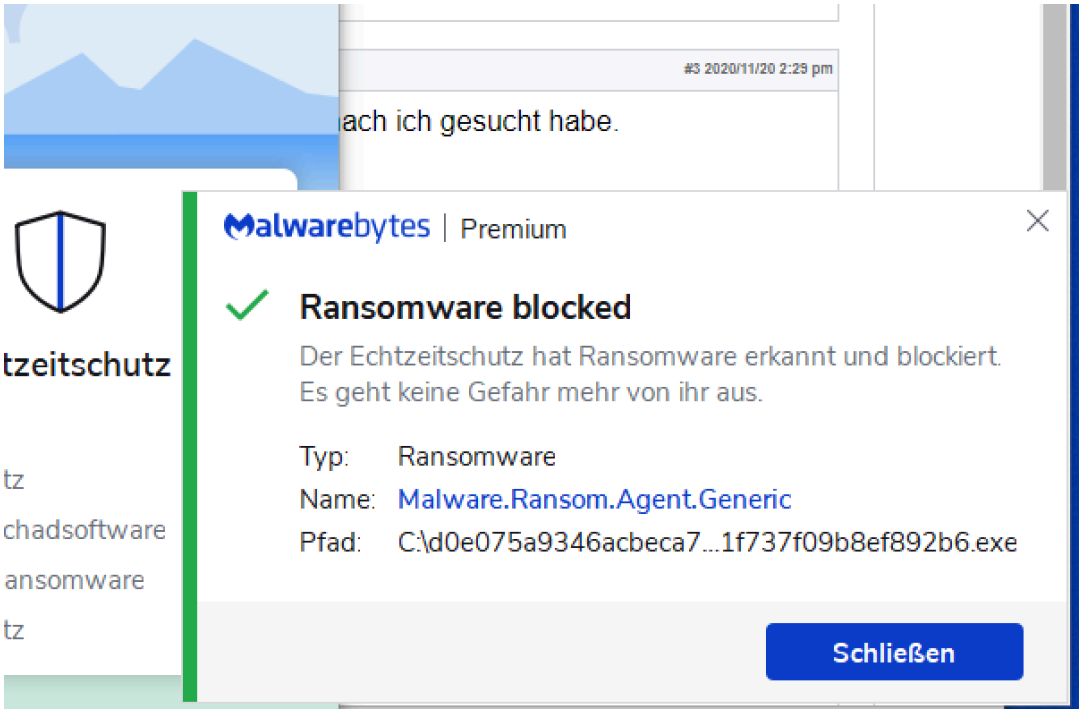
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

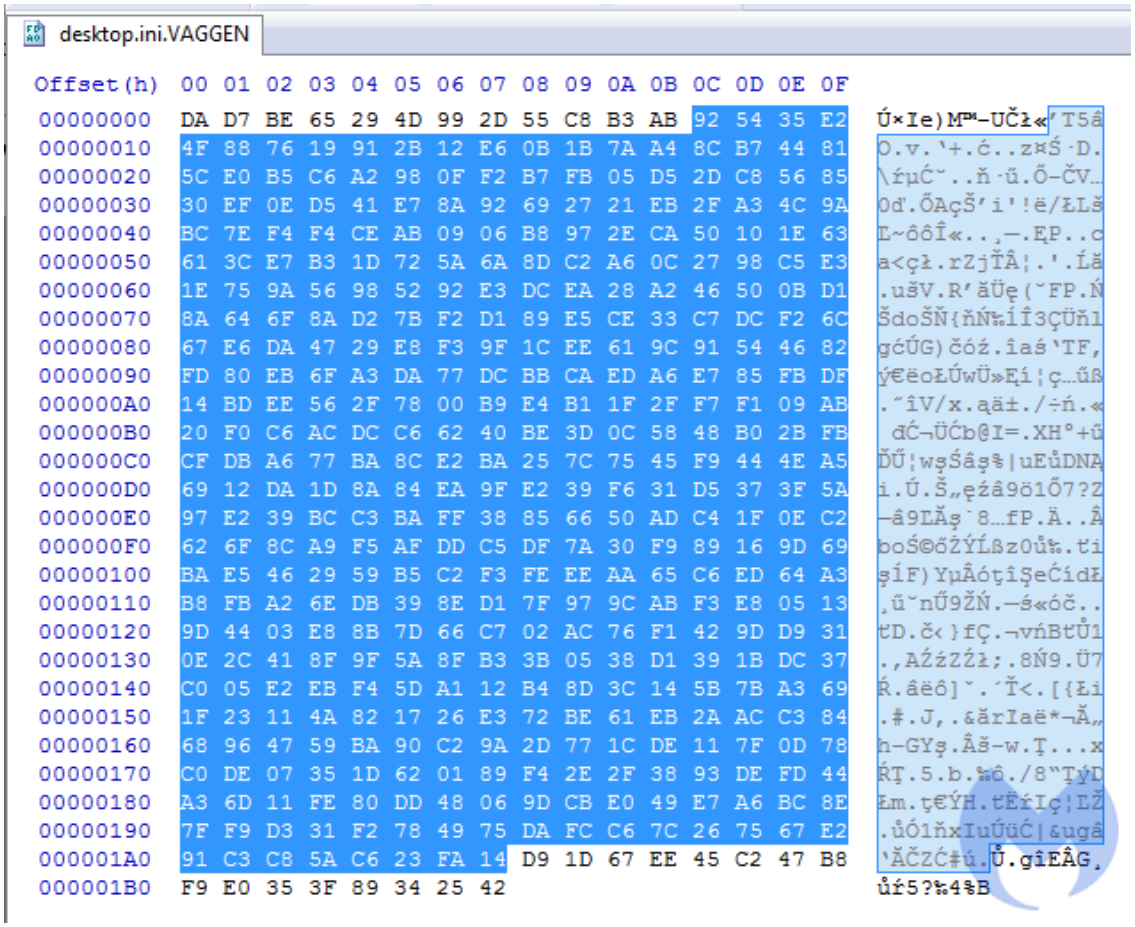
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

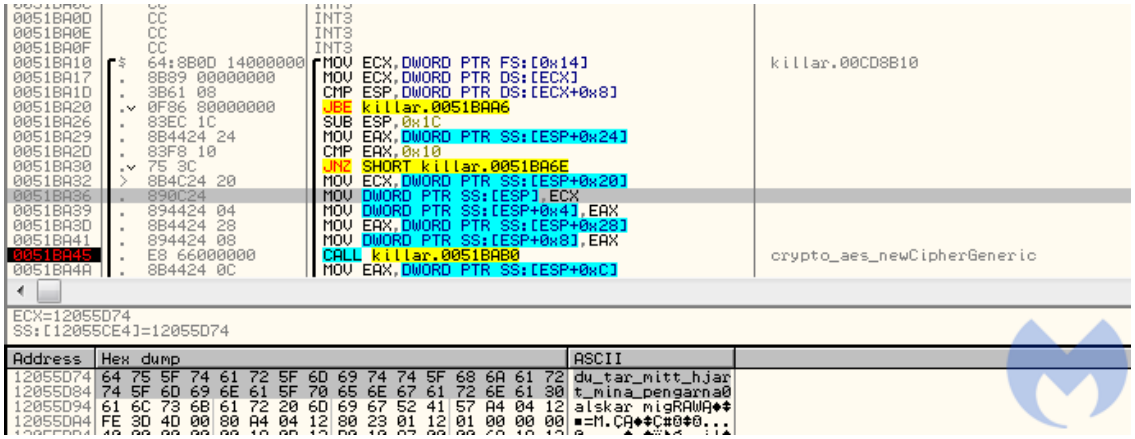


The content of the output file (with .VAGGEN extension) contains:

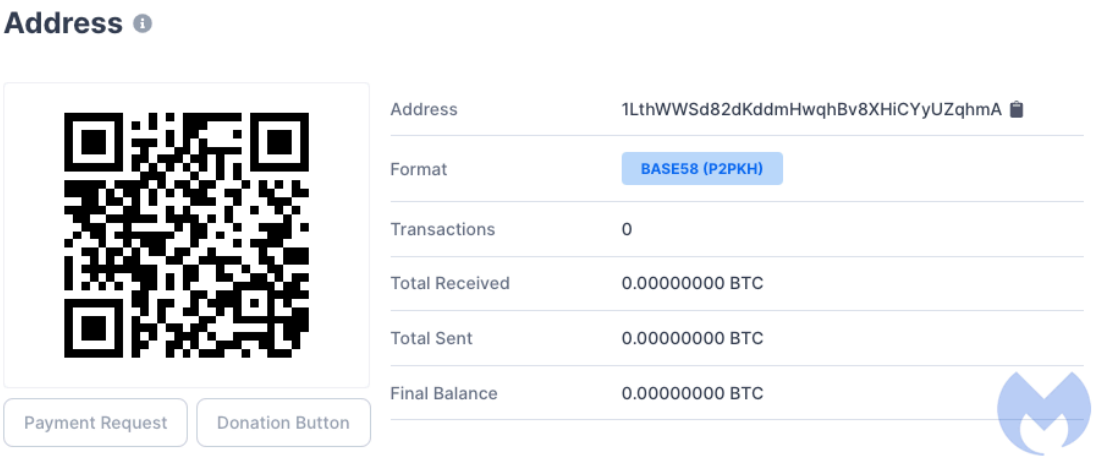
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

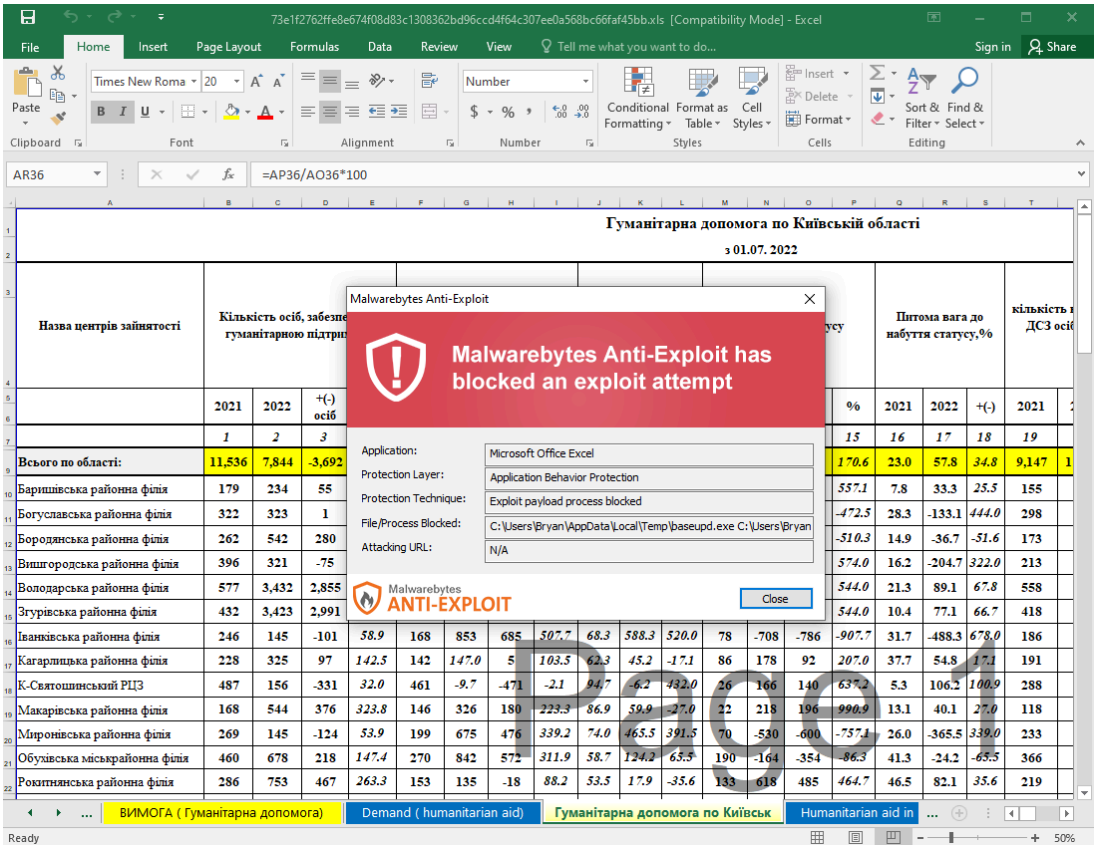
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

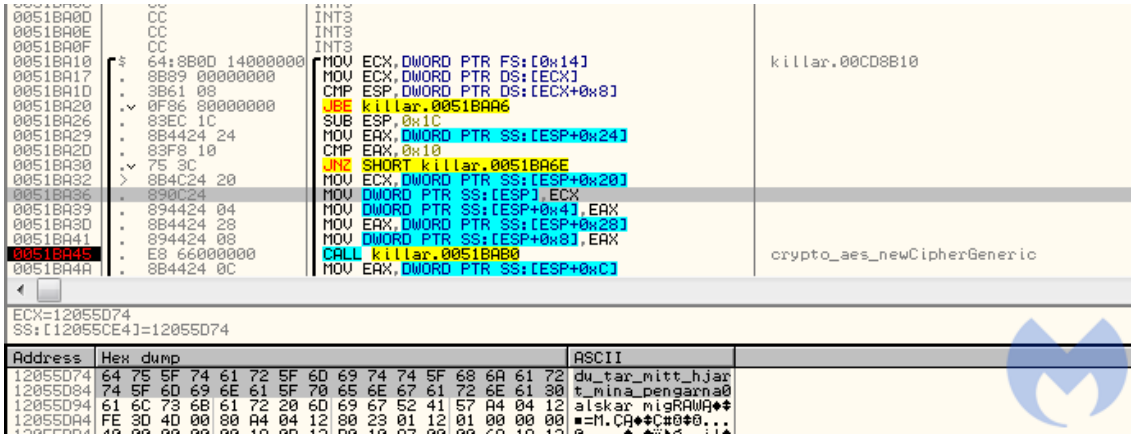
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

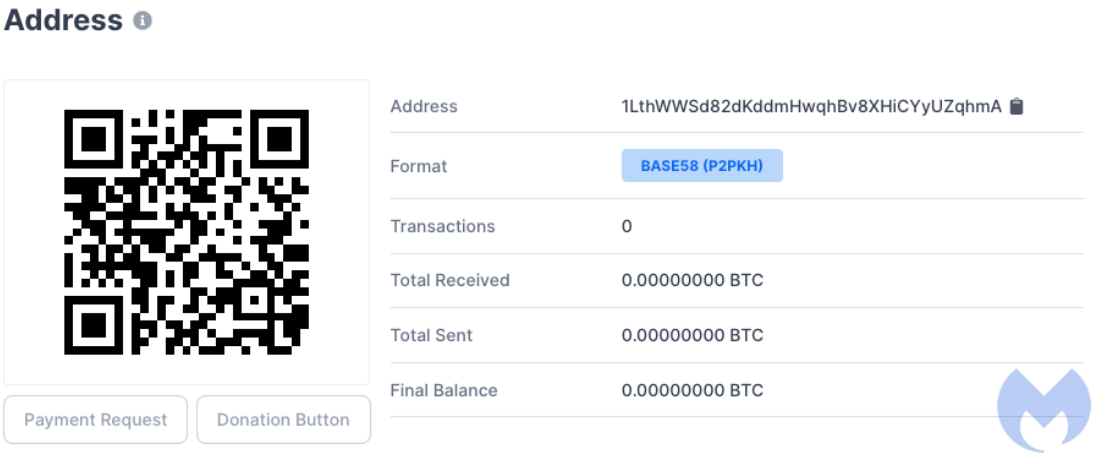
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

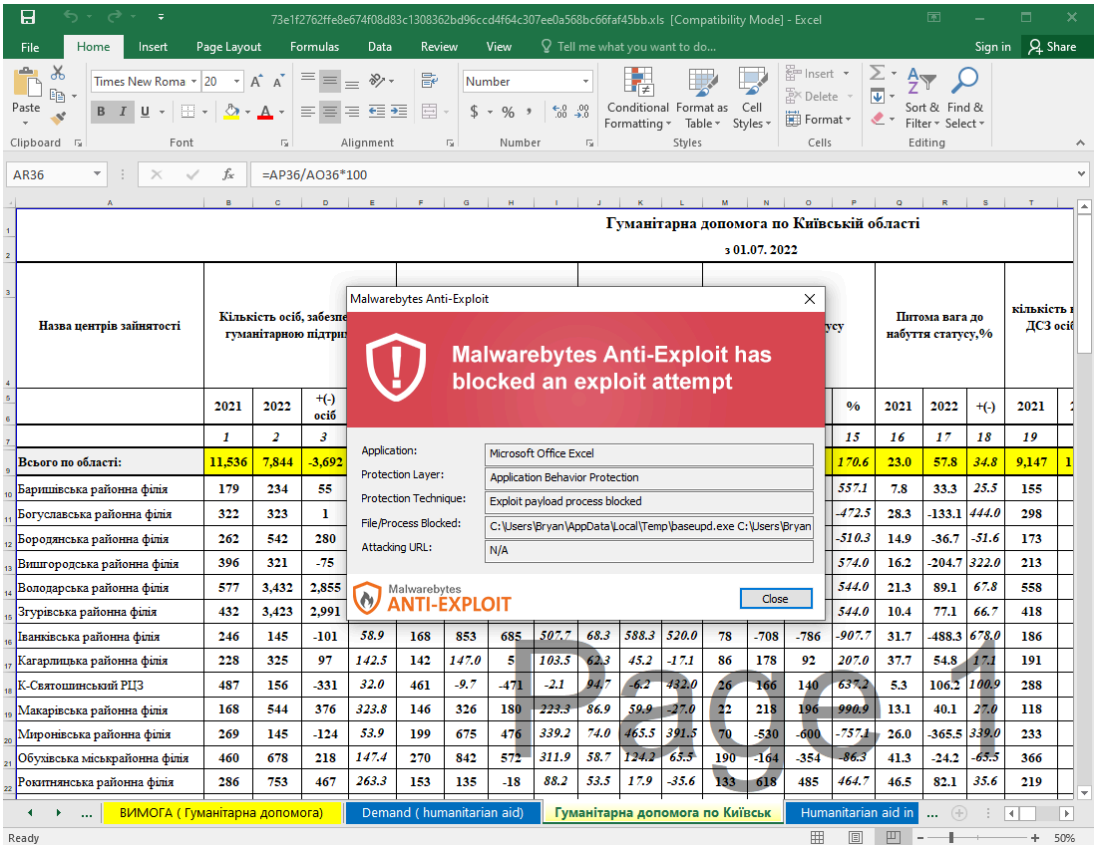
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
46  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
47    runtime_morestack_noctxt();
48  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
49  ((void (*)(void))loc_45CDA0)();
50  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
51  v38 = v4;
52  v30 = v5;
53  v31 = v6;
54  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
55  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
56  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
57  v32 = v10;
58  v11 = *(void (__cdecl **)(int))(v10 + 16);
59  v39 = v12;
60  v11(v12);
61  v29 = v13;
62  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
63  v40 = v15;
64  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
65  v17 = v27;
66  if ( v27 )
67  {
68    if ( v27 )
69      v17 = *(_DWORD *)(v27 + 4);
70    v43 = v17;
71    v44 = v28;
72    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
73  }
```

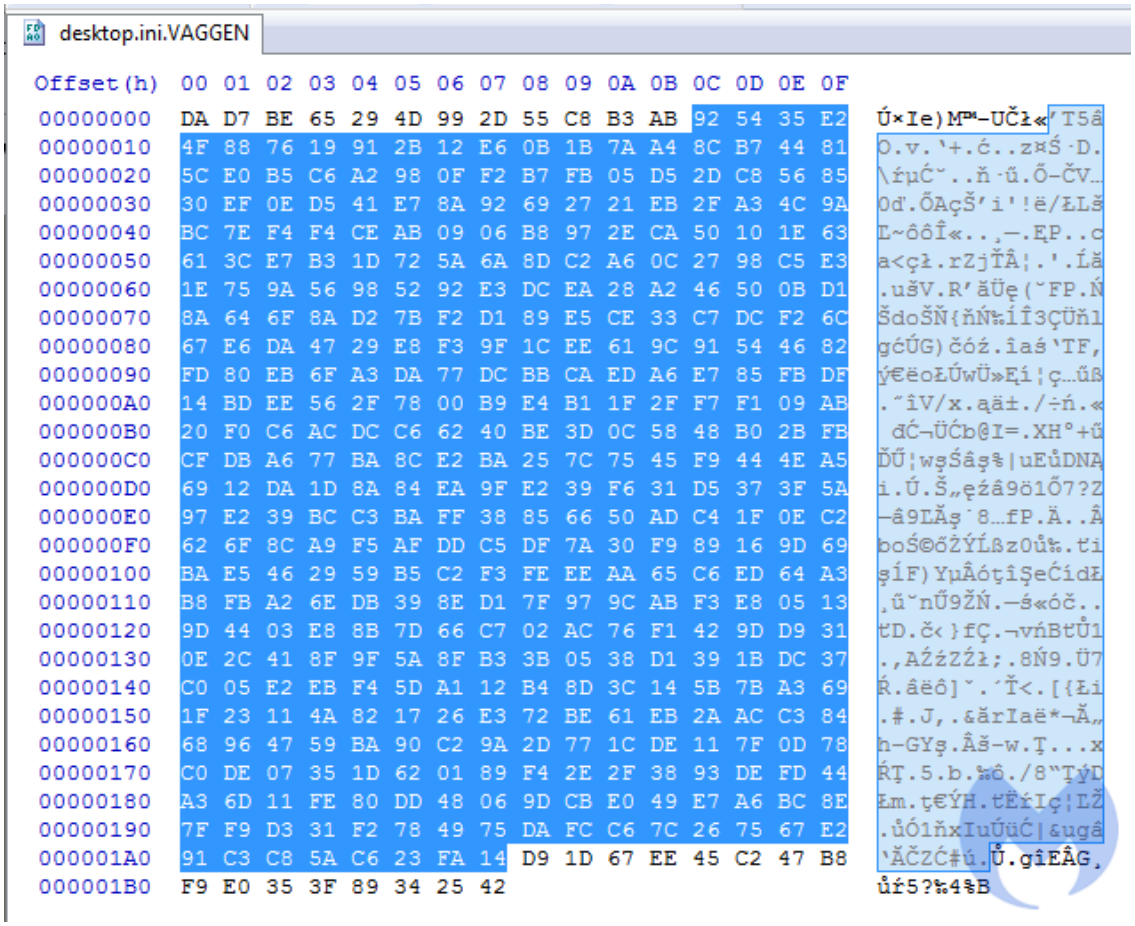


The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

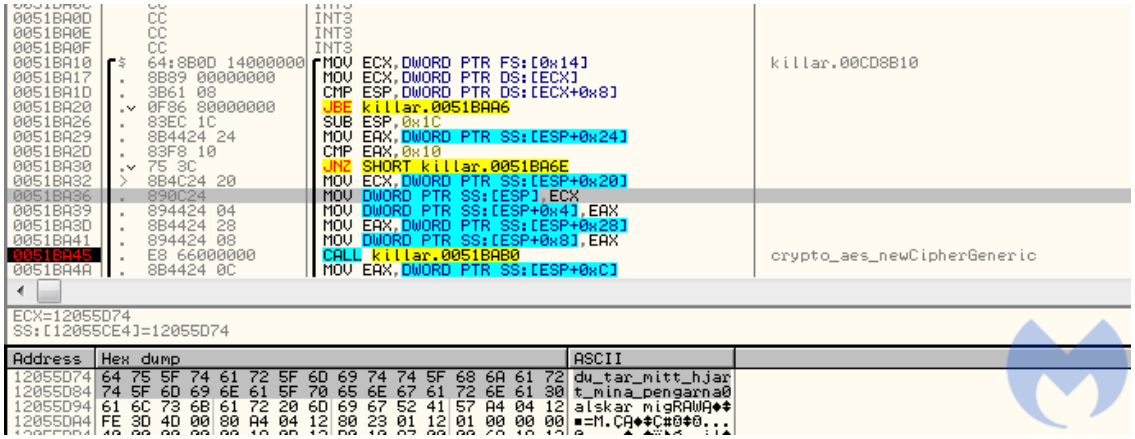


The content of the output file (with .VAGGEN extension) contains:

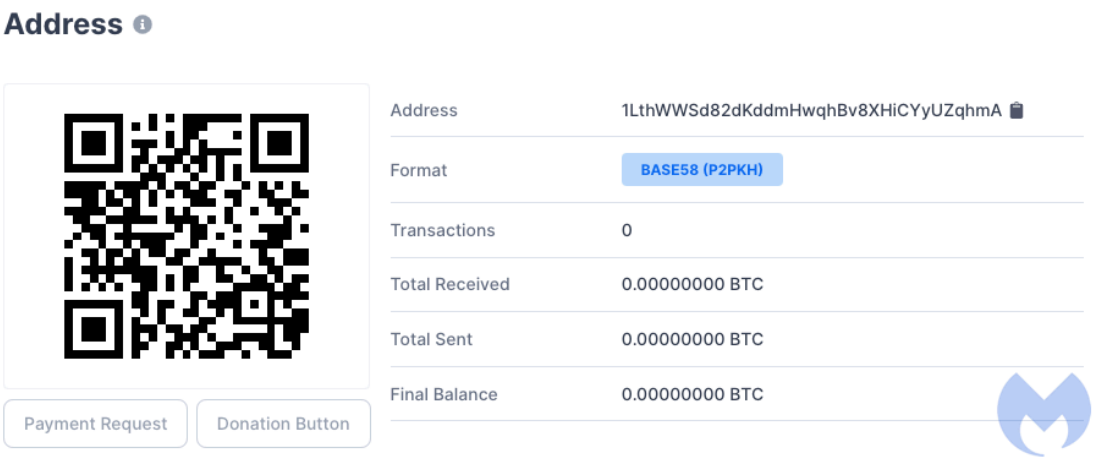
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

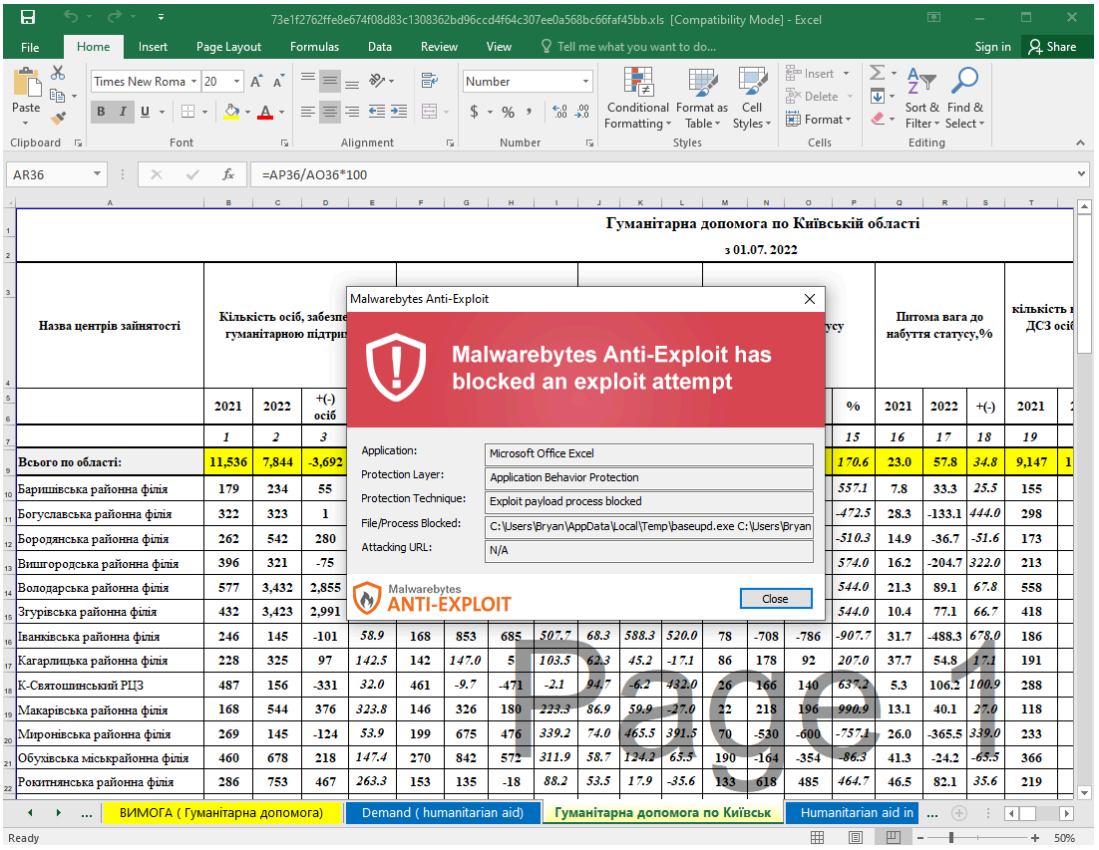
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

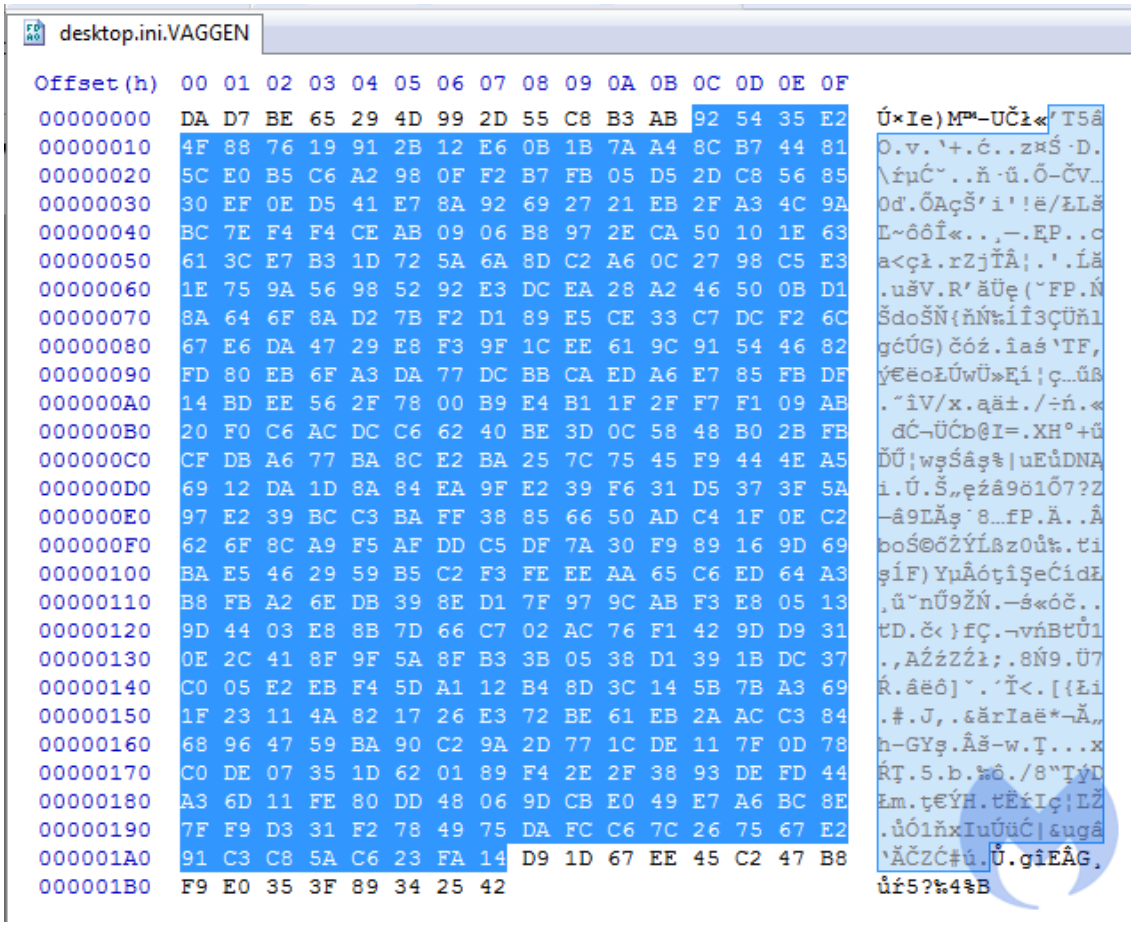
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

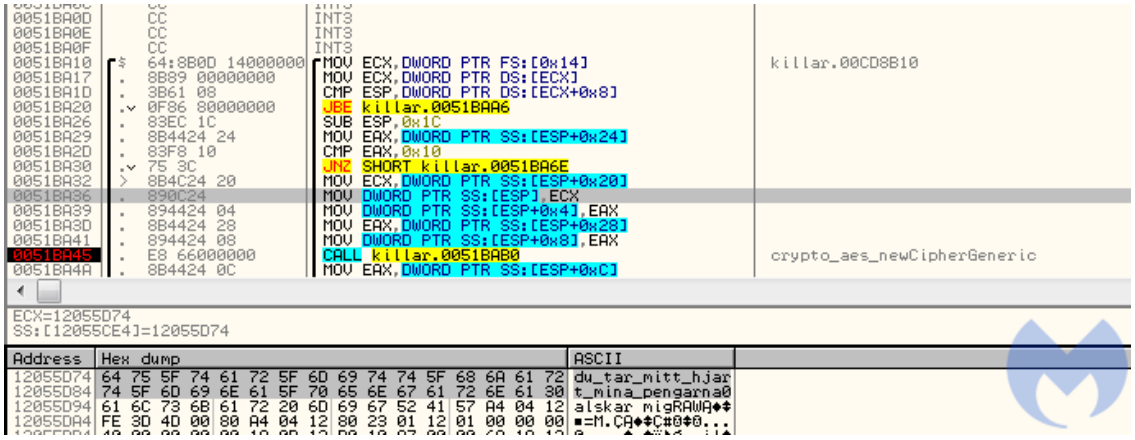


The content of the output file (with .VAGGEN extension) contains:

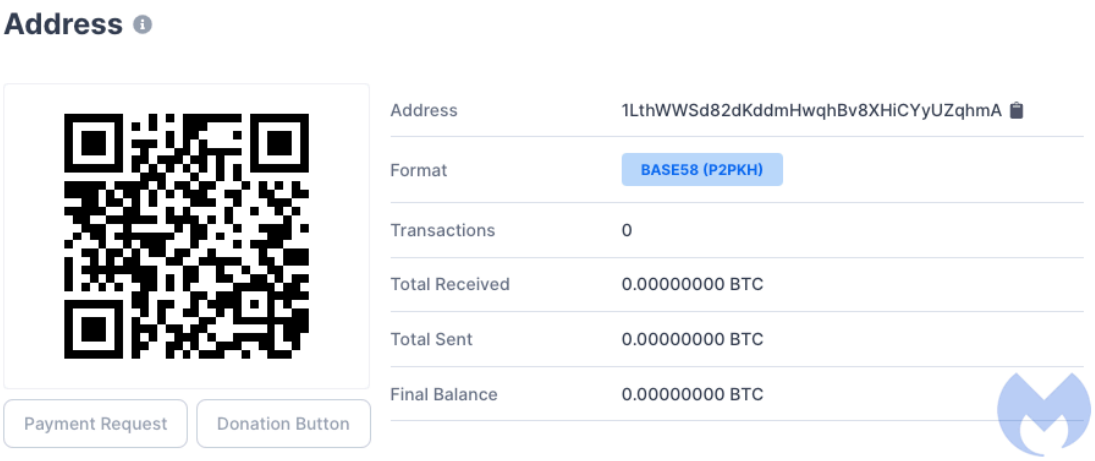
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

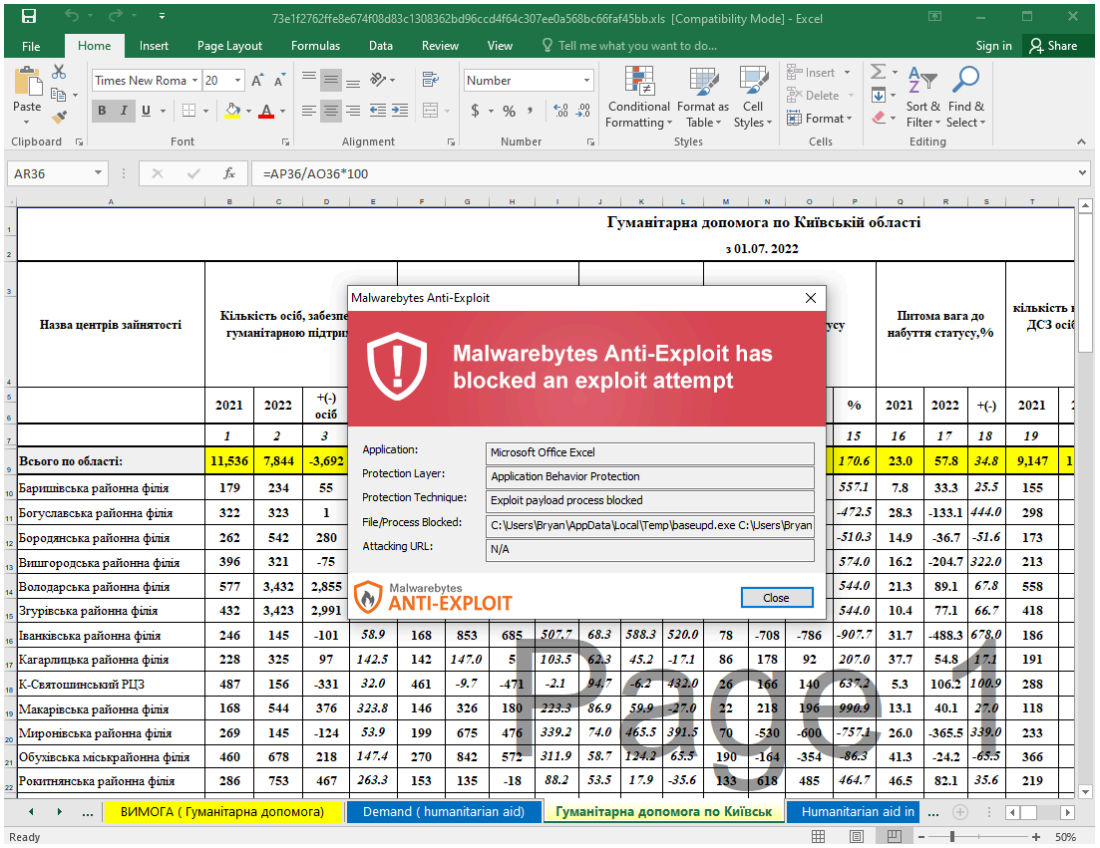
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

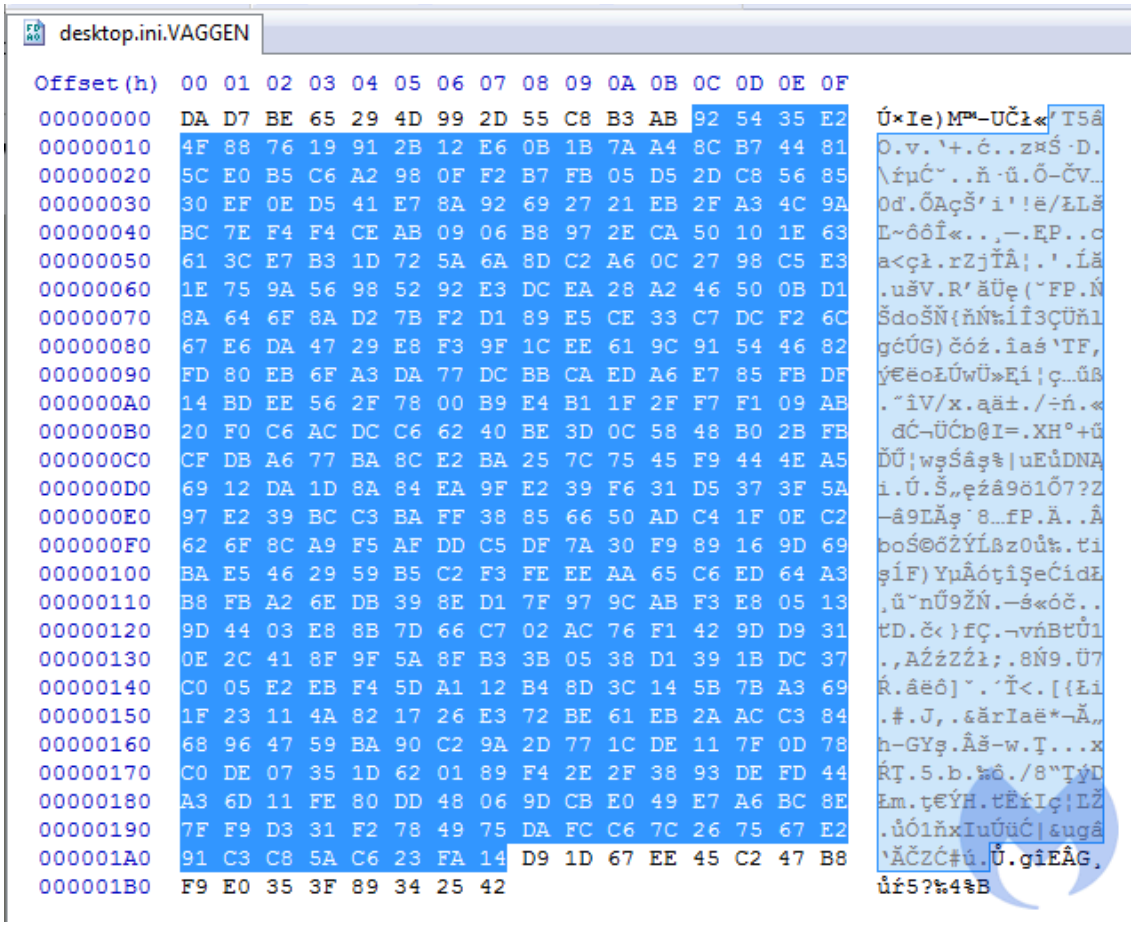
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

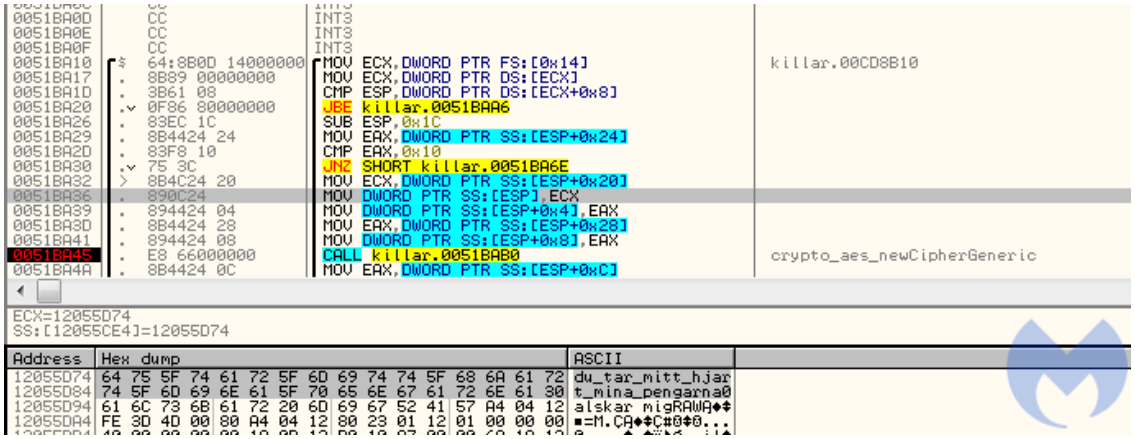


The content of the output file (with .VAGGEN extension) contains:

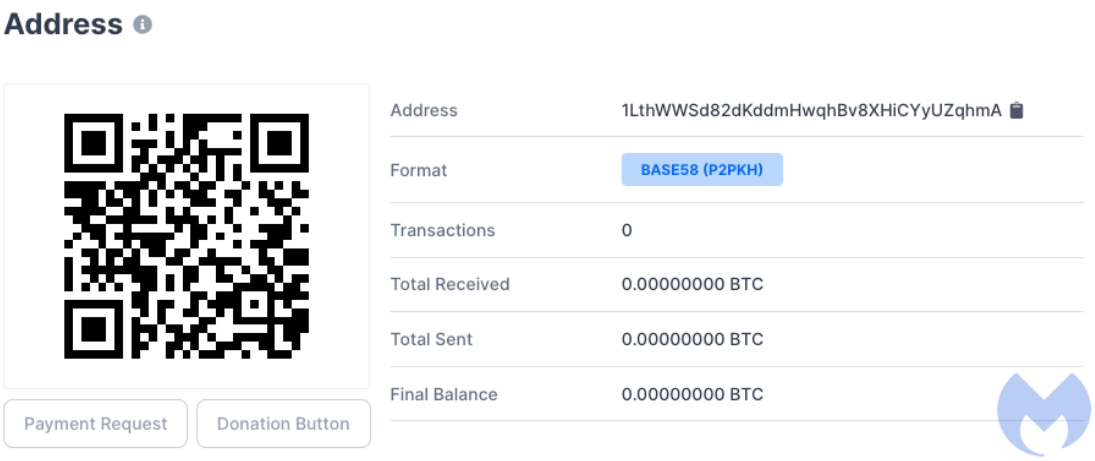
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

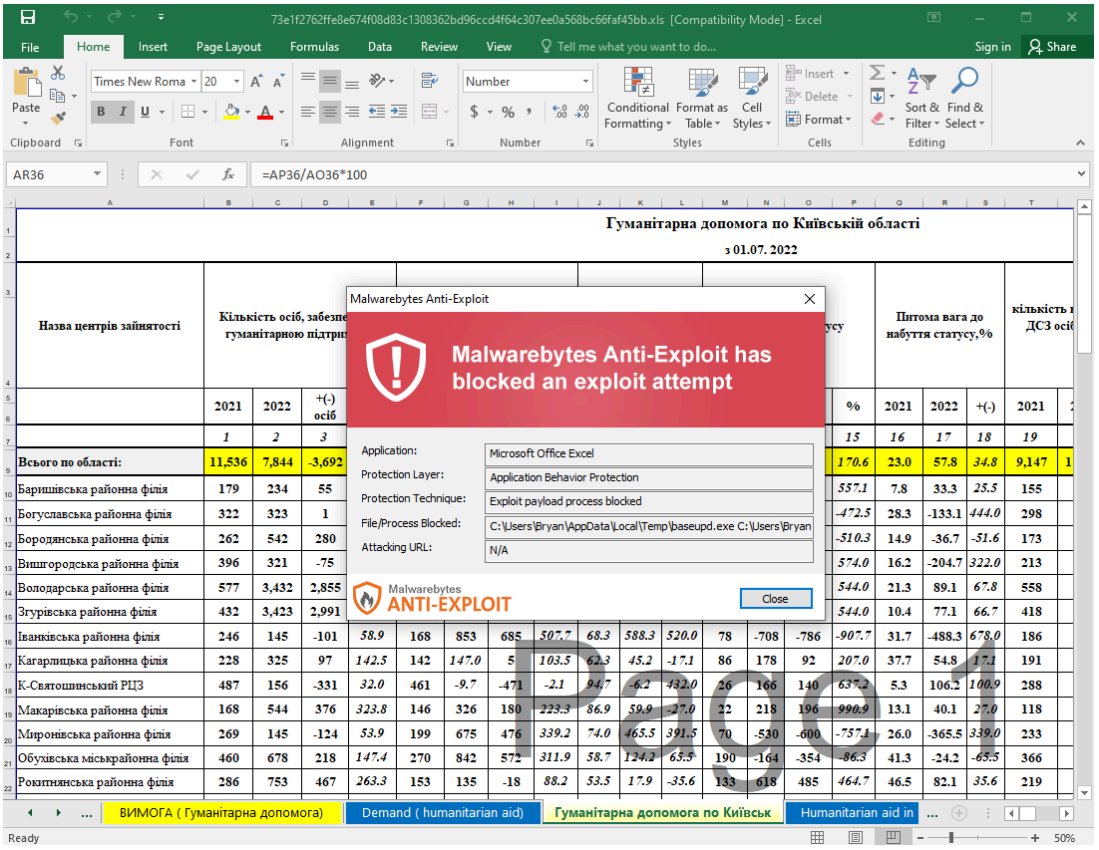
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

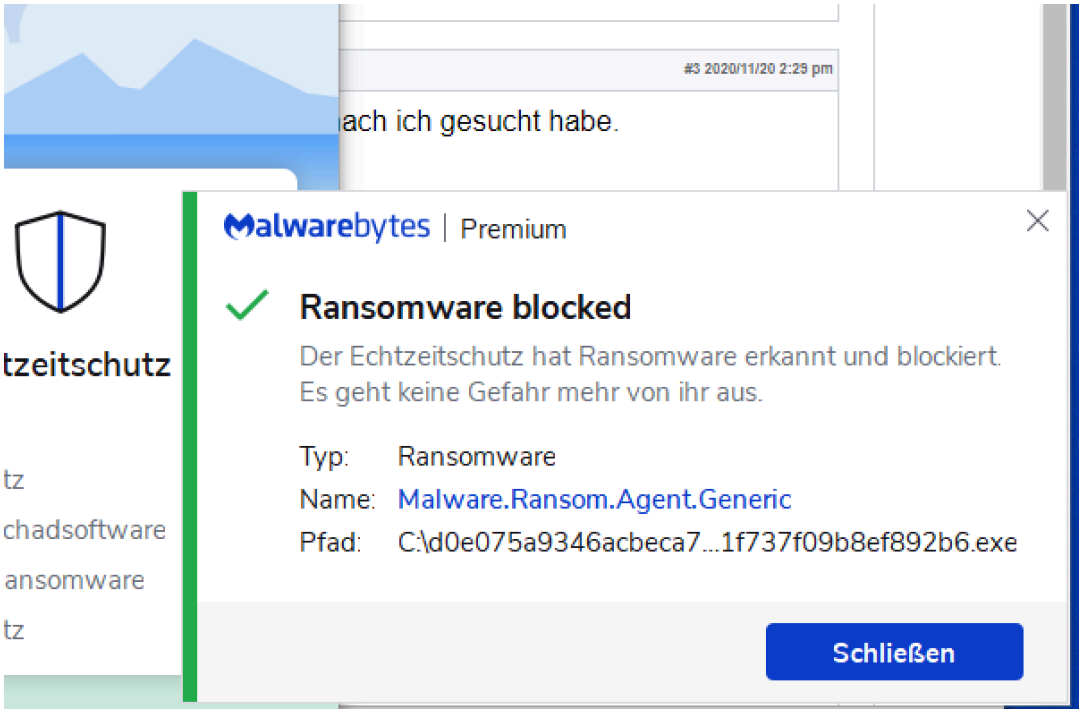
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

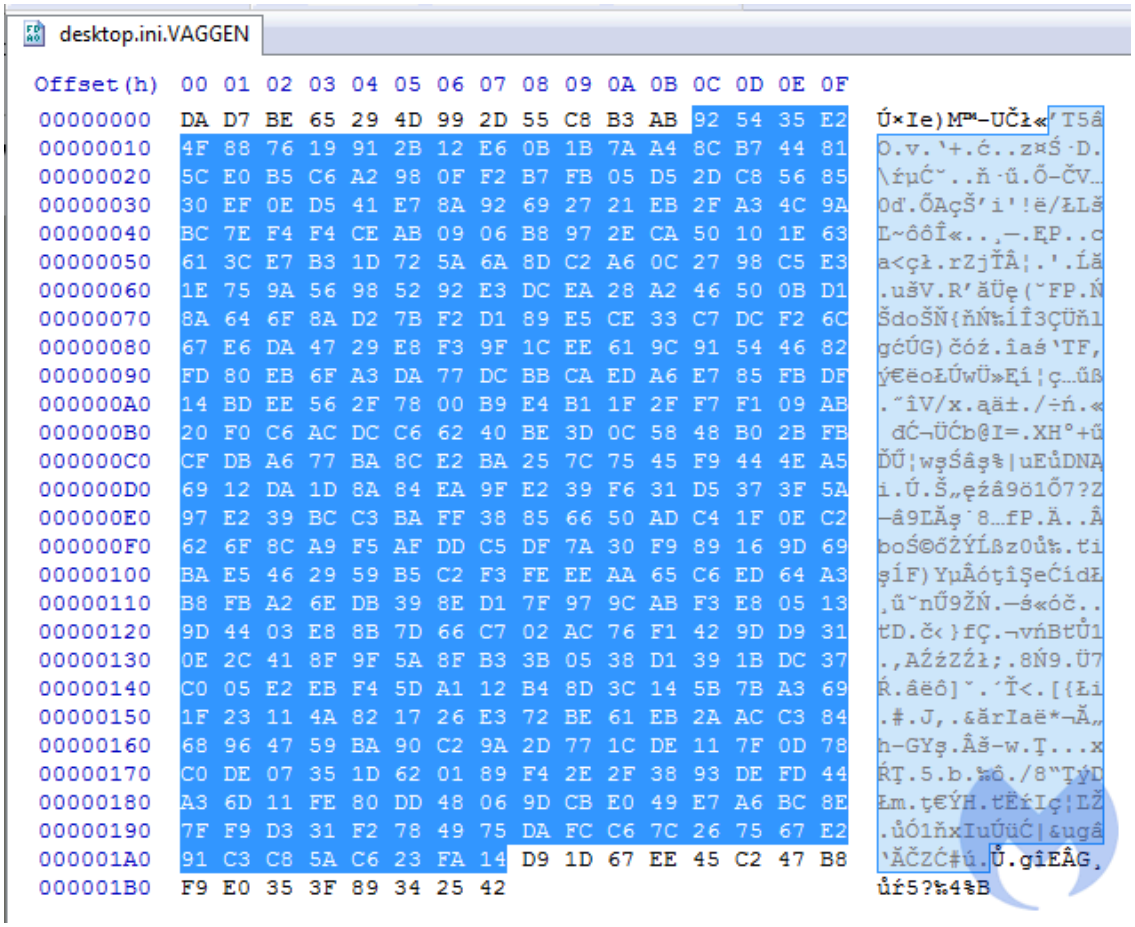
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

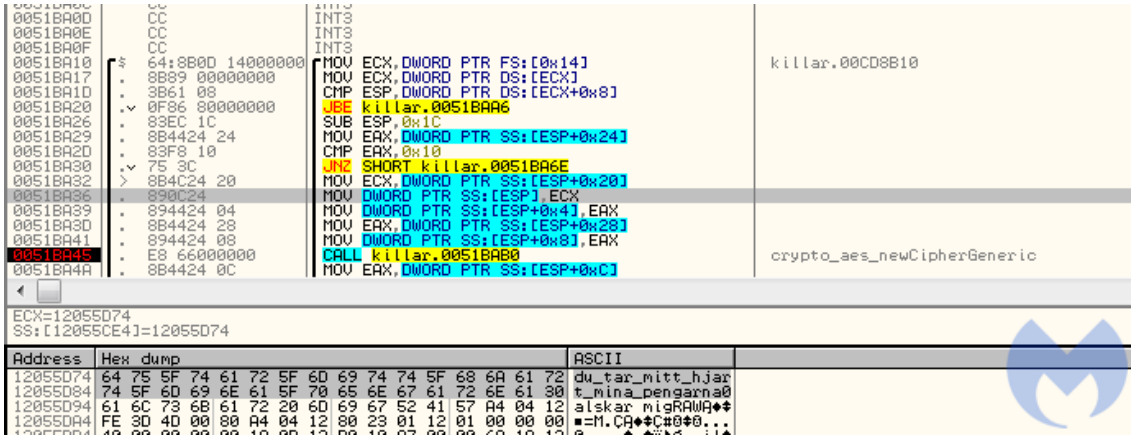


The content of the output file (with .VAGGEN extension) contains:

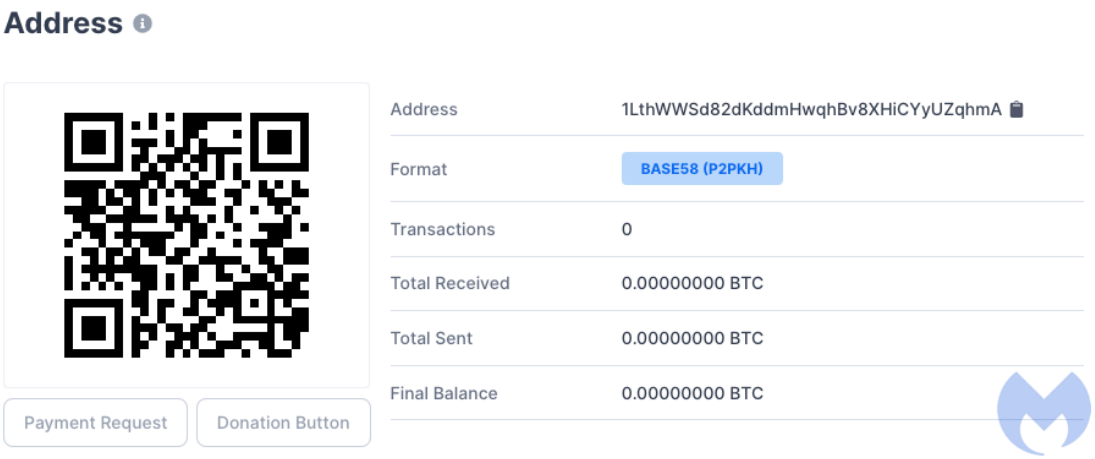
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

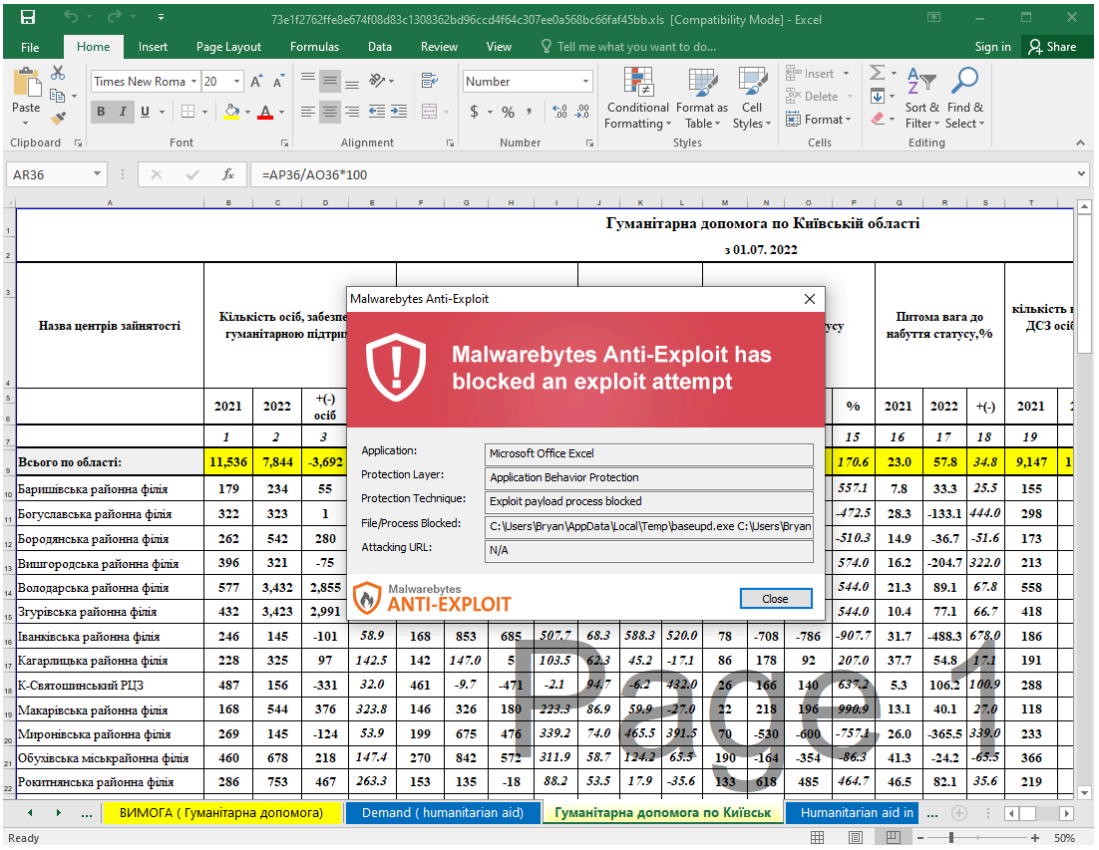
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

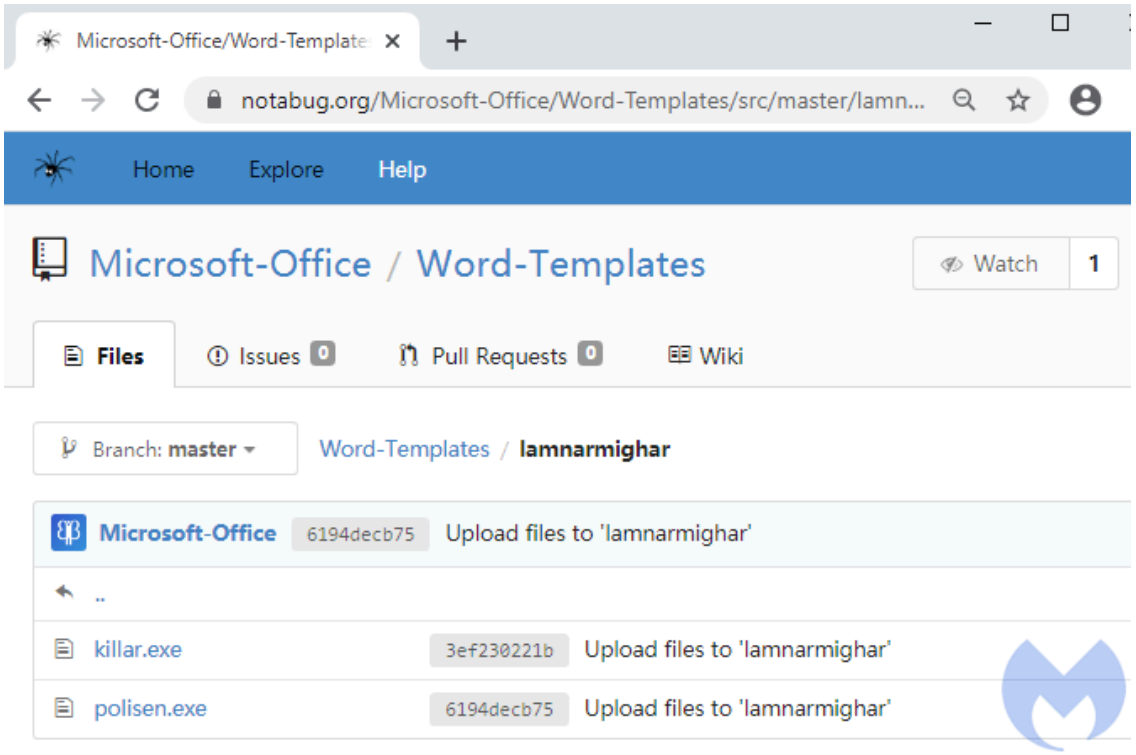
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



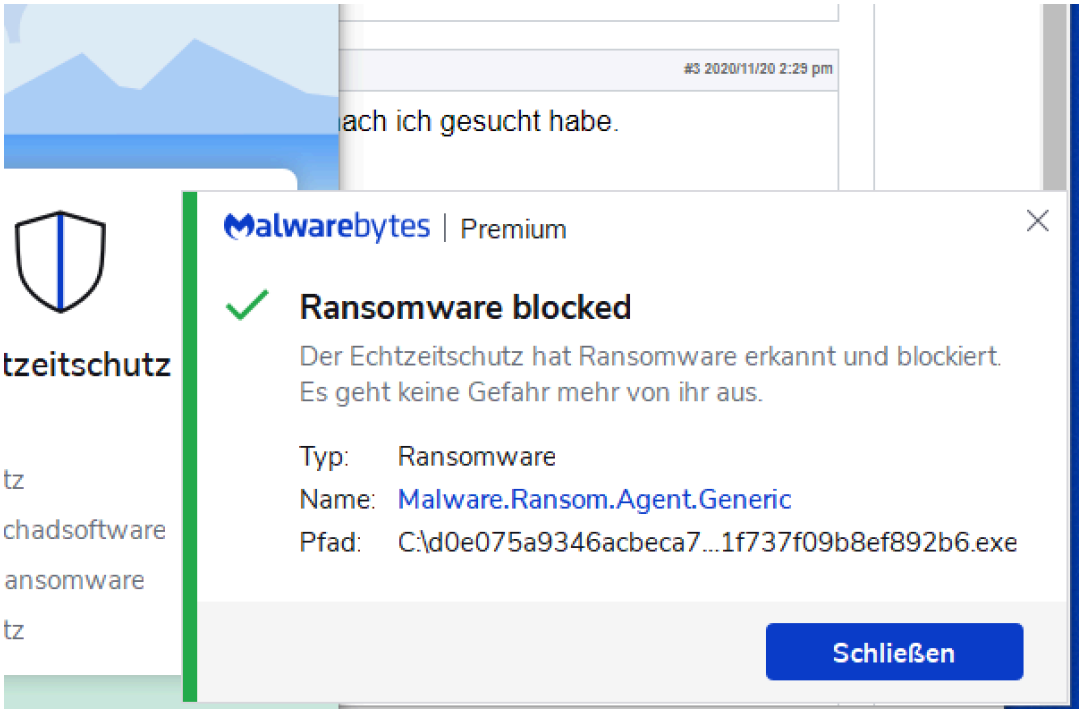
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

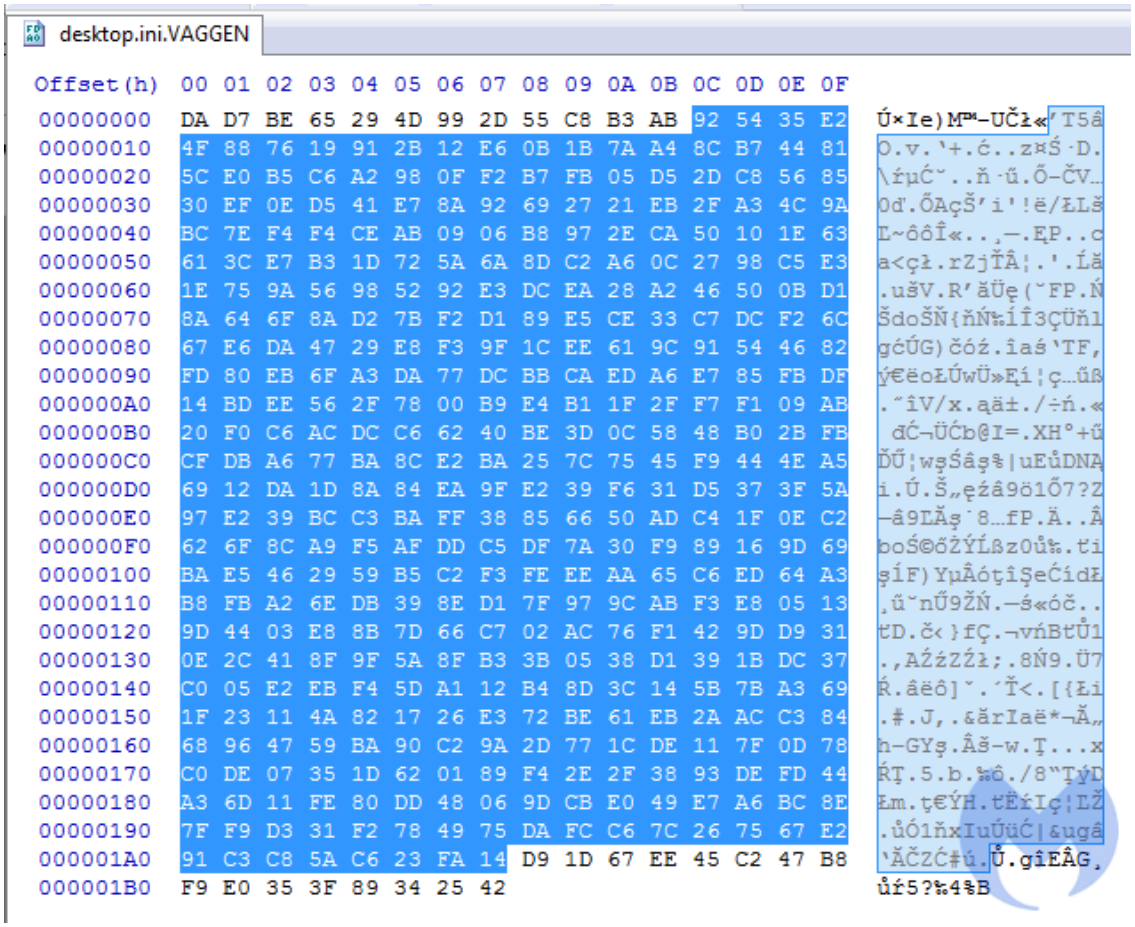
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

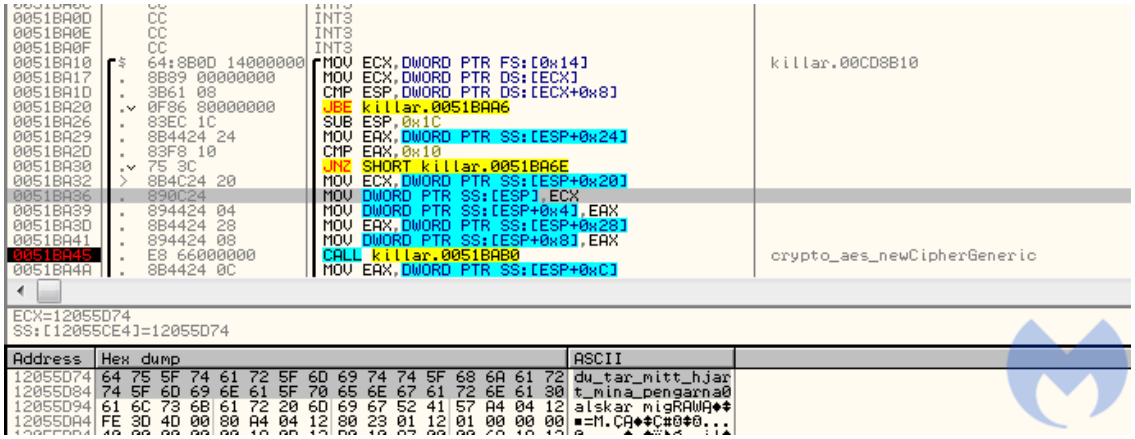


The content of the output file (with .VAGGEN extension) contains:

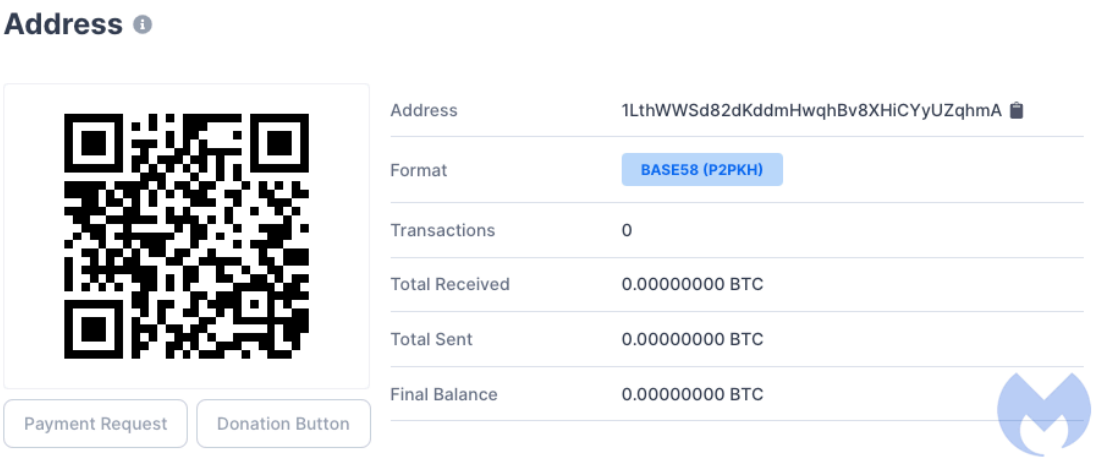
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

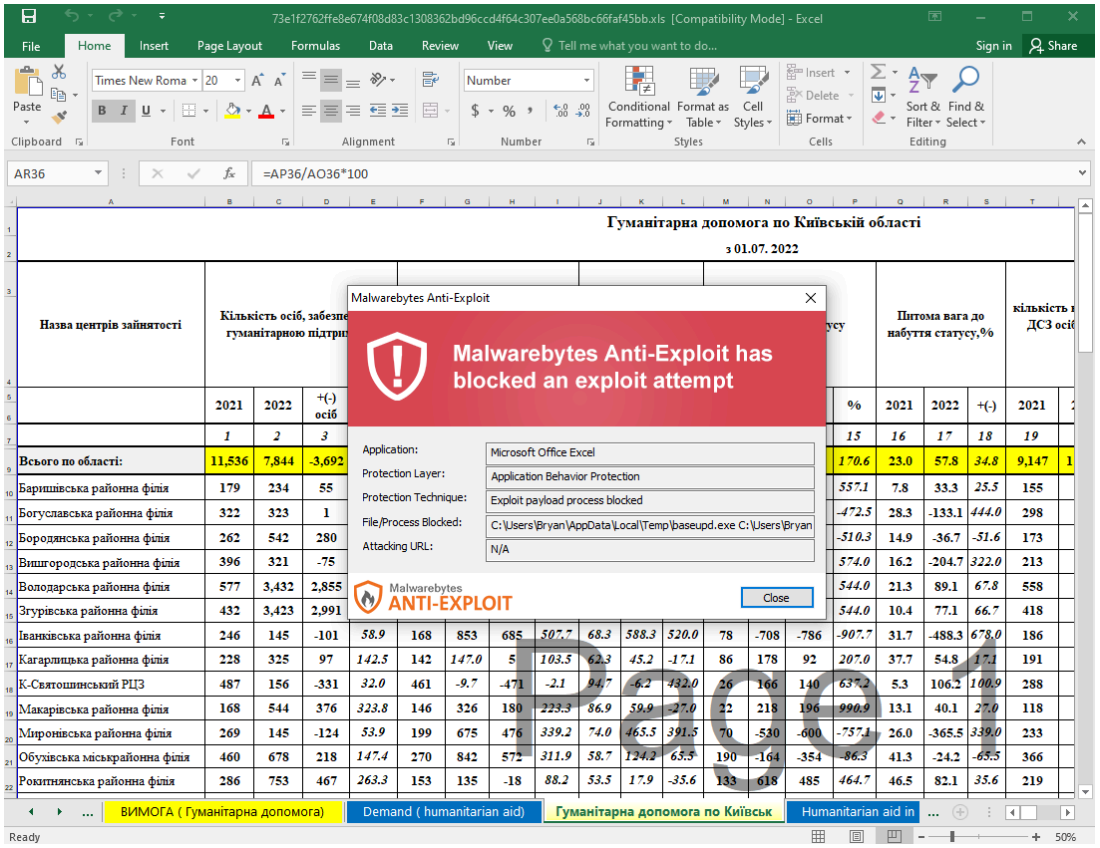
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

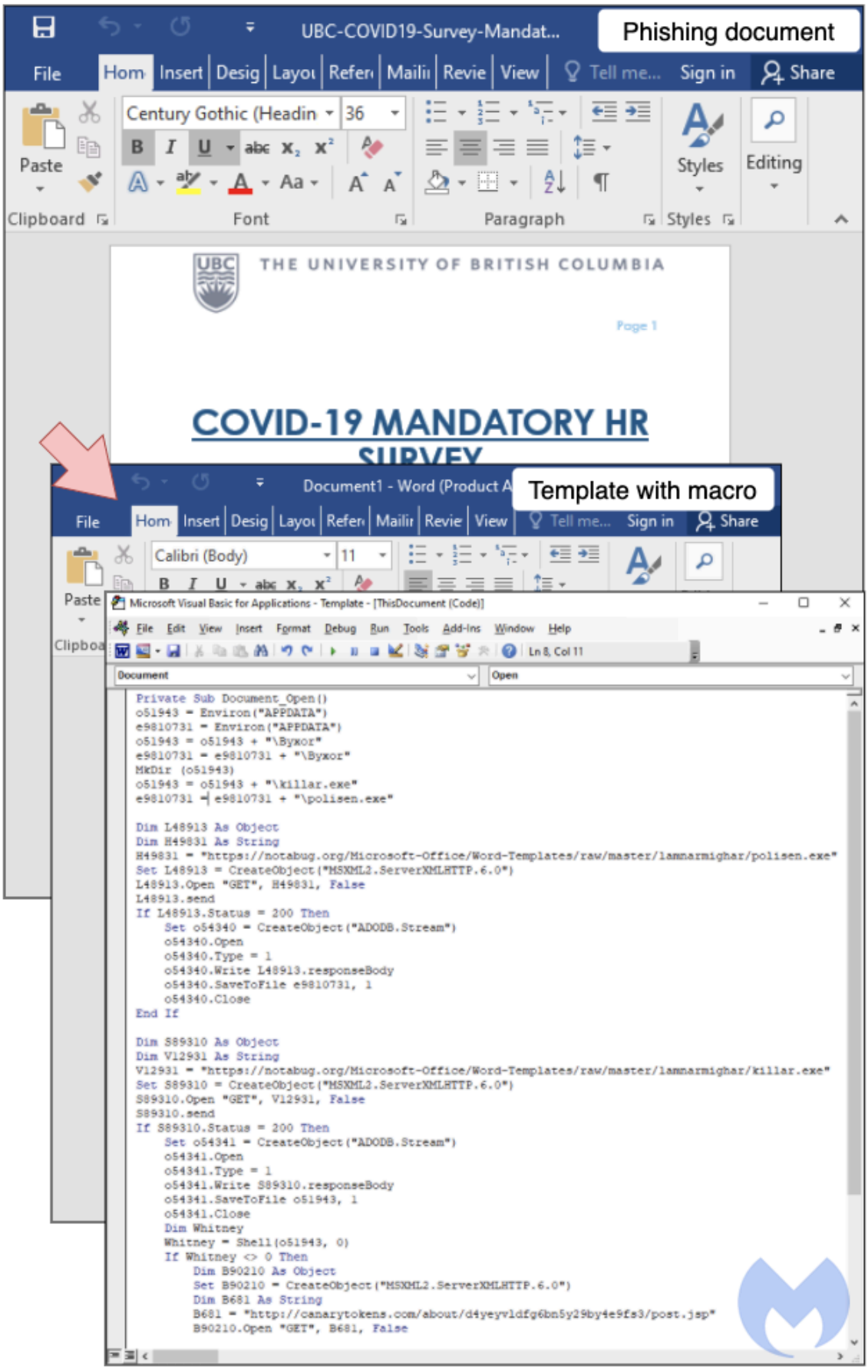
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

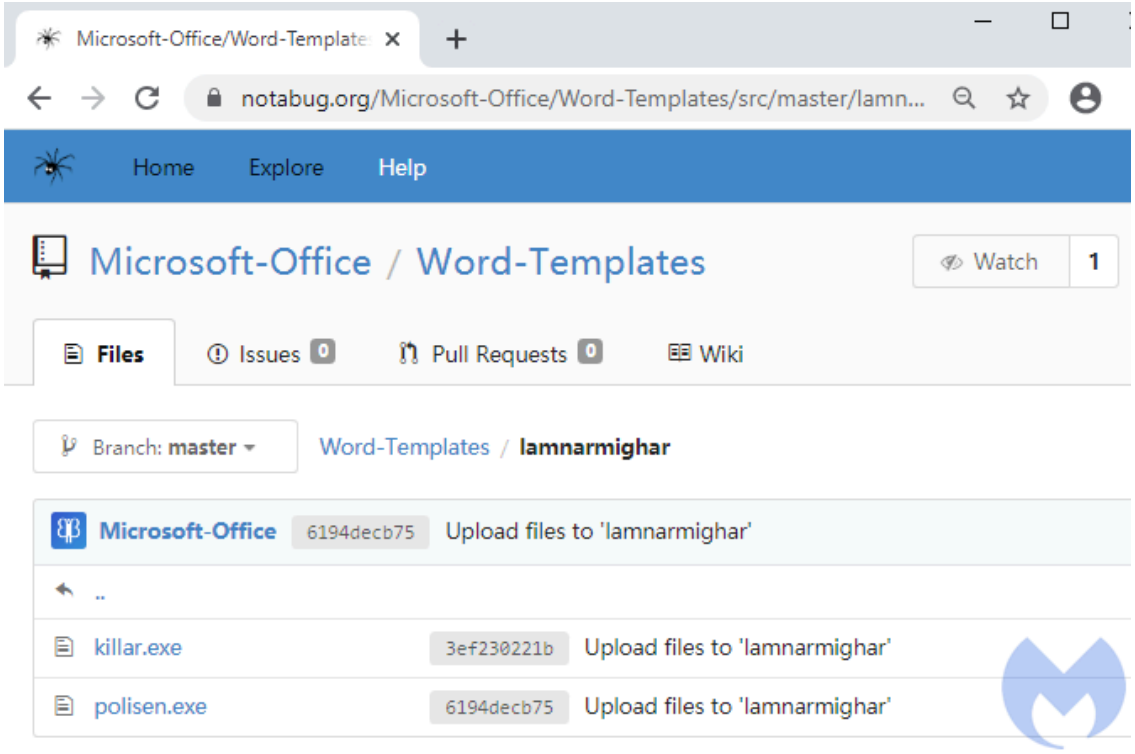
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



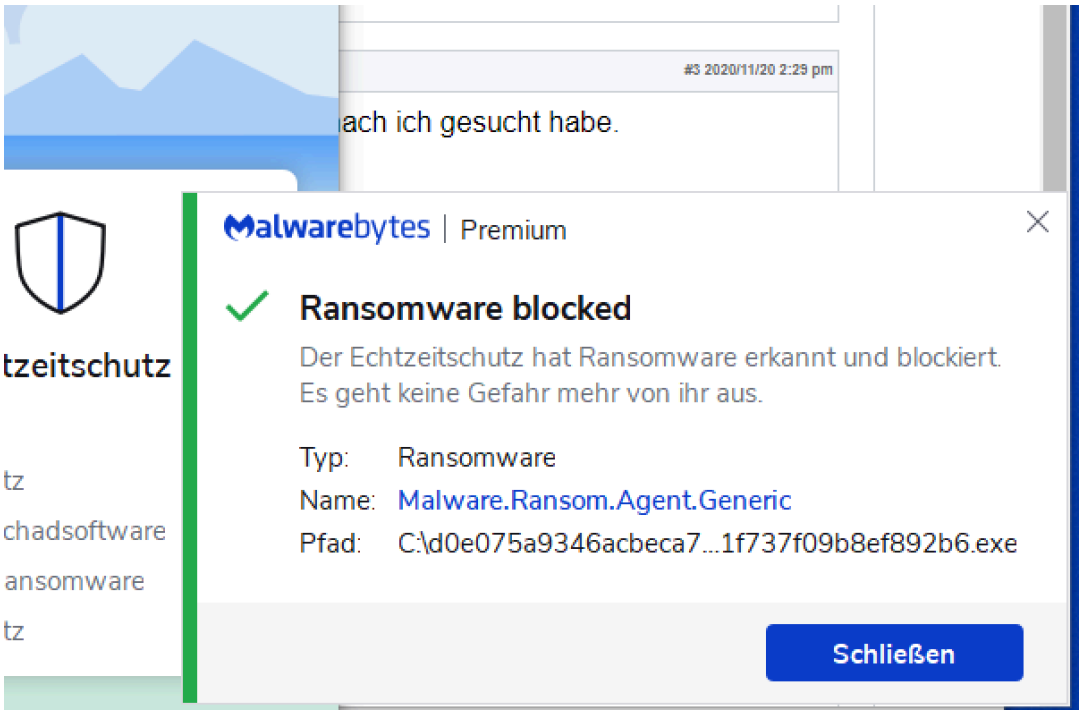
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

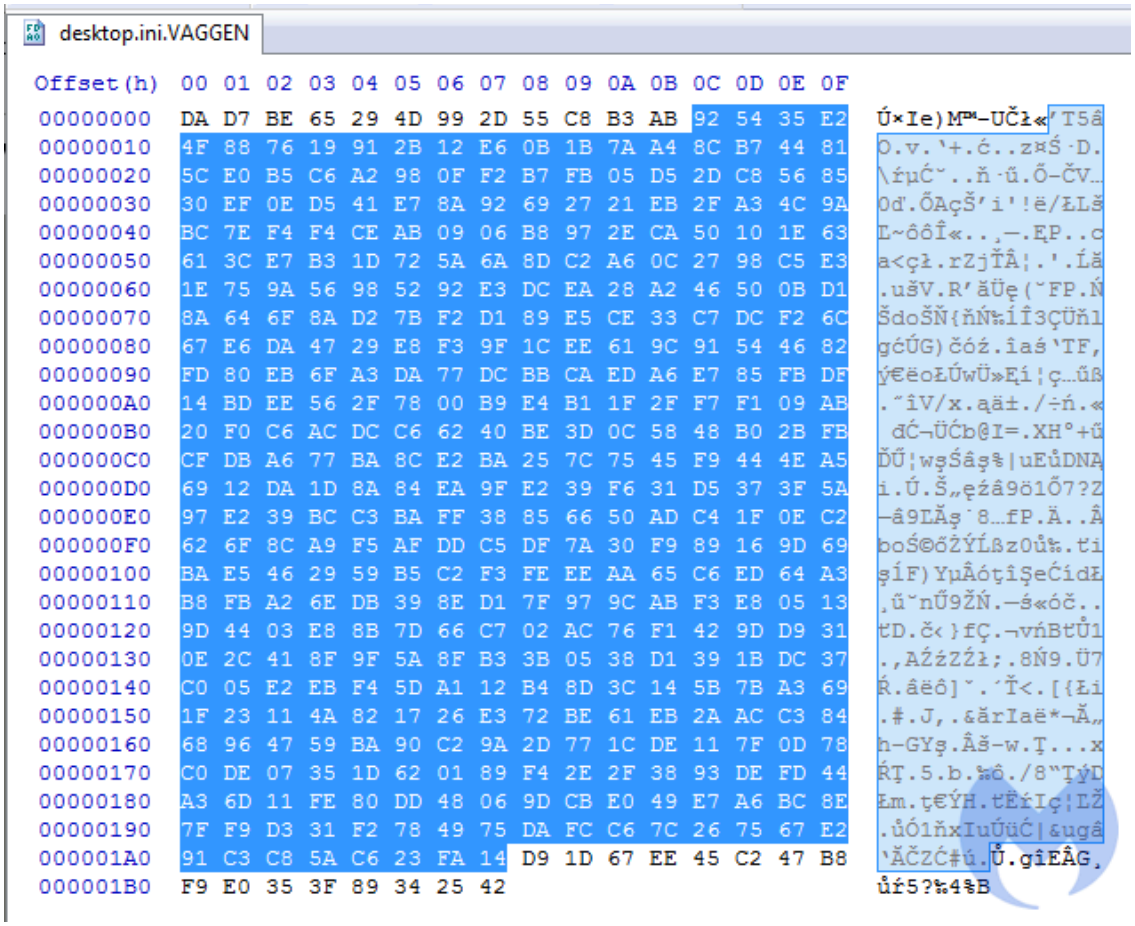
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

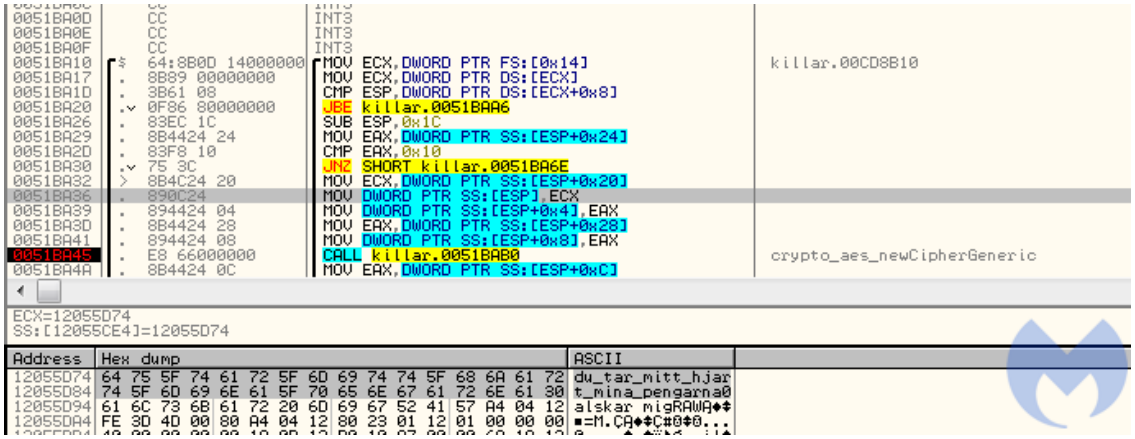


The content of the output file (with .VAGGEN extension) contains:

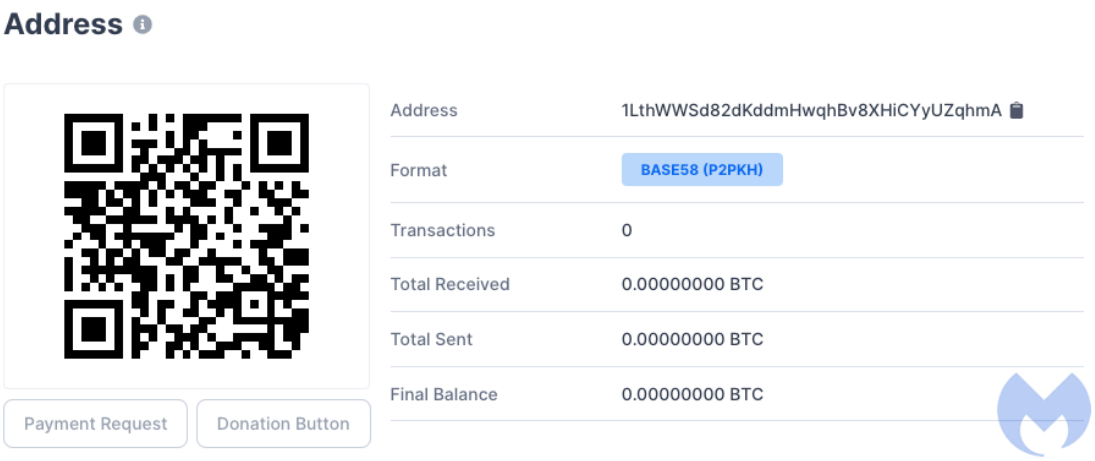
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

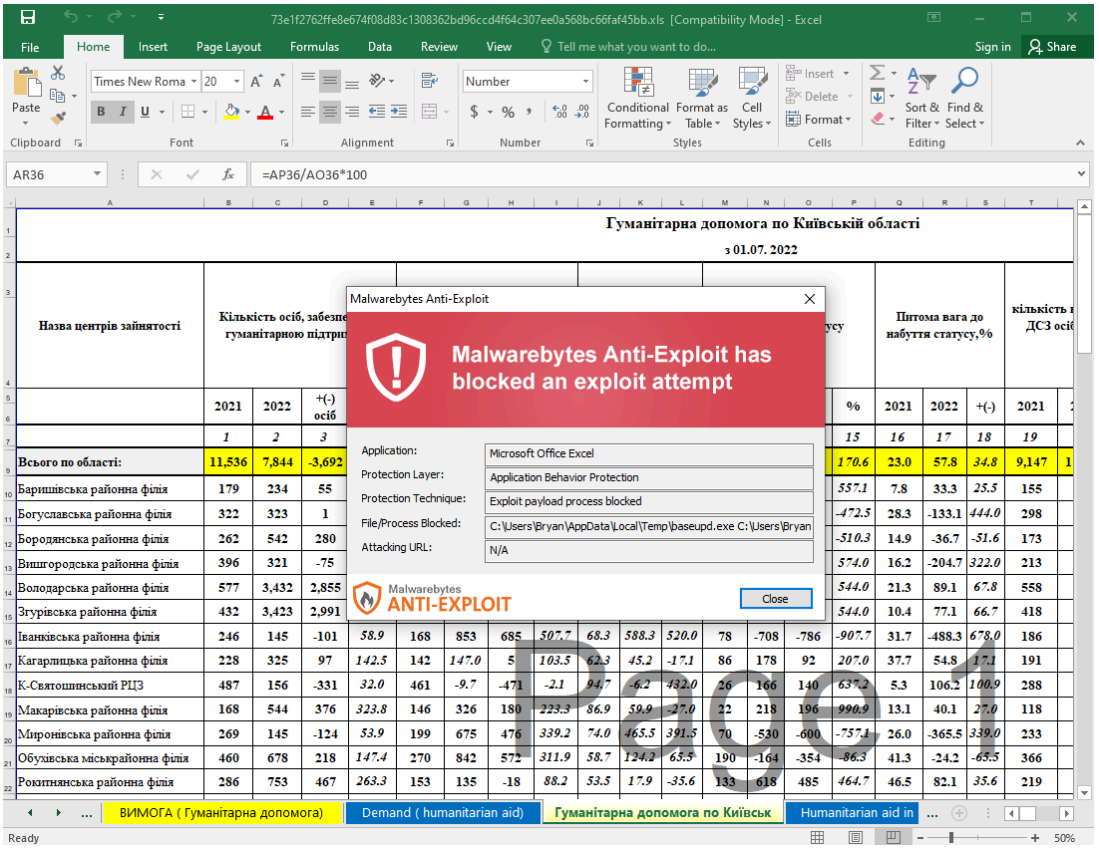
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

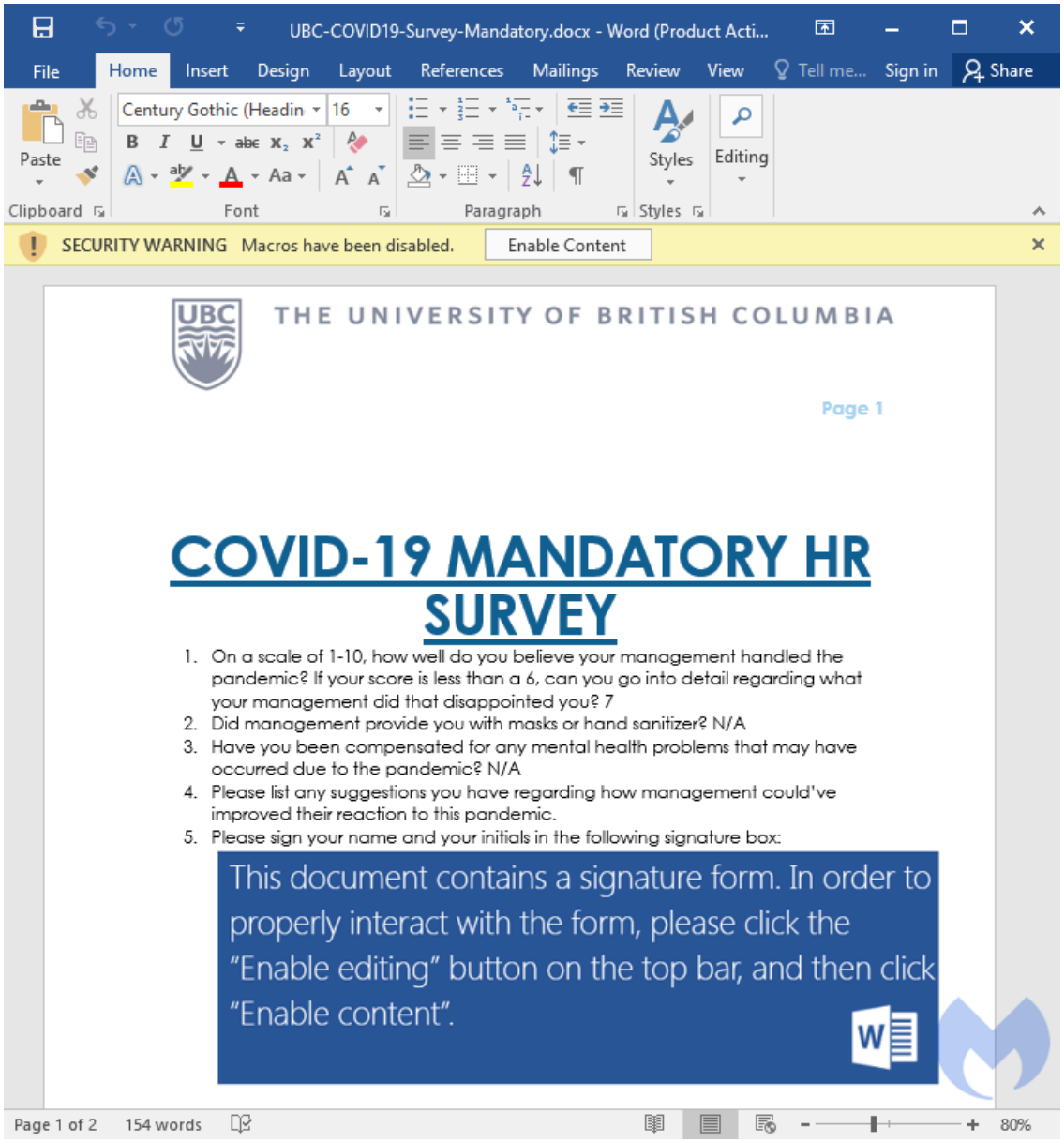
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

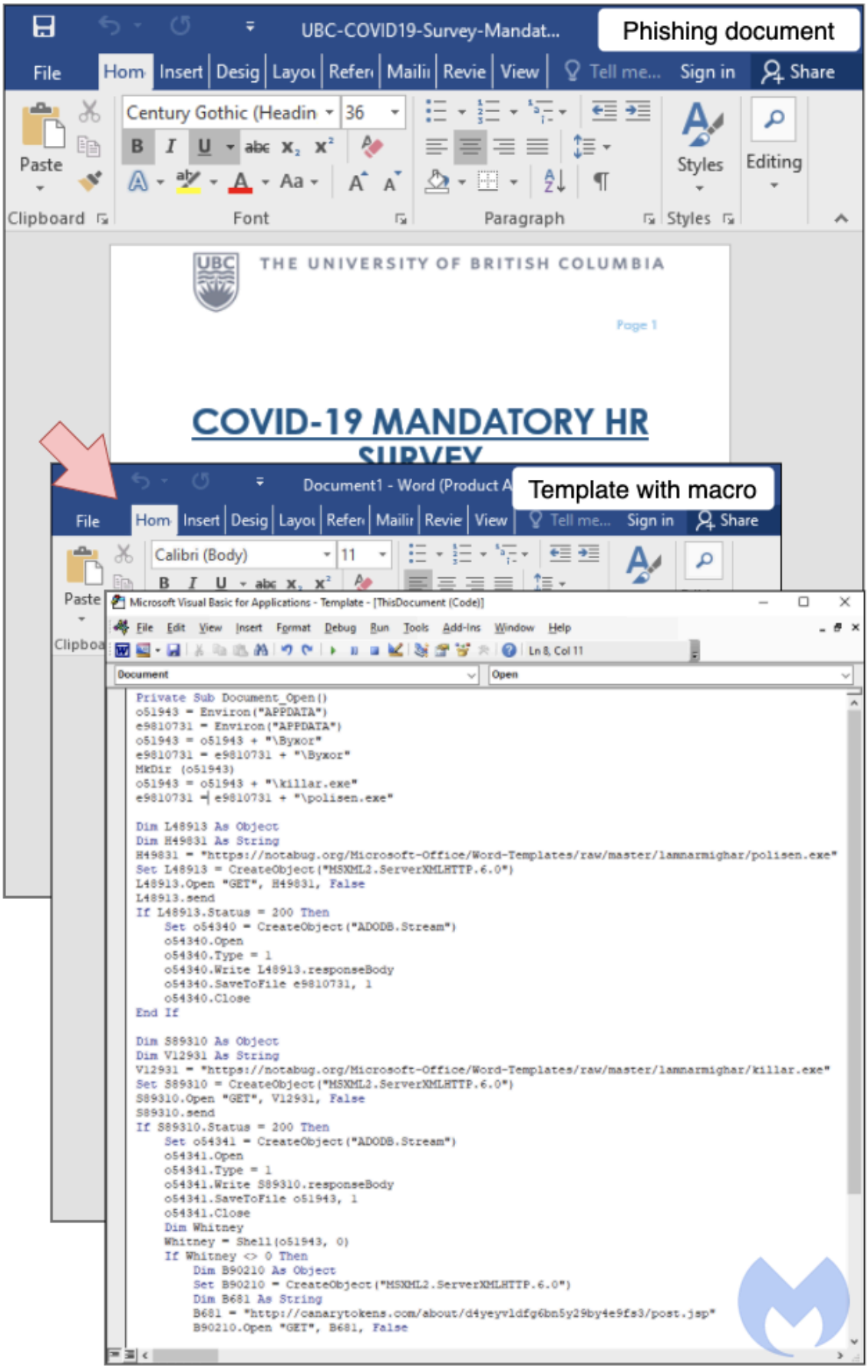
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

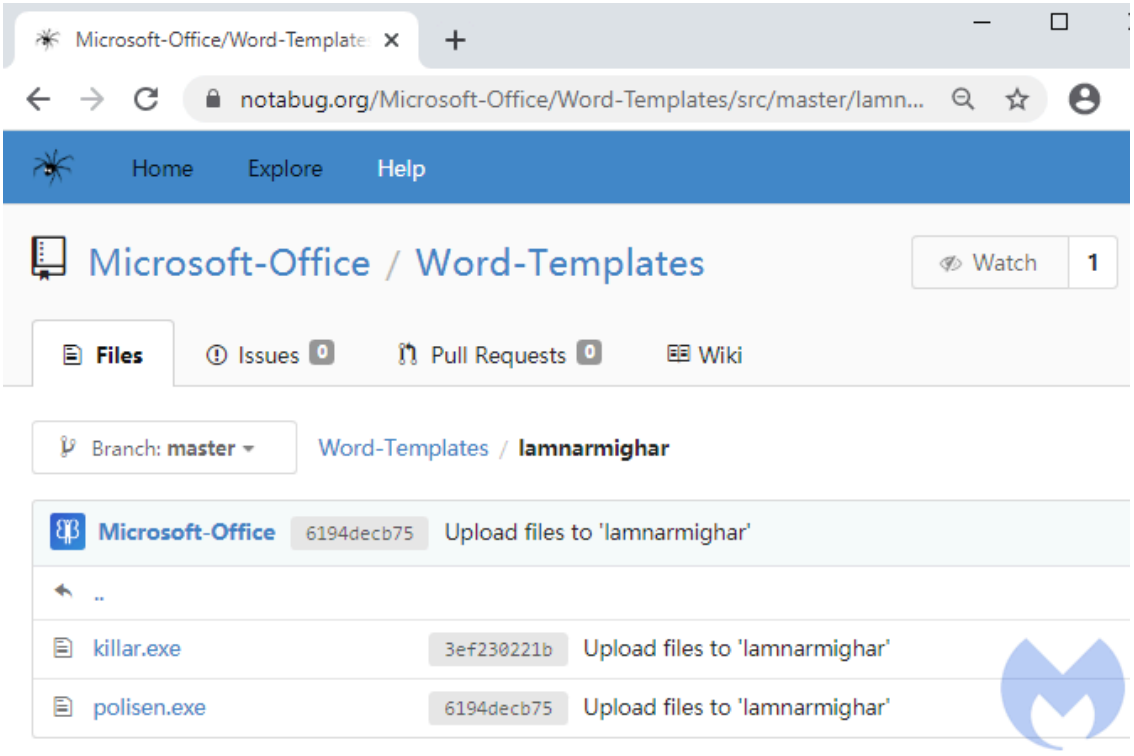
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



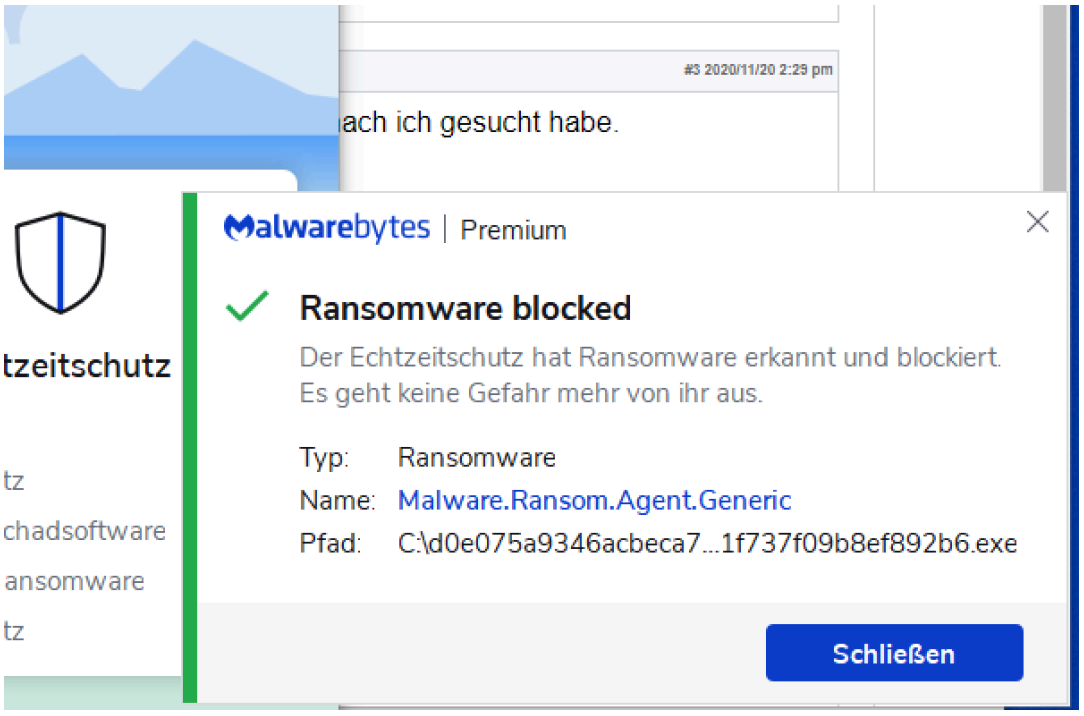
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

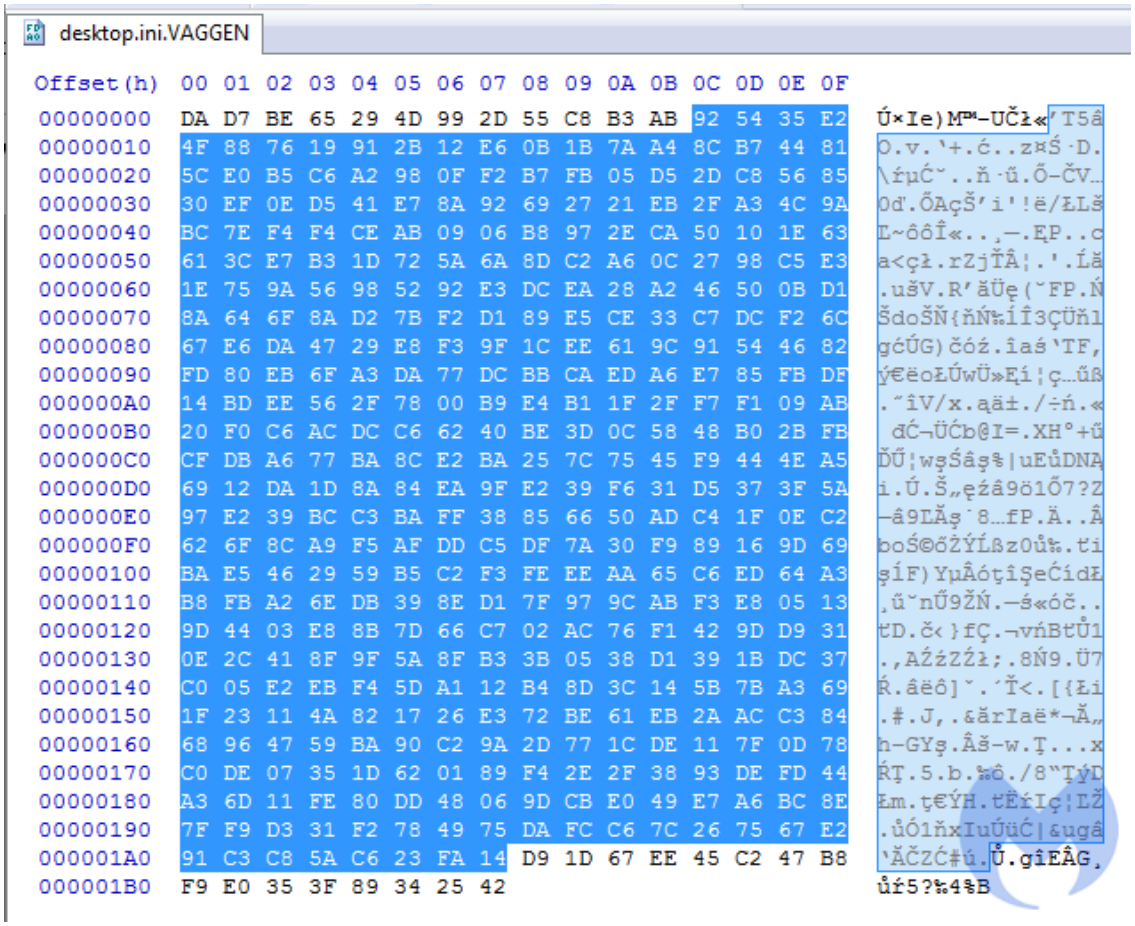
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

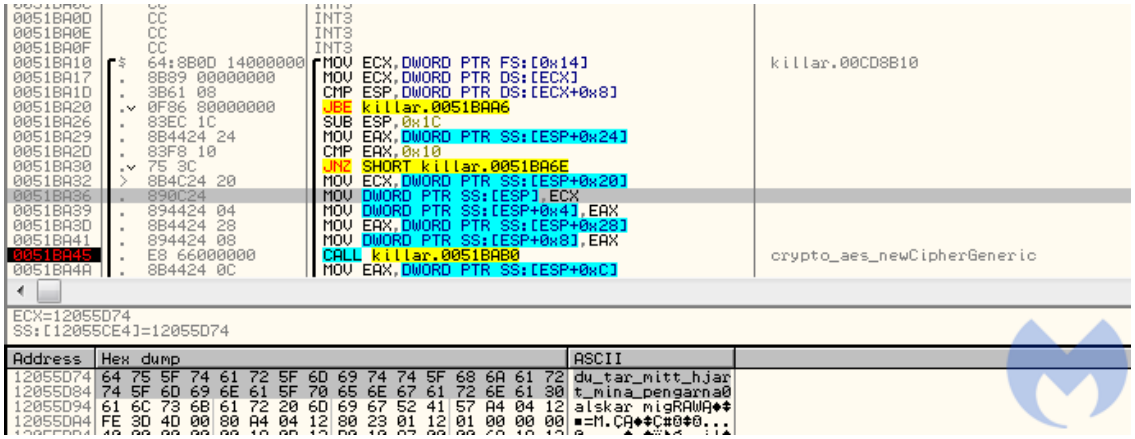


The content of the output file (with .VAGGEN extension) contains:

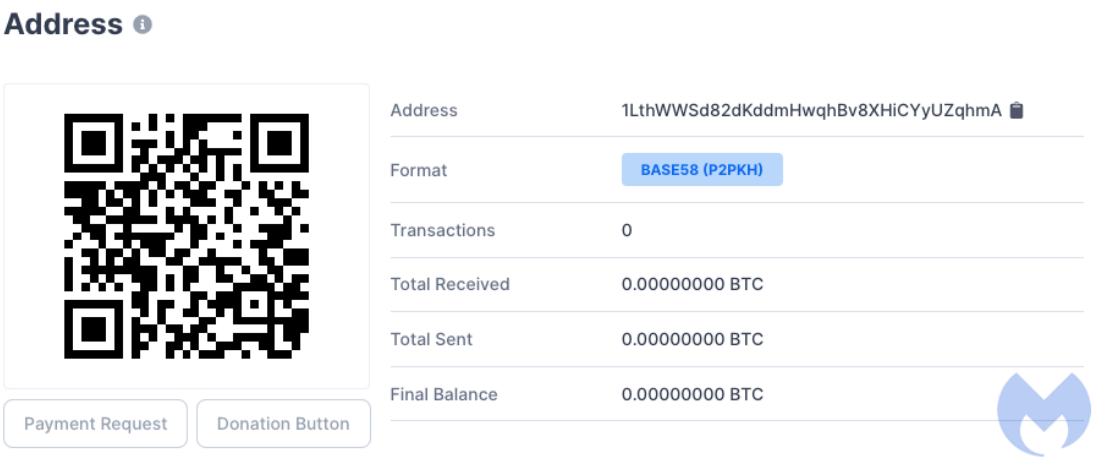
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

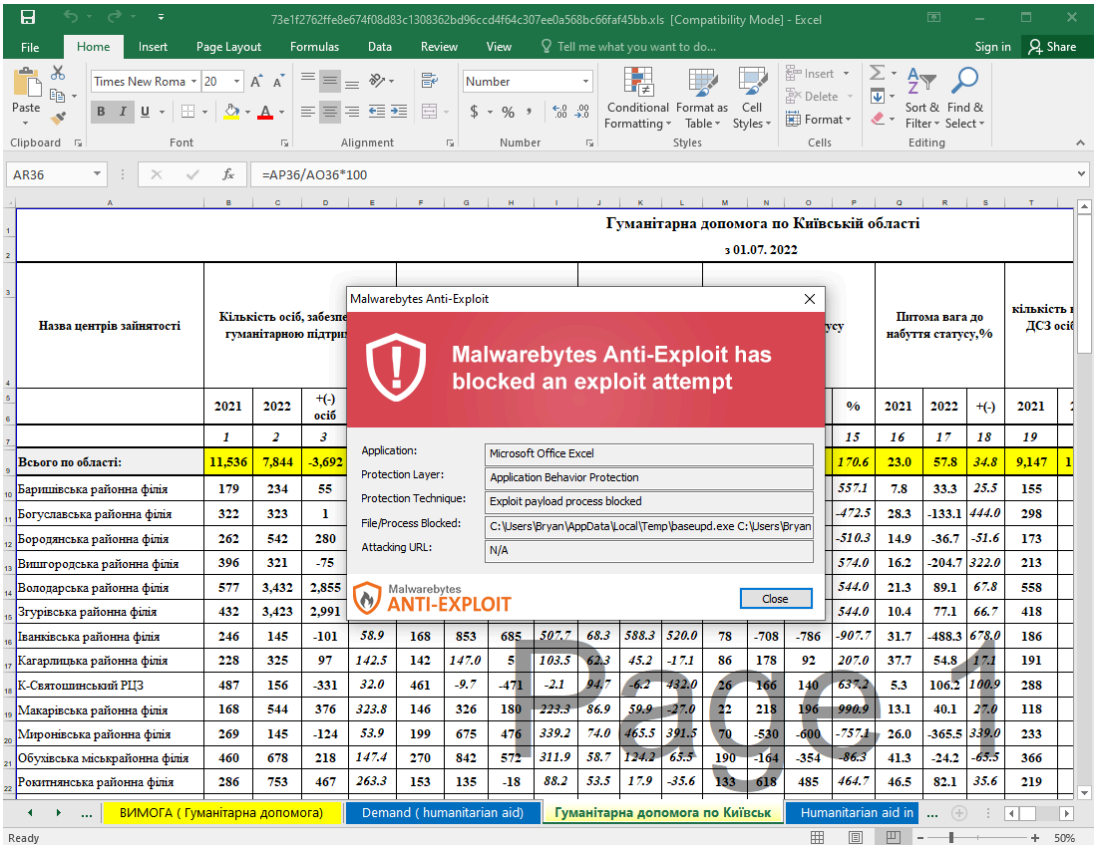
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

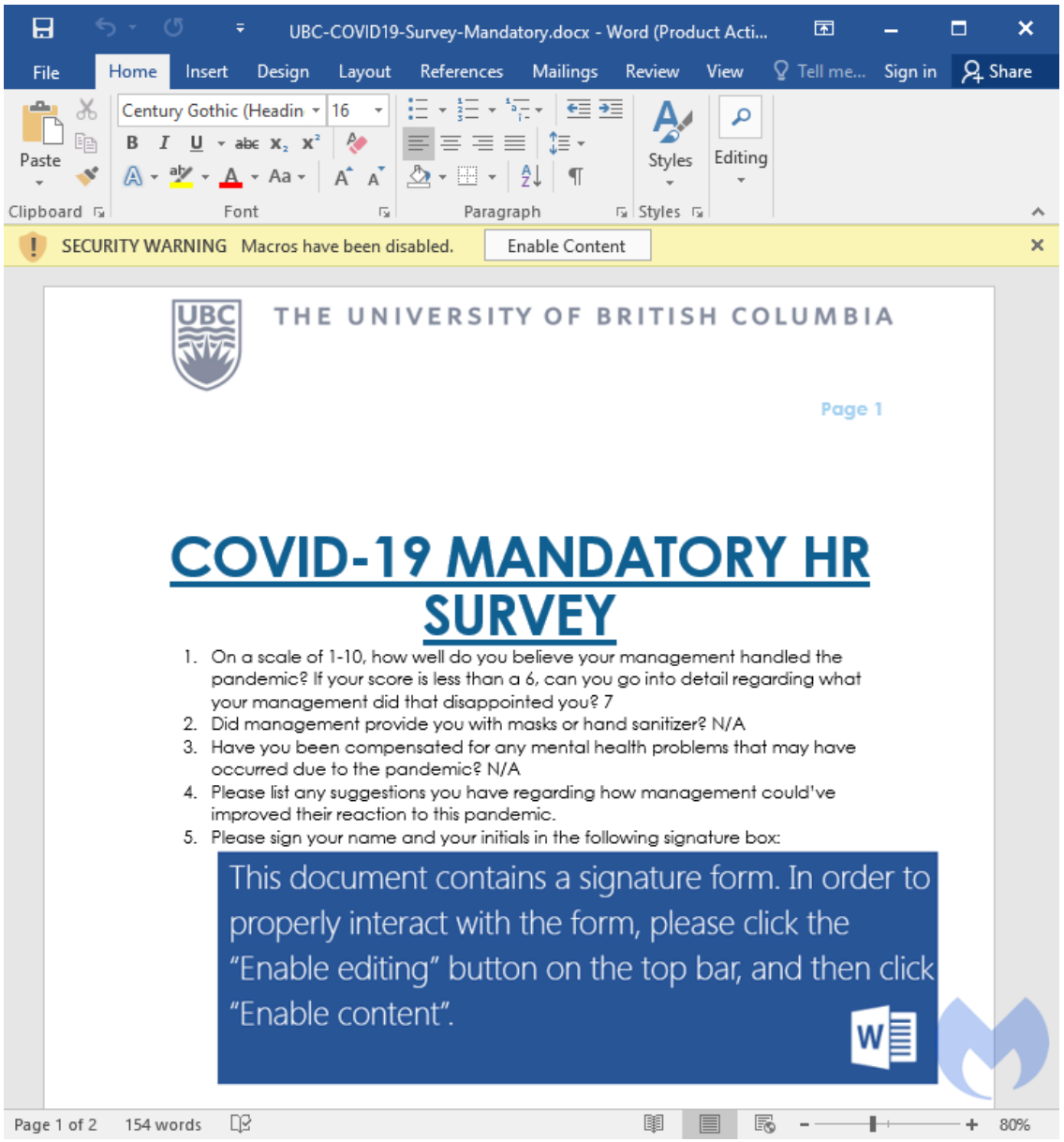
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

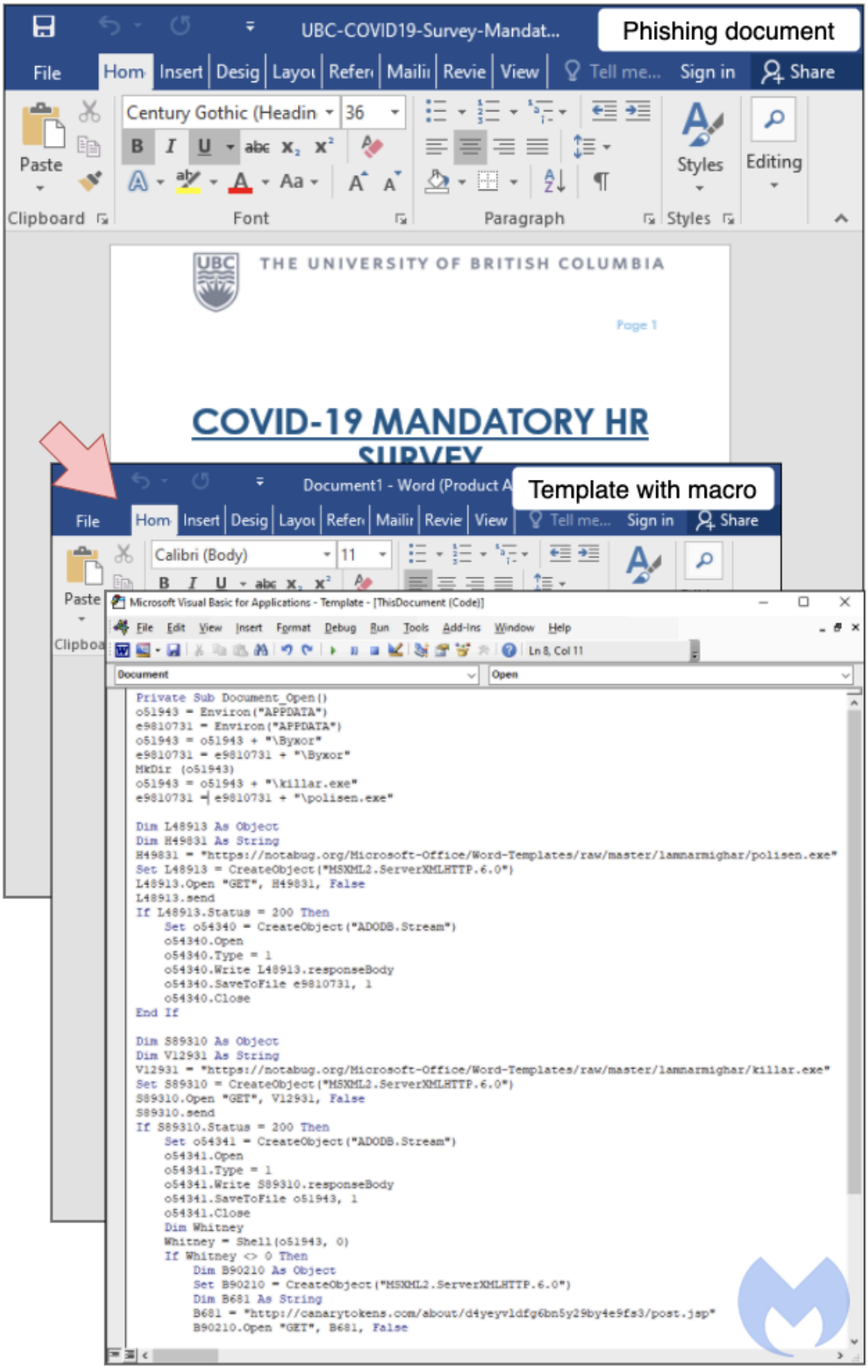
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

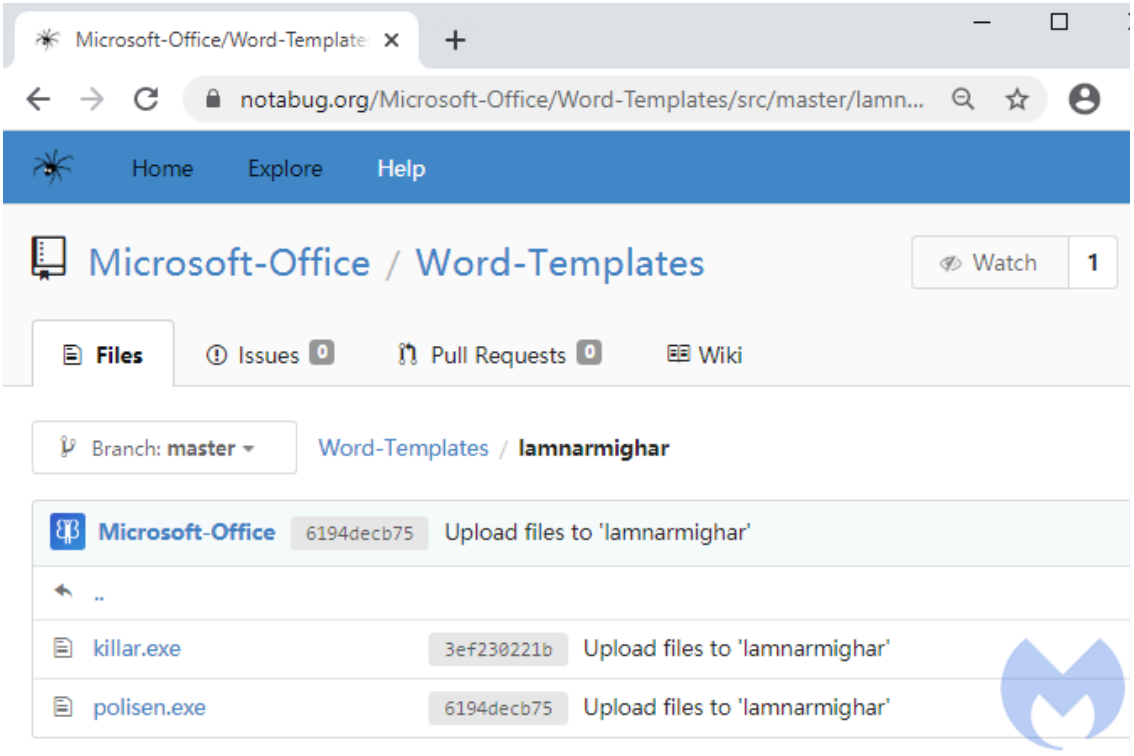
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



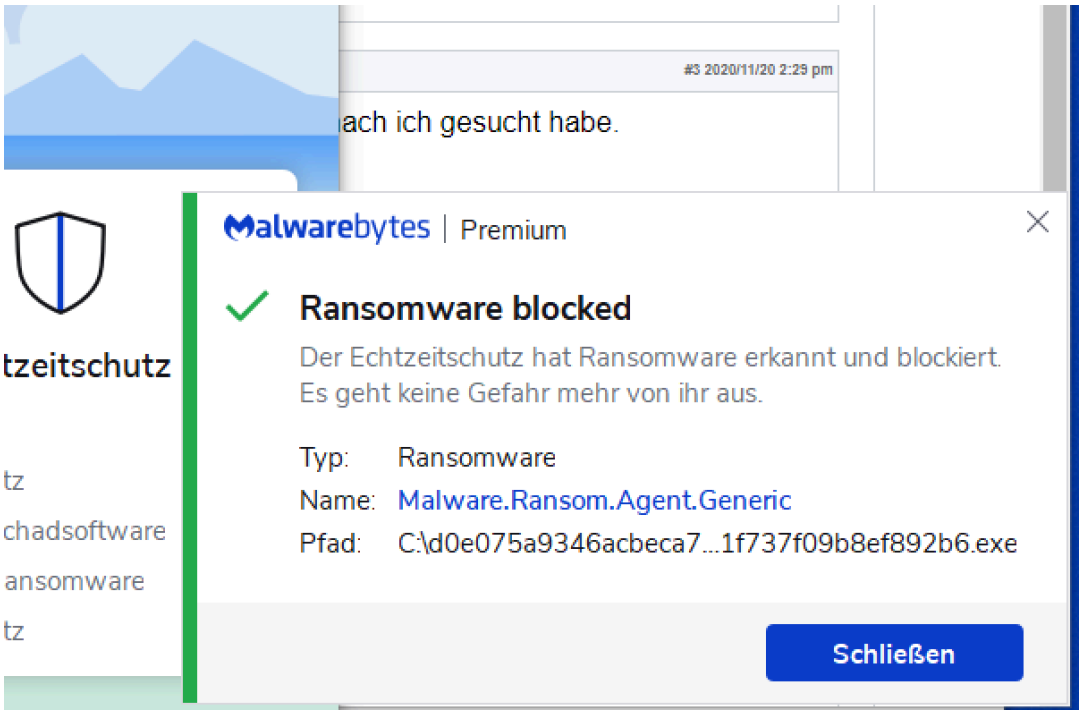
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

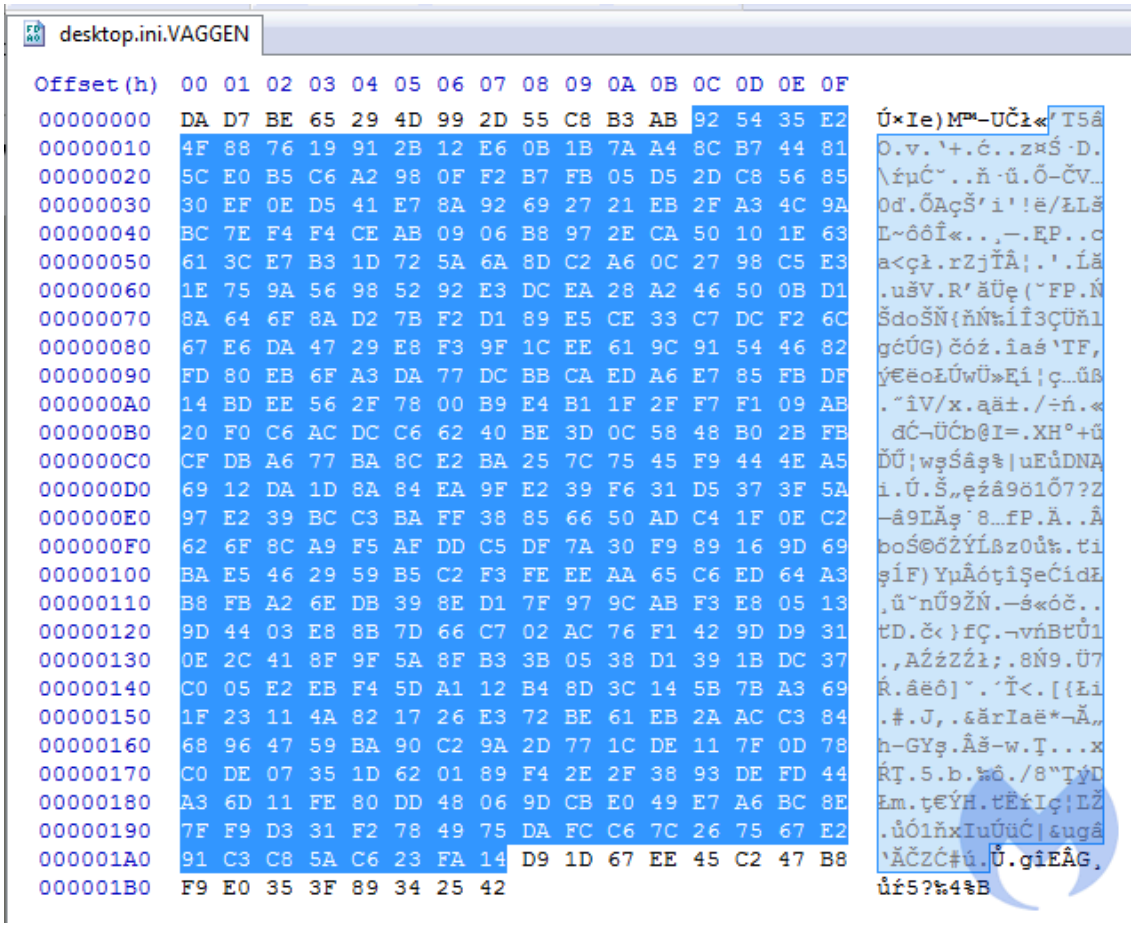
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



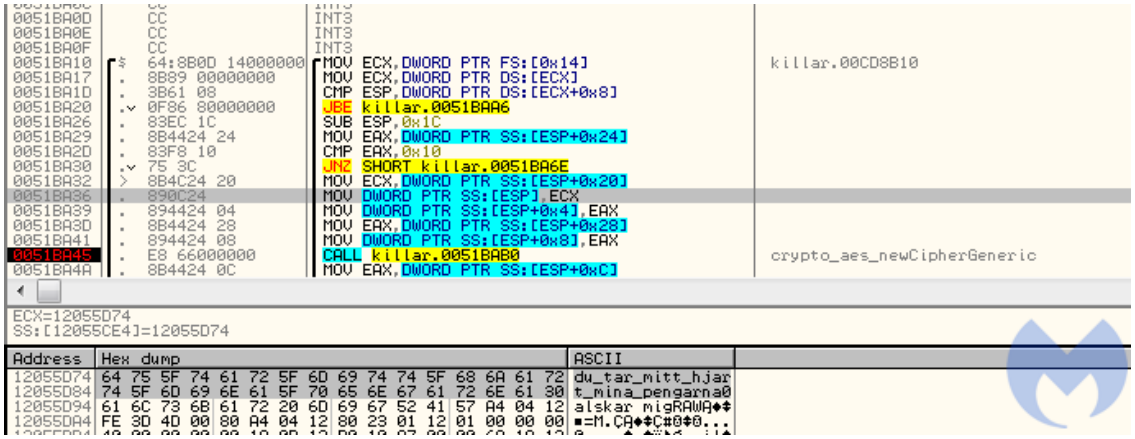
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

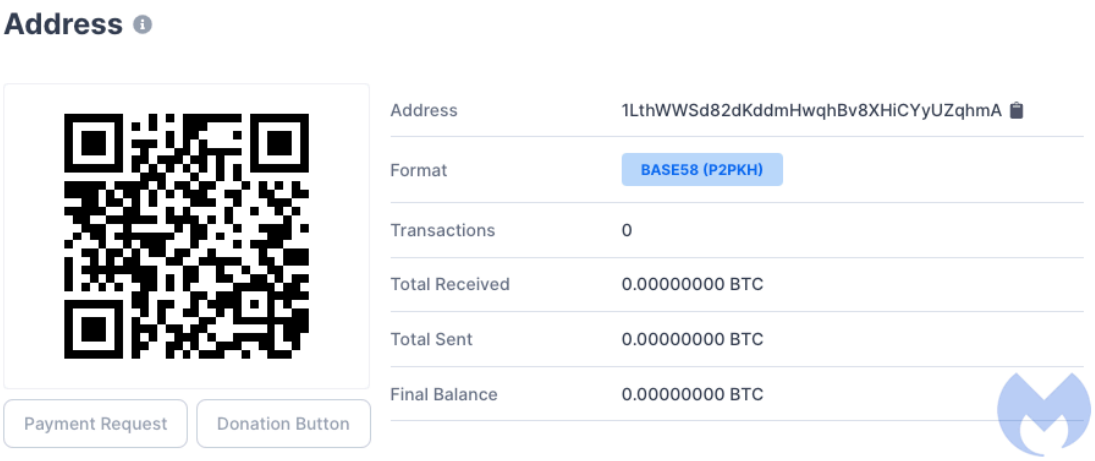
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

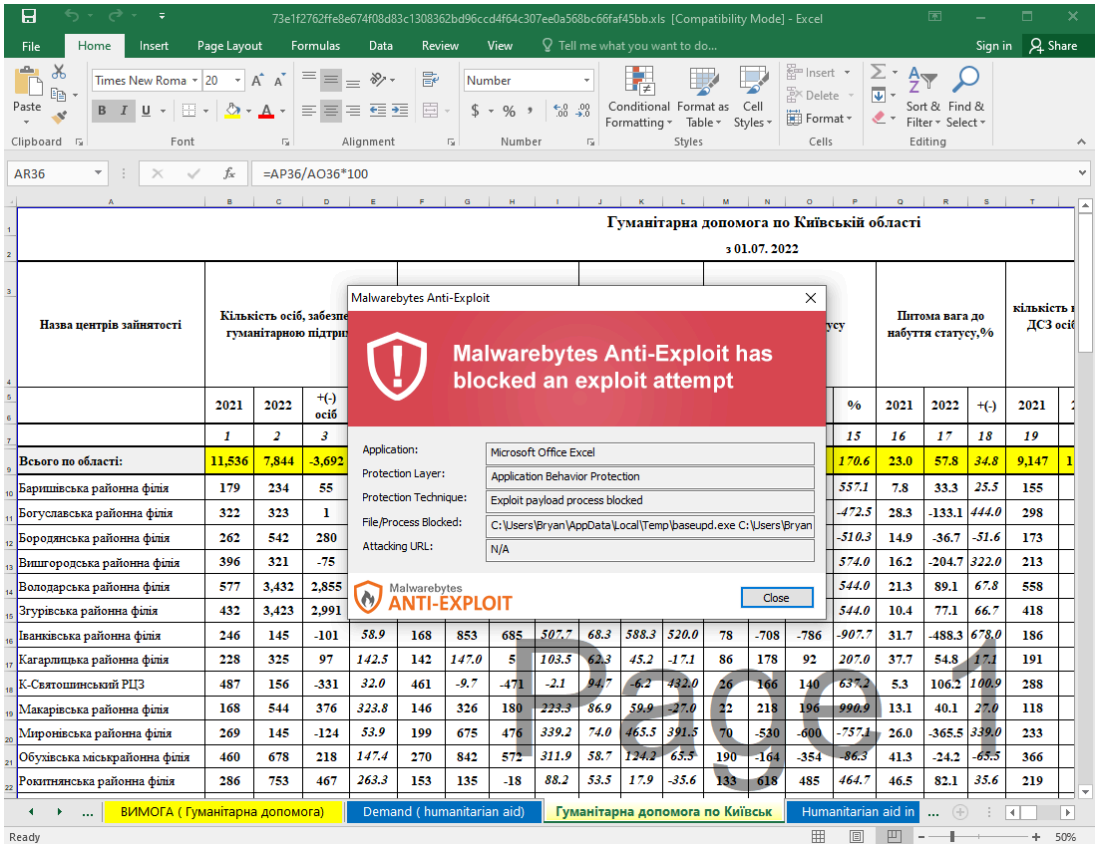
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

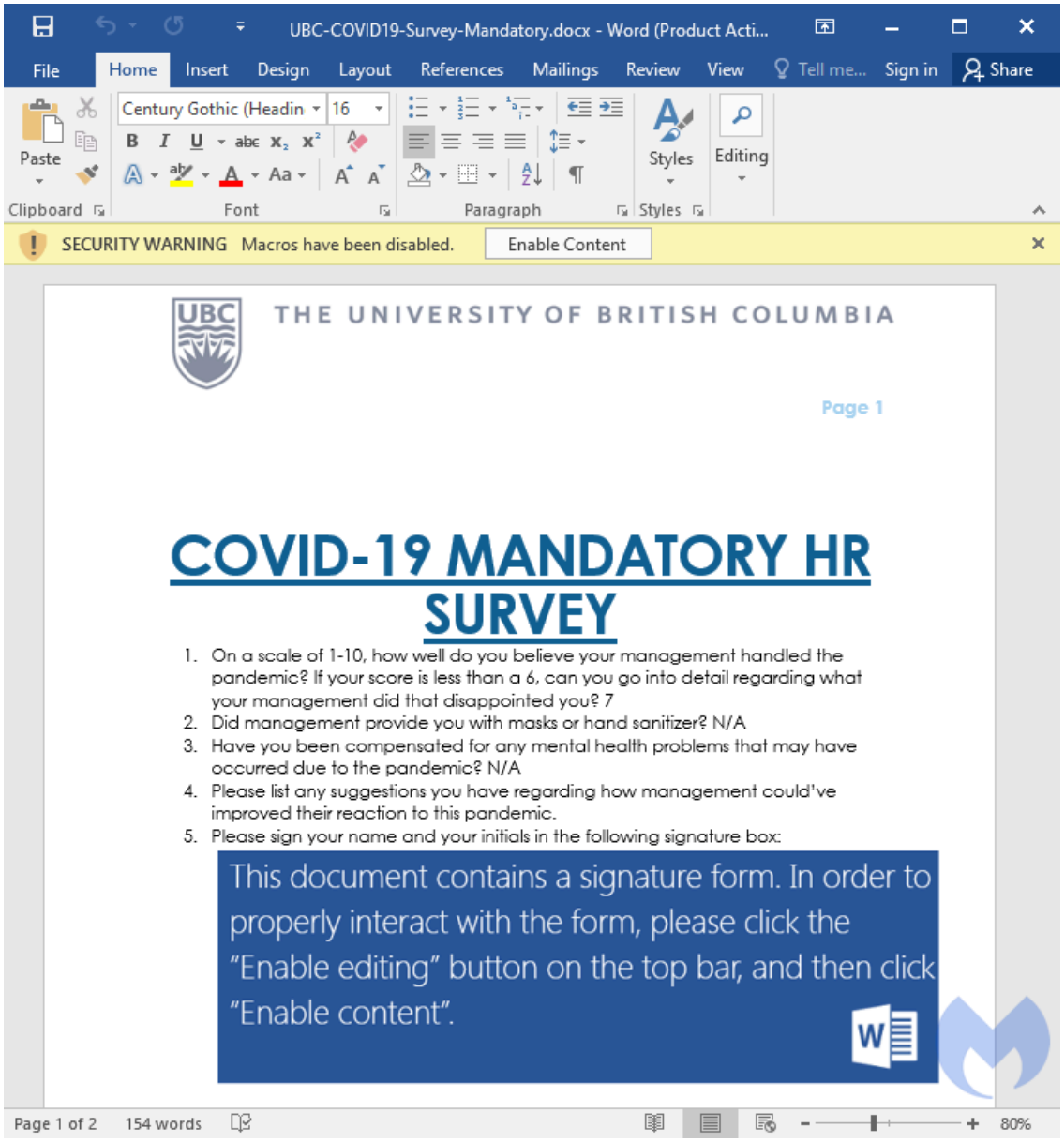
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

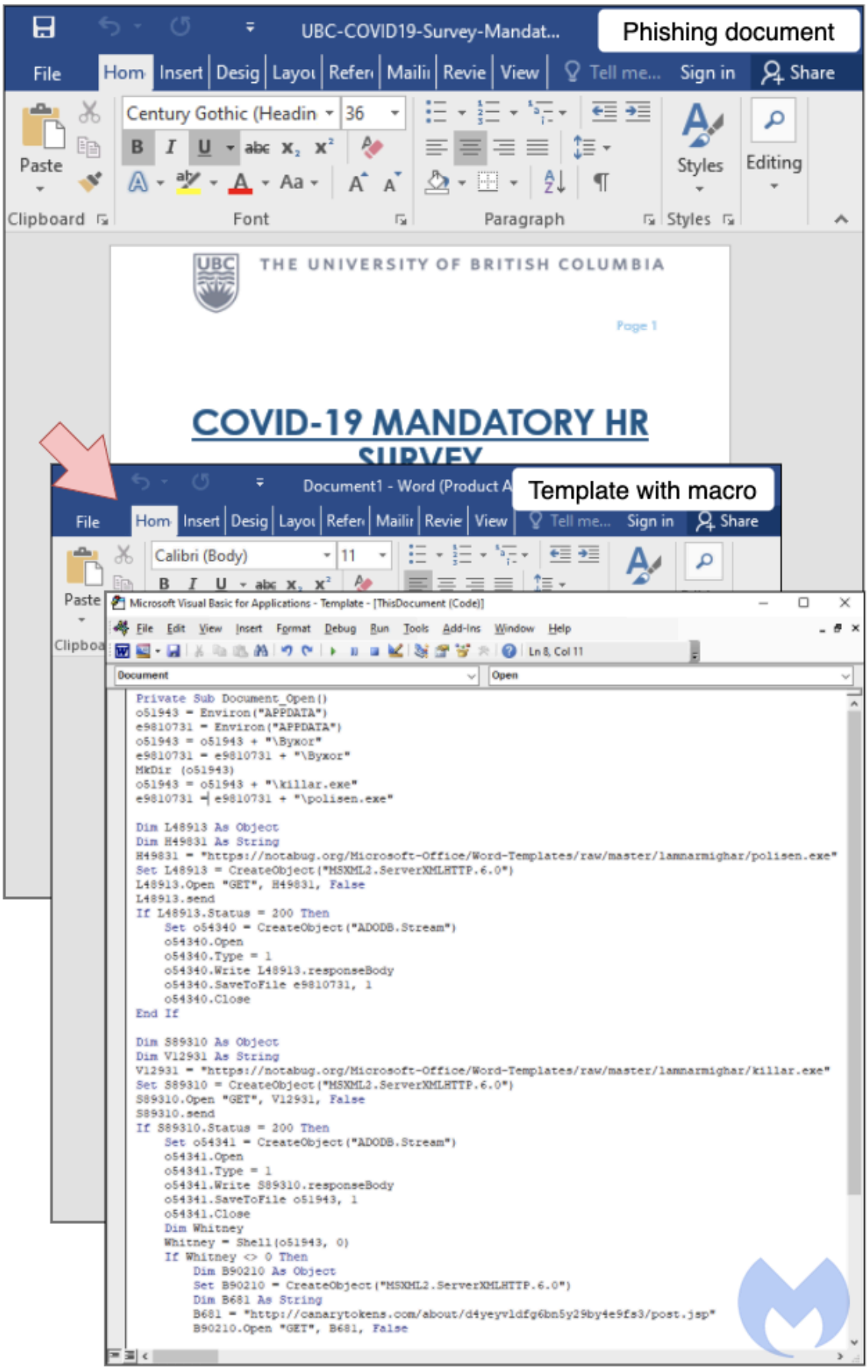
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

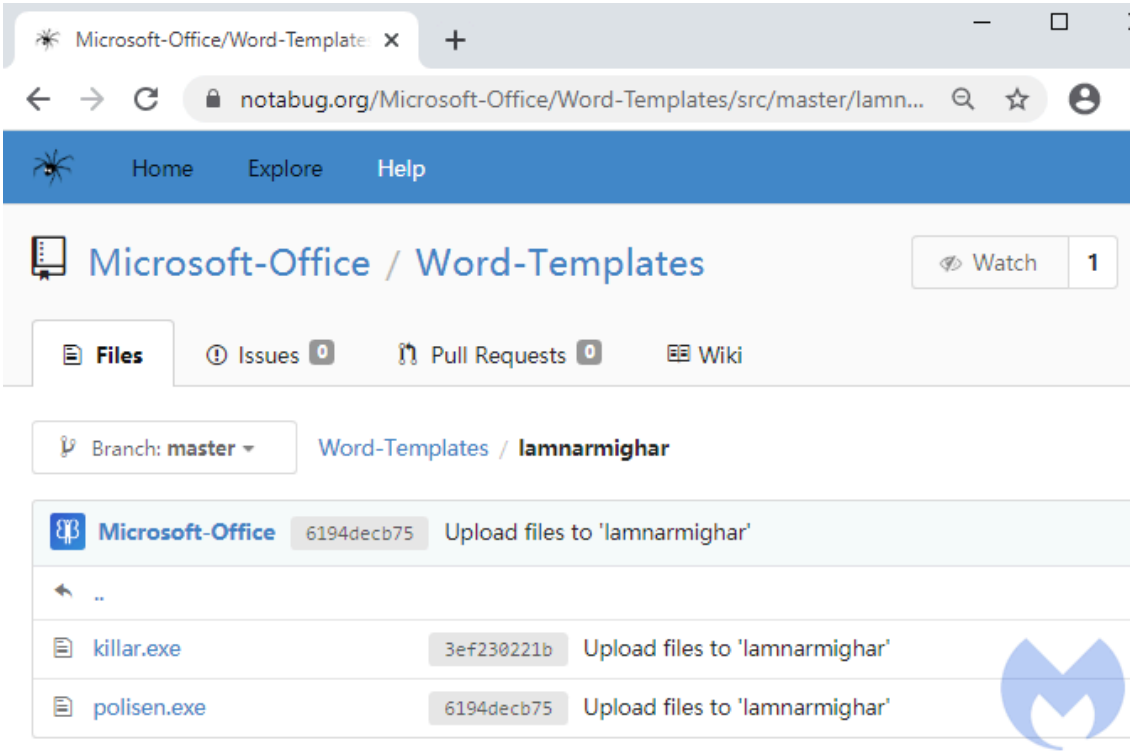
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



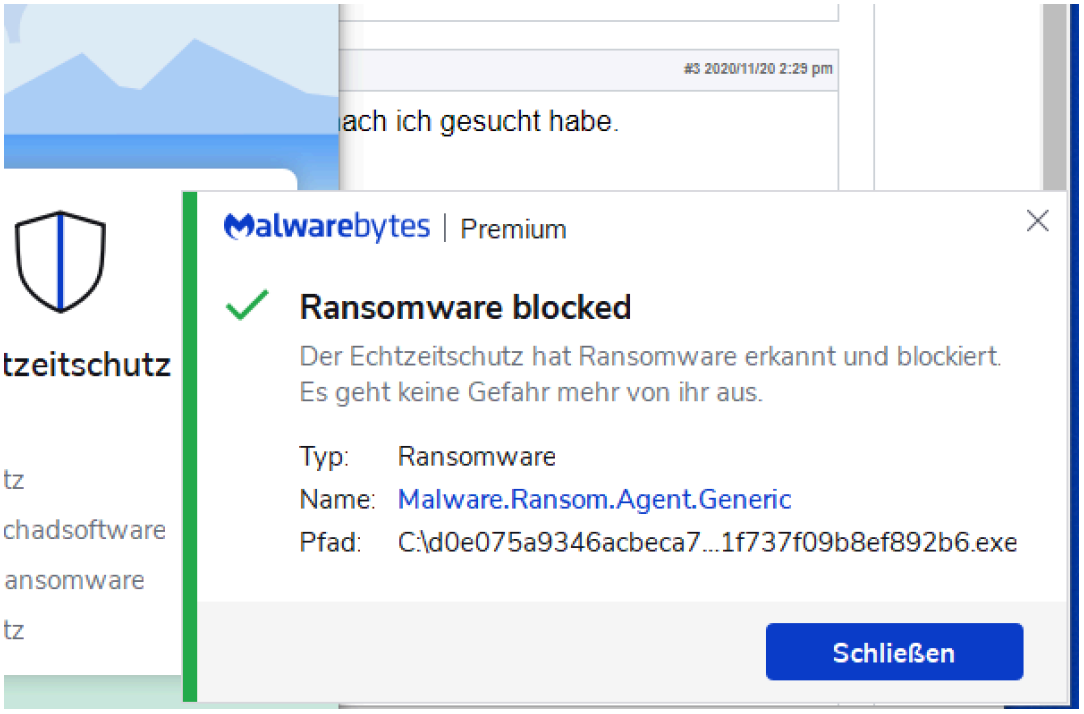
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```



Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

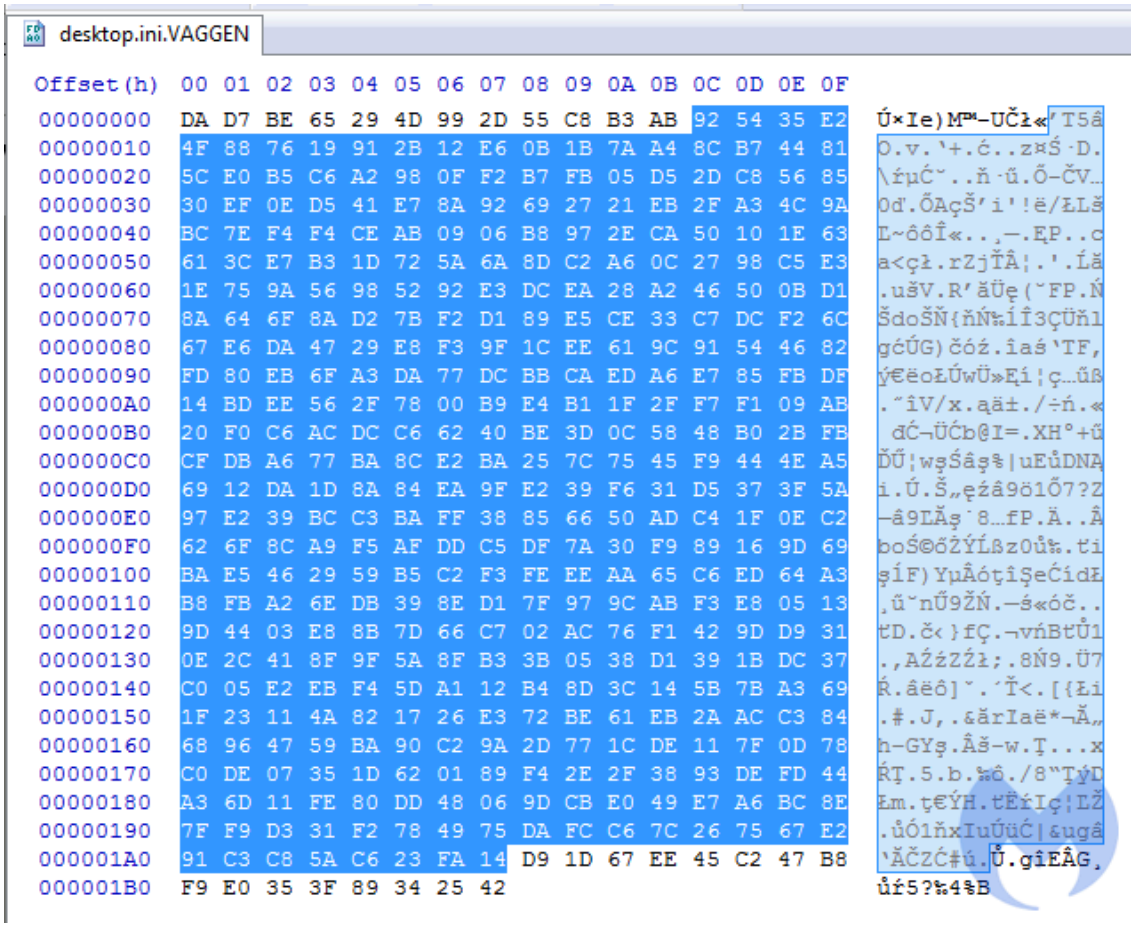


The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

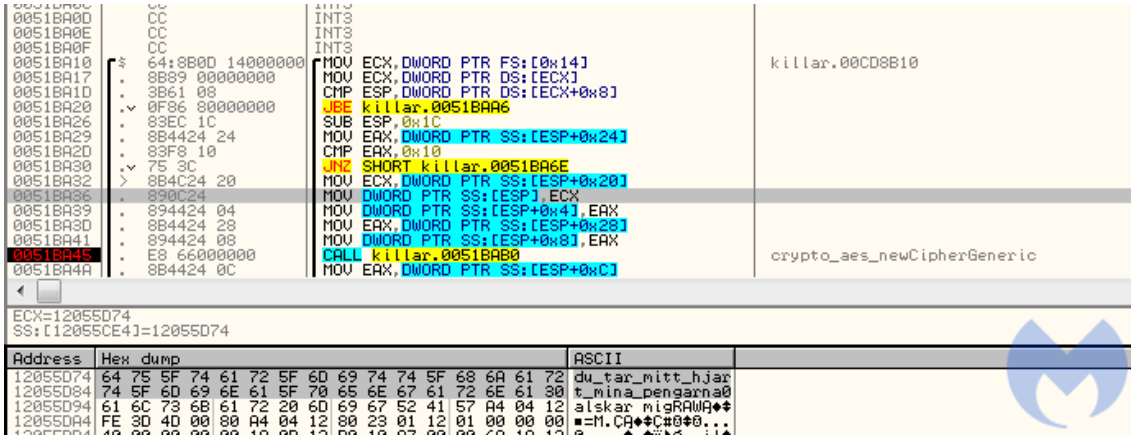


The content of the output file (with .VAGGEN extension) contains:

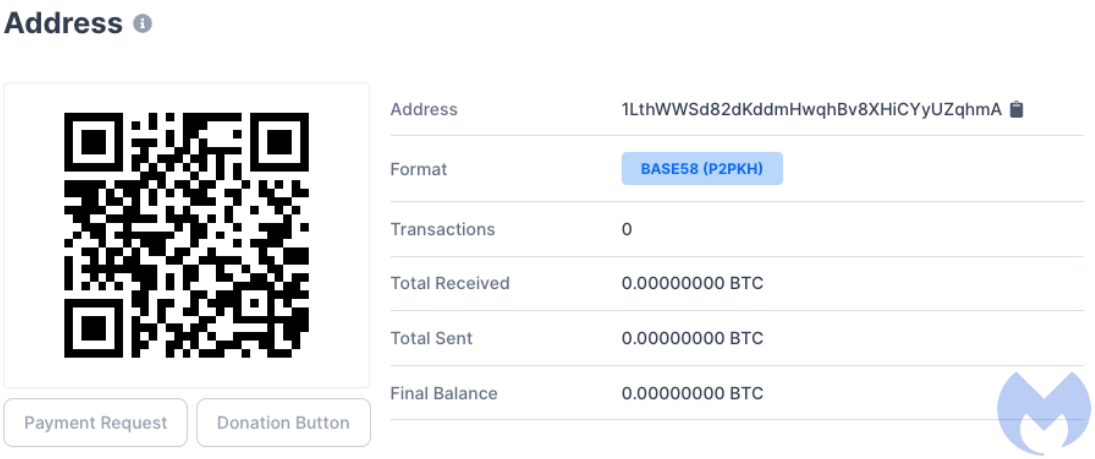
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

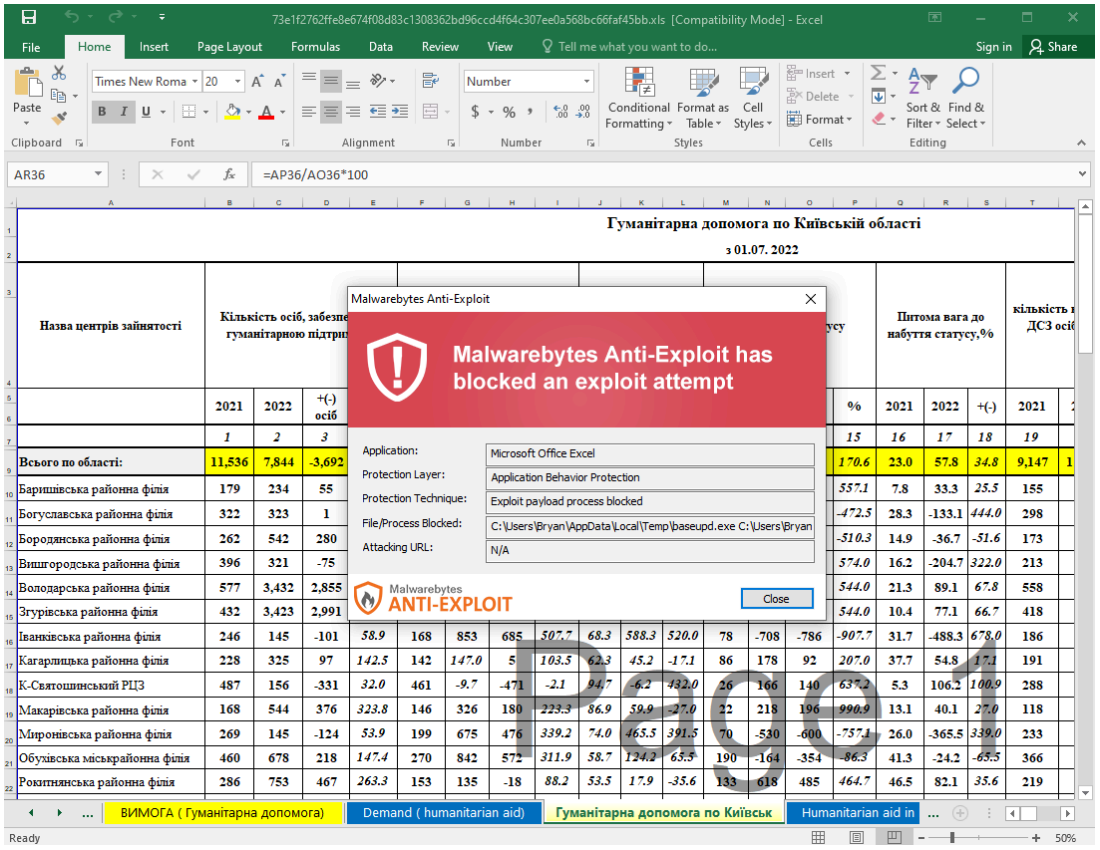
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

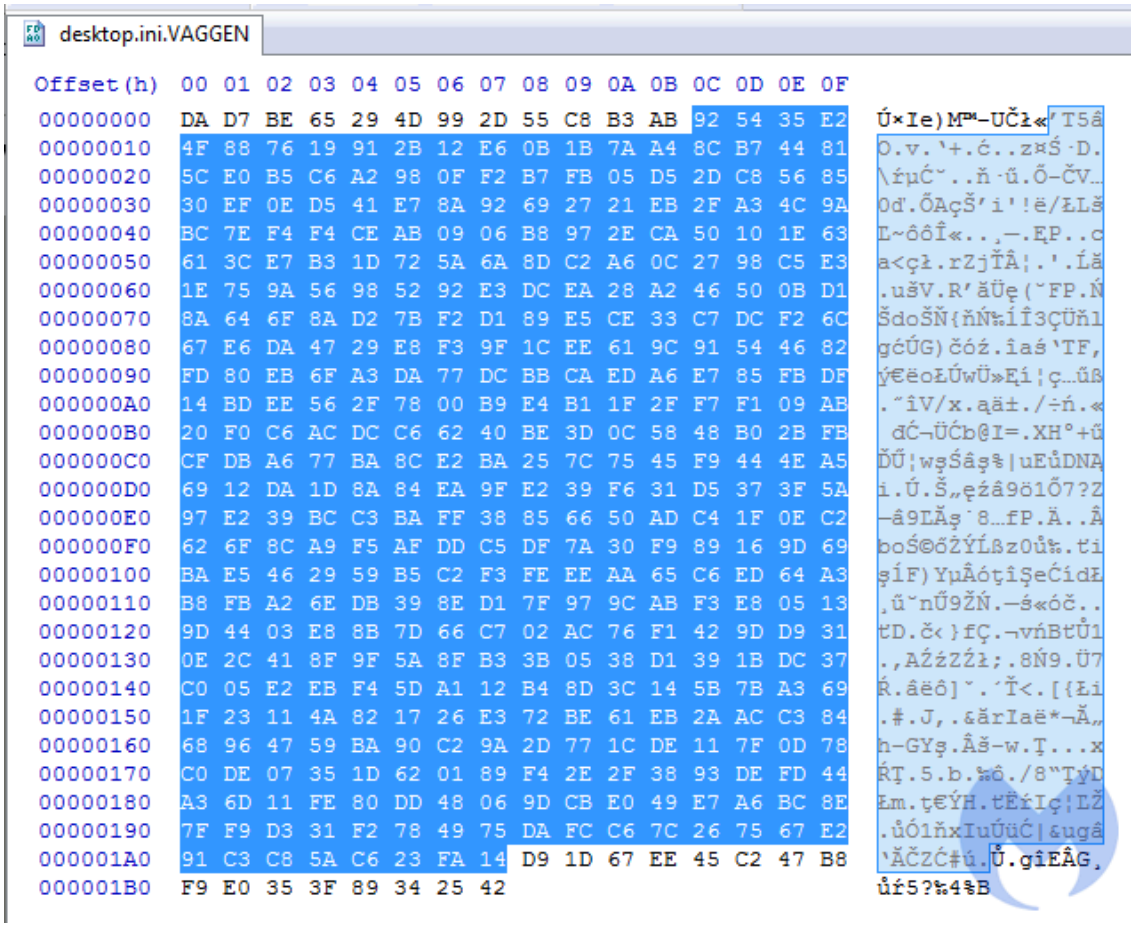
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

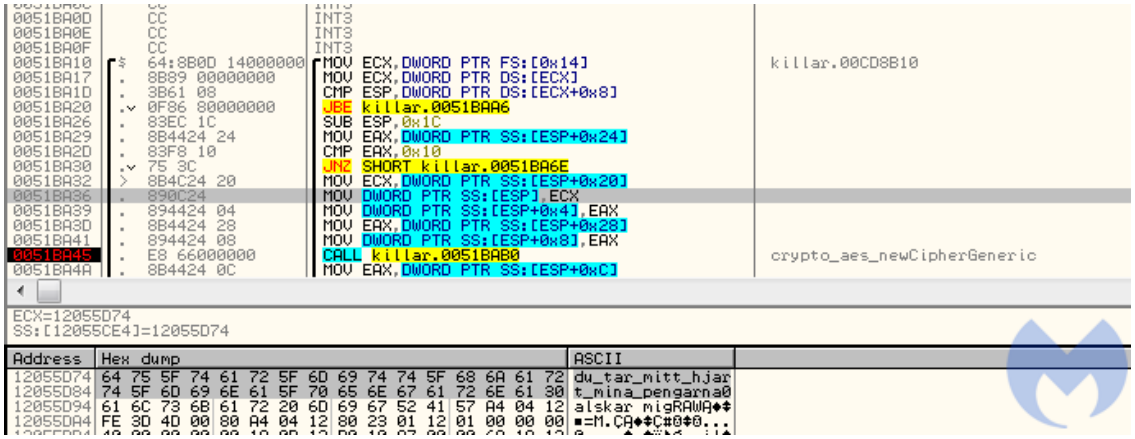
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

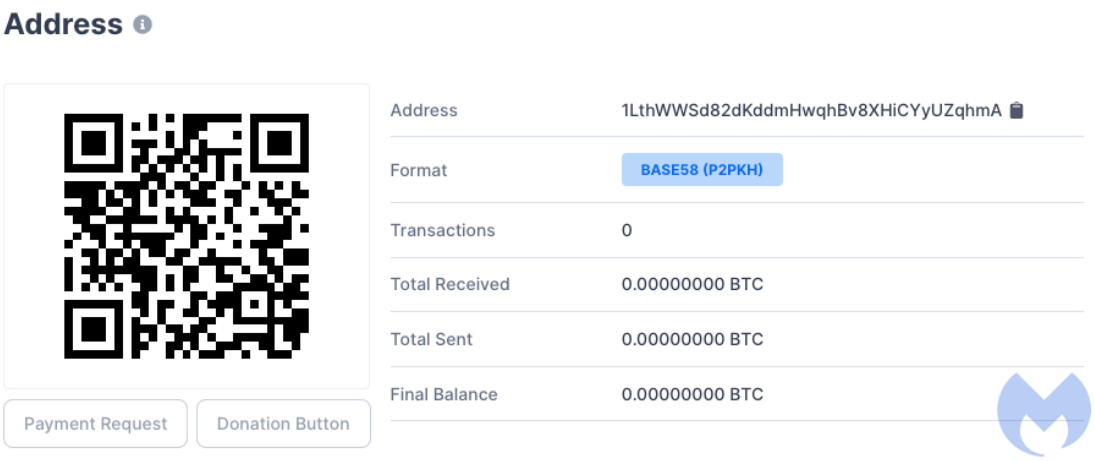
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

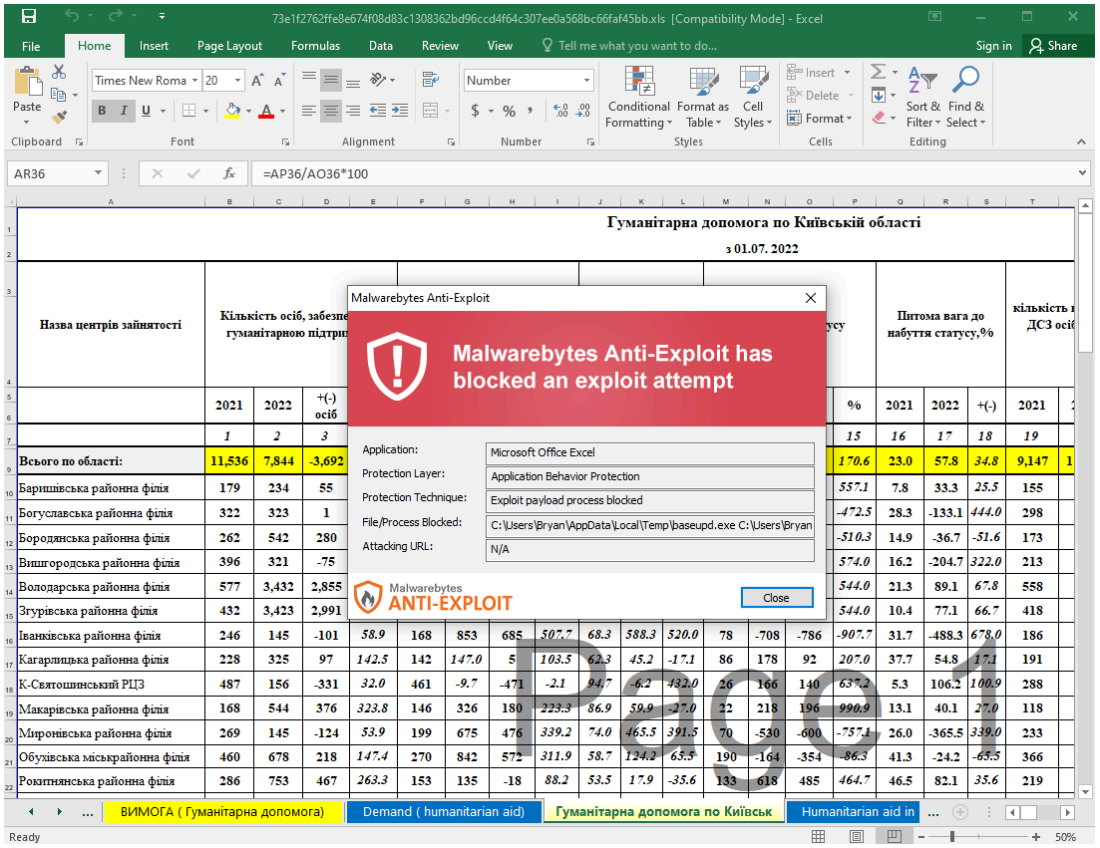
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
46  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
47    runtime_morestack_noctxt();
48  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
49  ((void (*)(void))loc_45CDA0)();
50  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
51  v38 = v4;
52  v30 = v5;
53  v31 = v6;
54  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
55  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
56  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
57  v32 = v10;
58  v11 = *(void (__cdecl **)(int))(v10 + 16);
59  v39 = v12;
60  v11(v12);
61  v29 = v13;
62  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
63  v40 = v15;
64  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
65  v17 = v27;
66  if ( v27 )
67  {
68    if ( v27 )
69      v17 = *(_DWORD *)(v27 + 4);
70    v43 = v17;
71    v44 = v28;
72    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
73  }
```

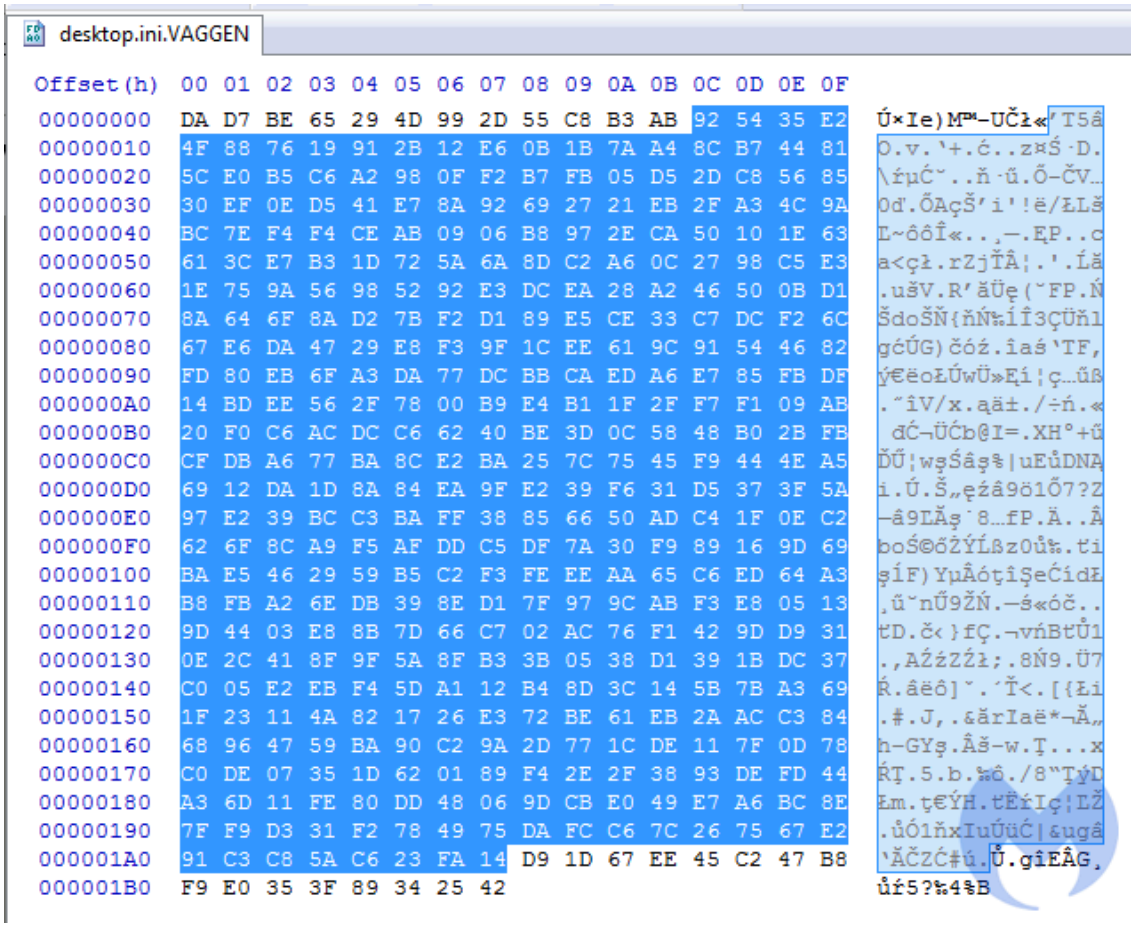
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



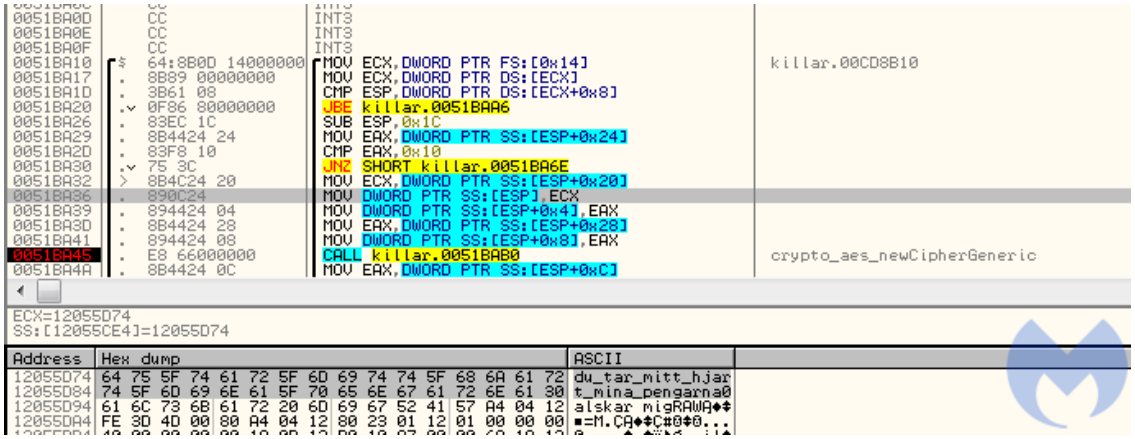
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

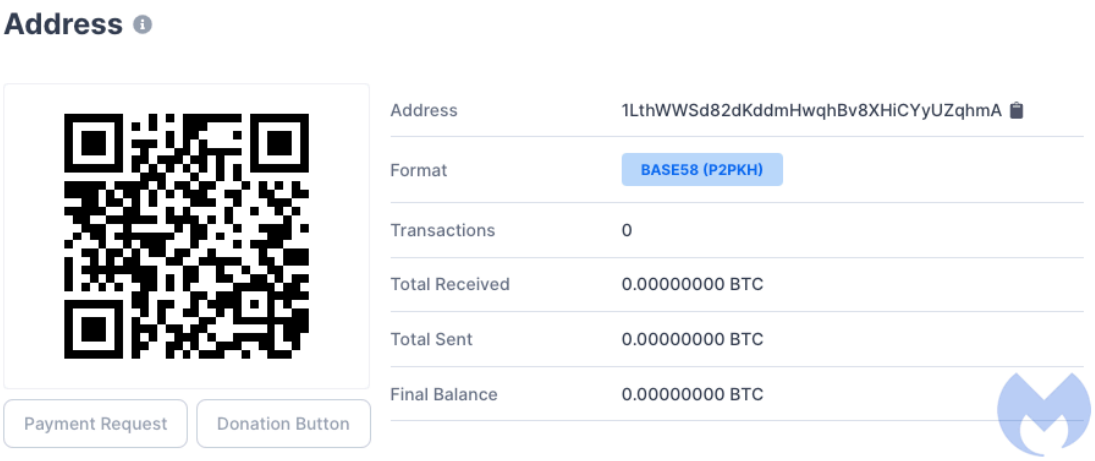
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

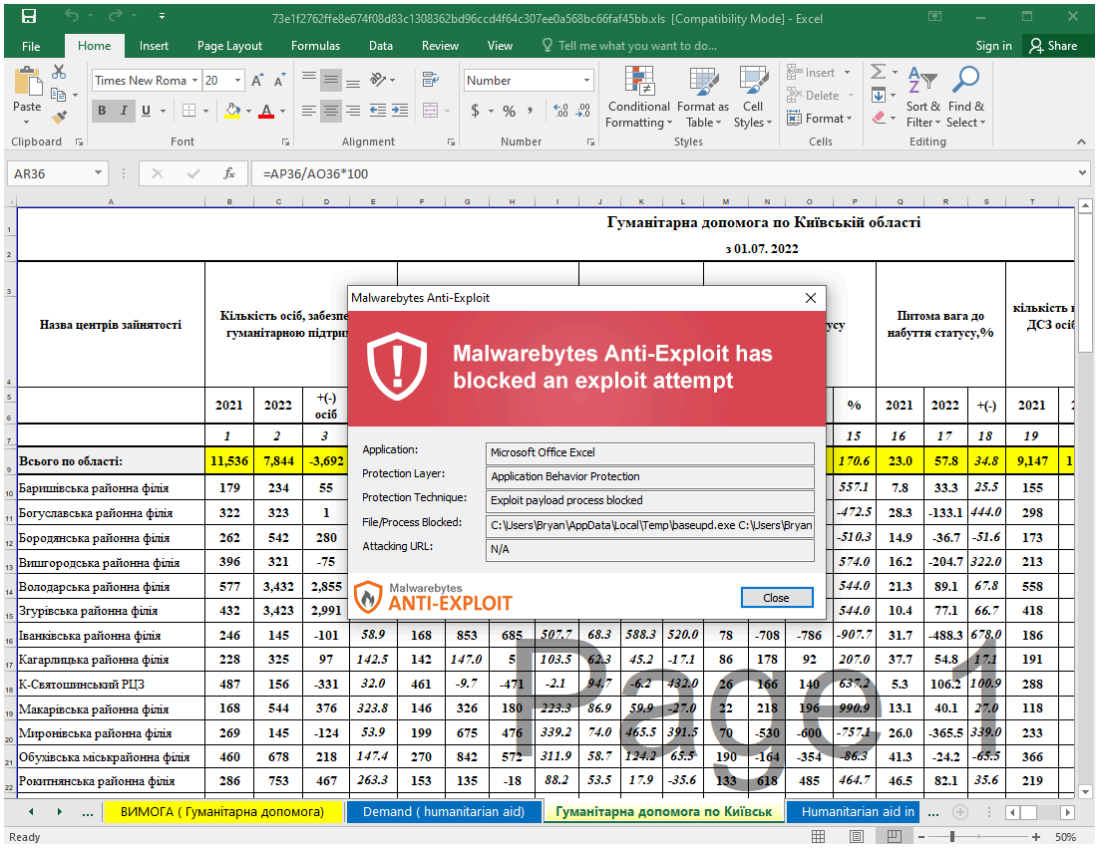
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

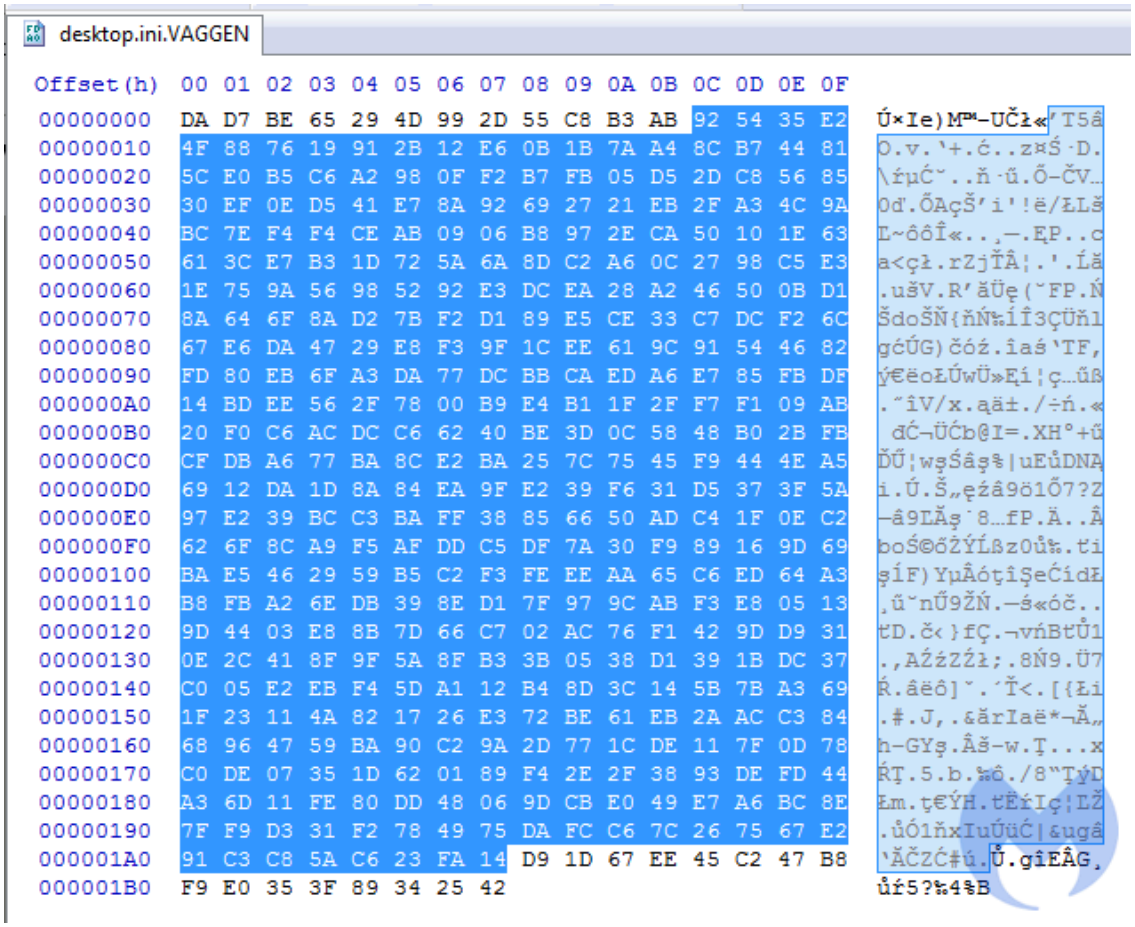
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

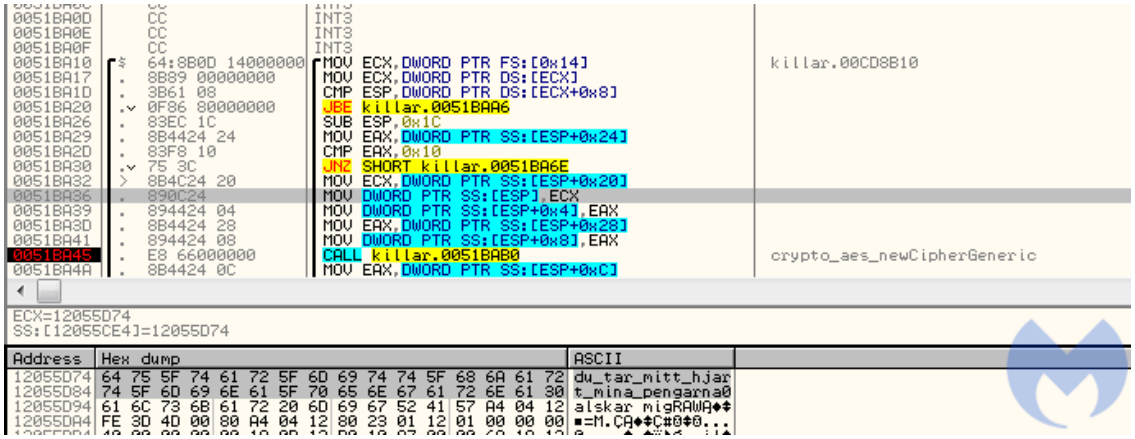


The content of the output file (with .VAGGEN extension) contains:

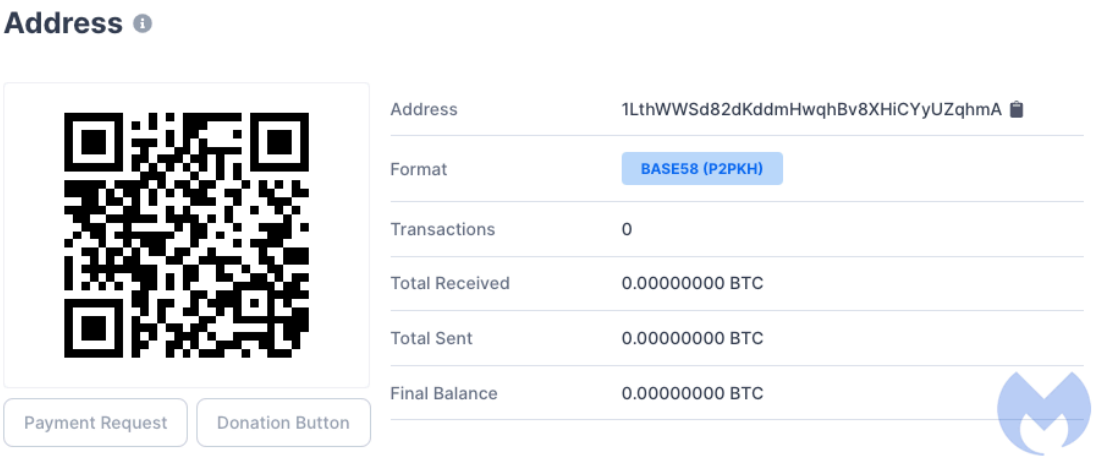
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

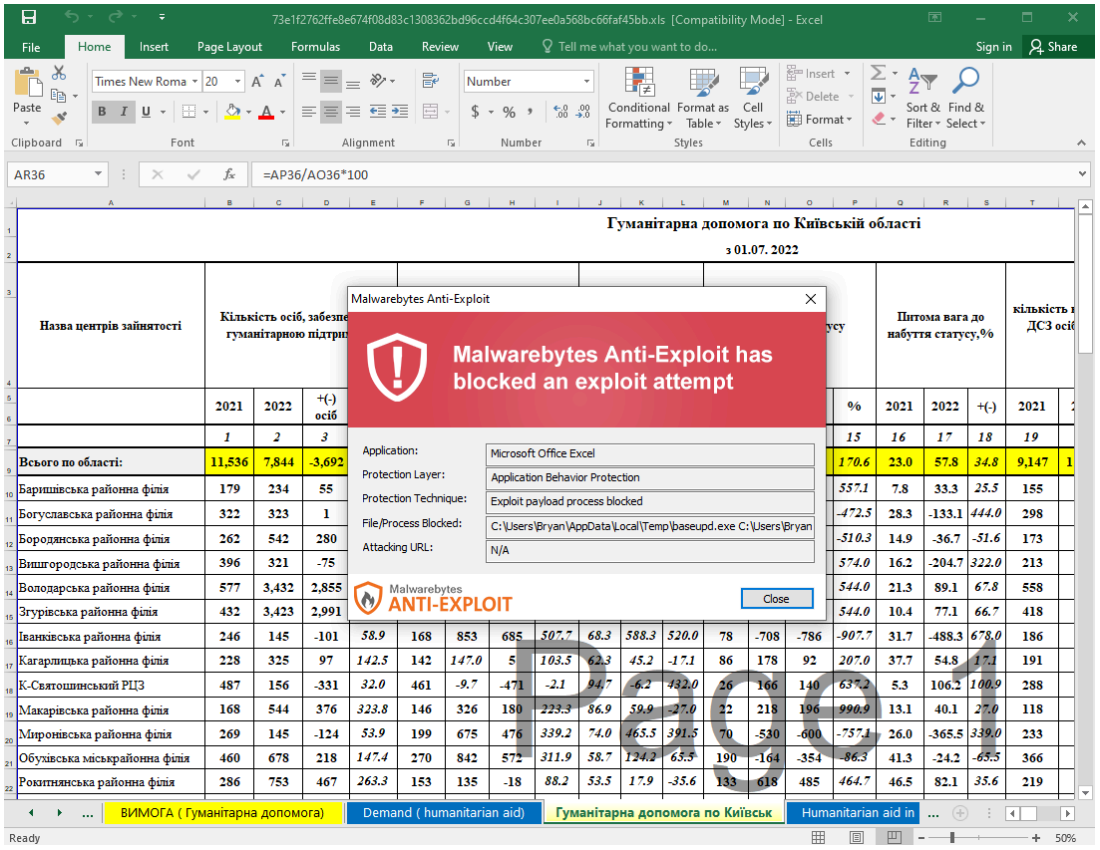
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

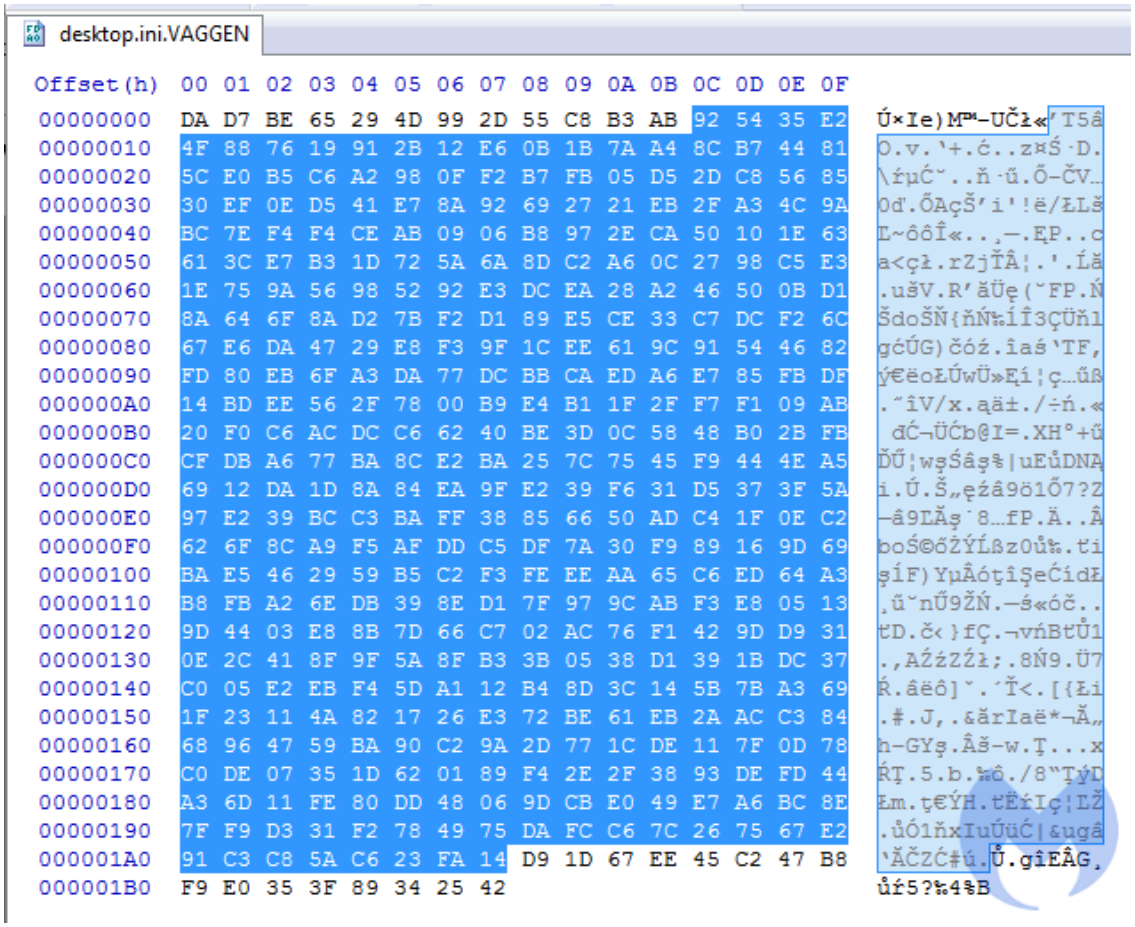
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

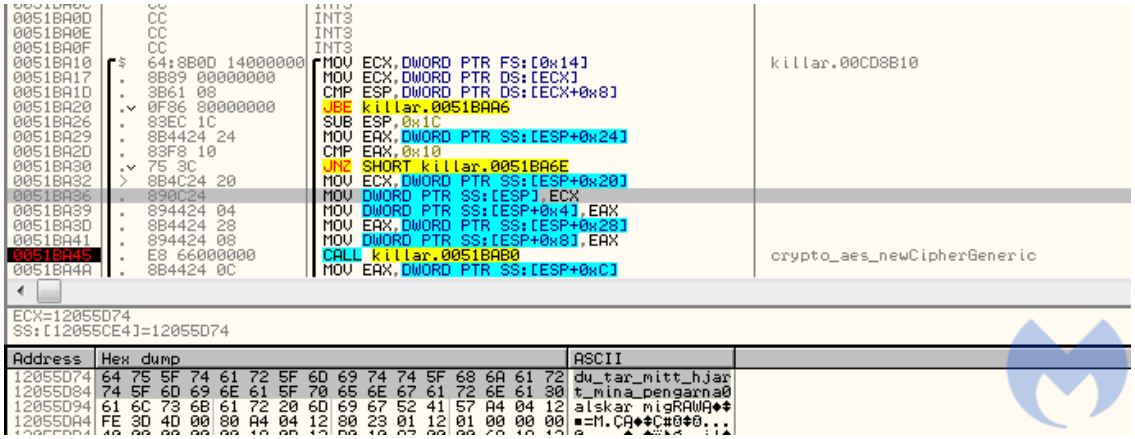


The content of the output file (with .VAGGEN extension) contains:

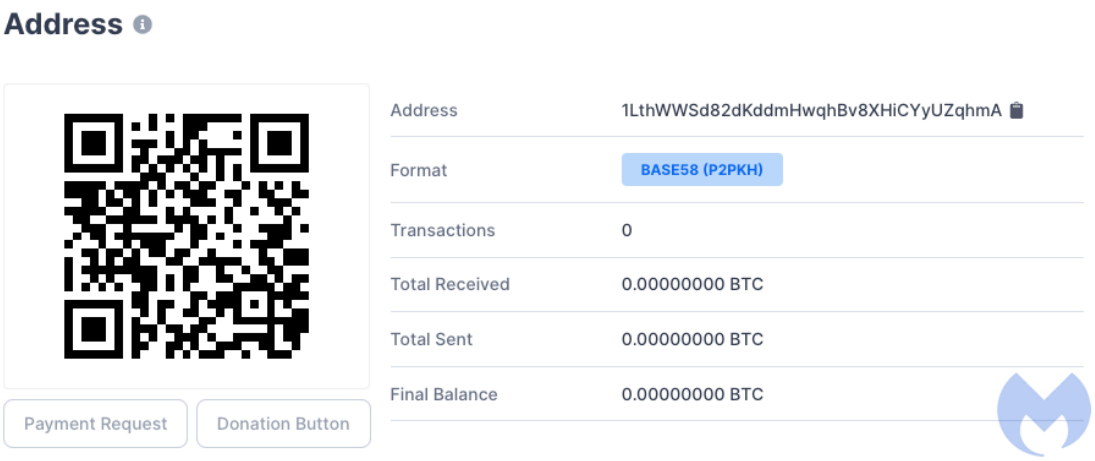
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

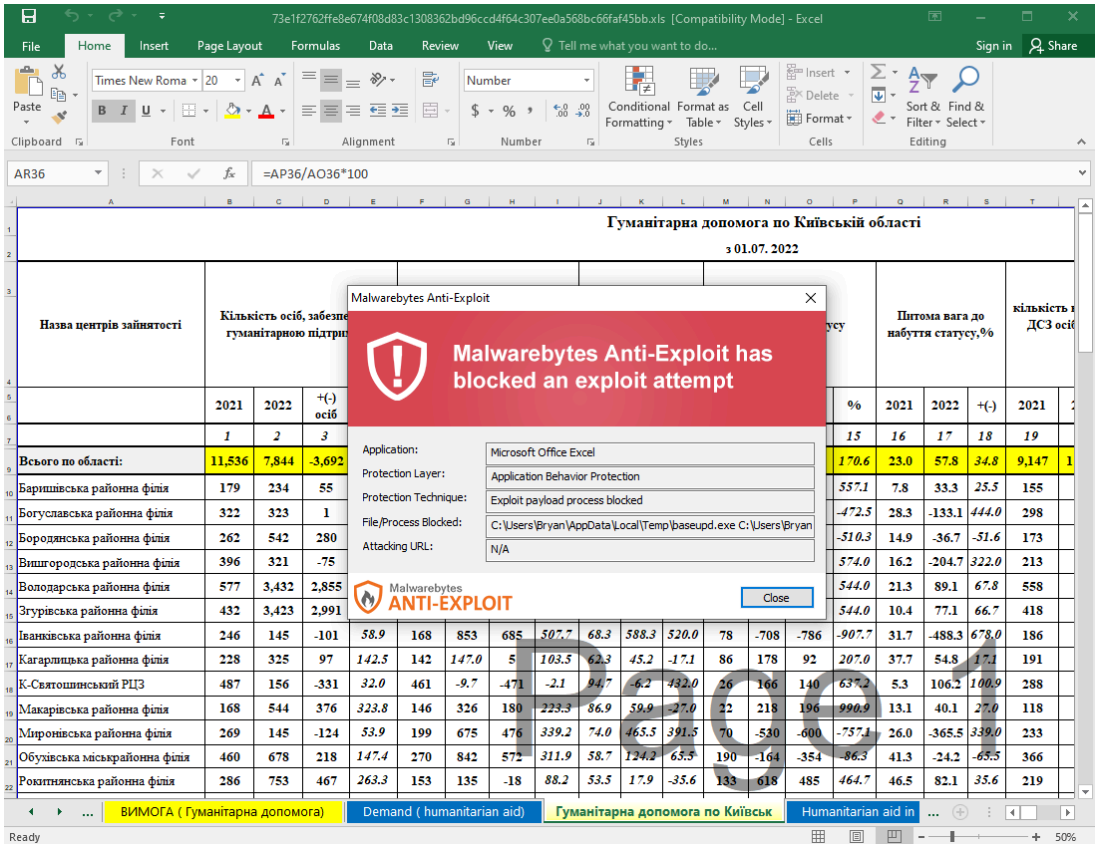
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

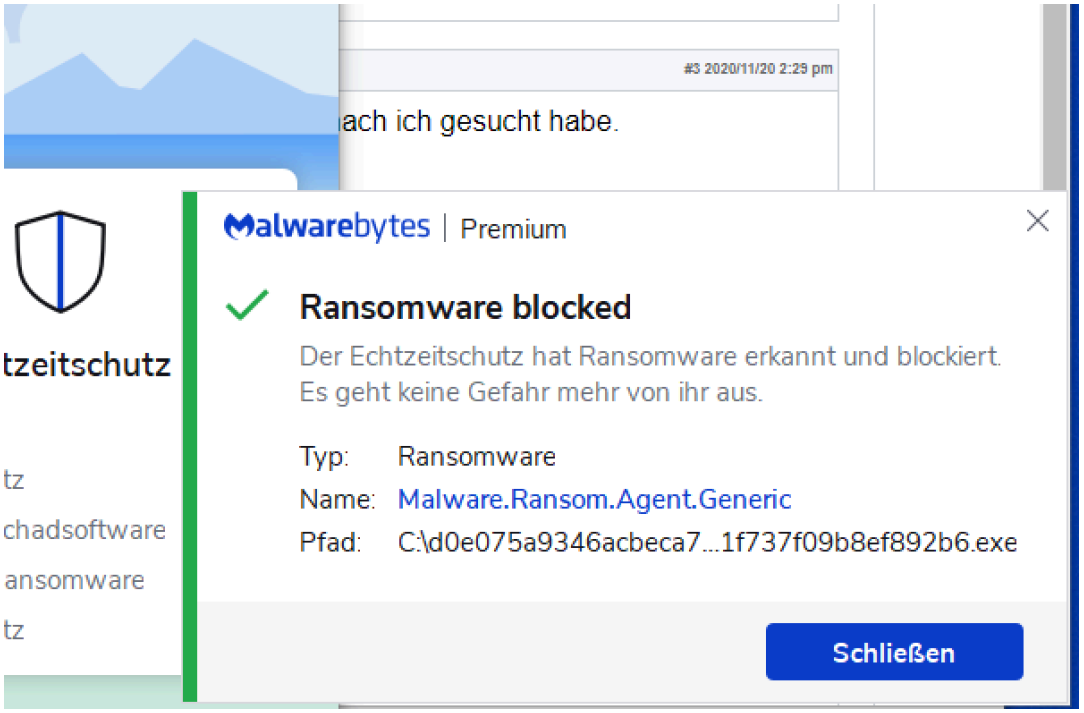
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```



Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

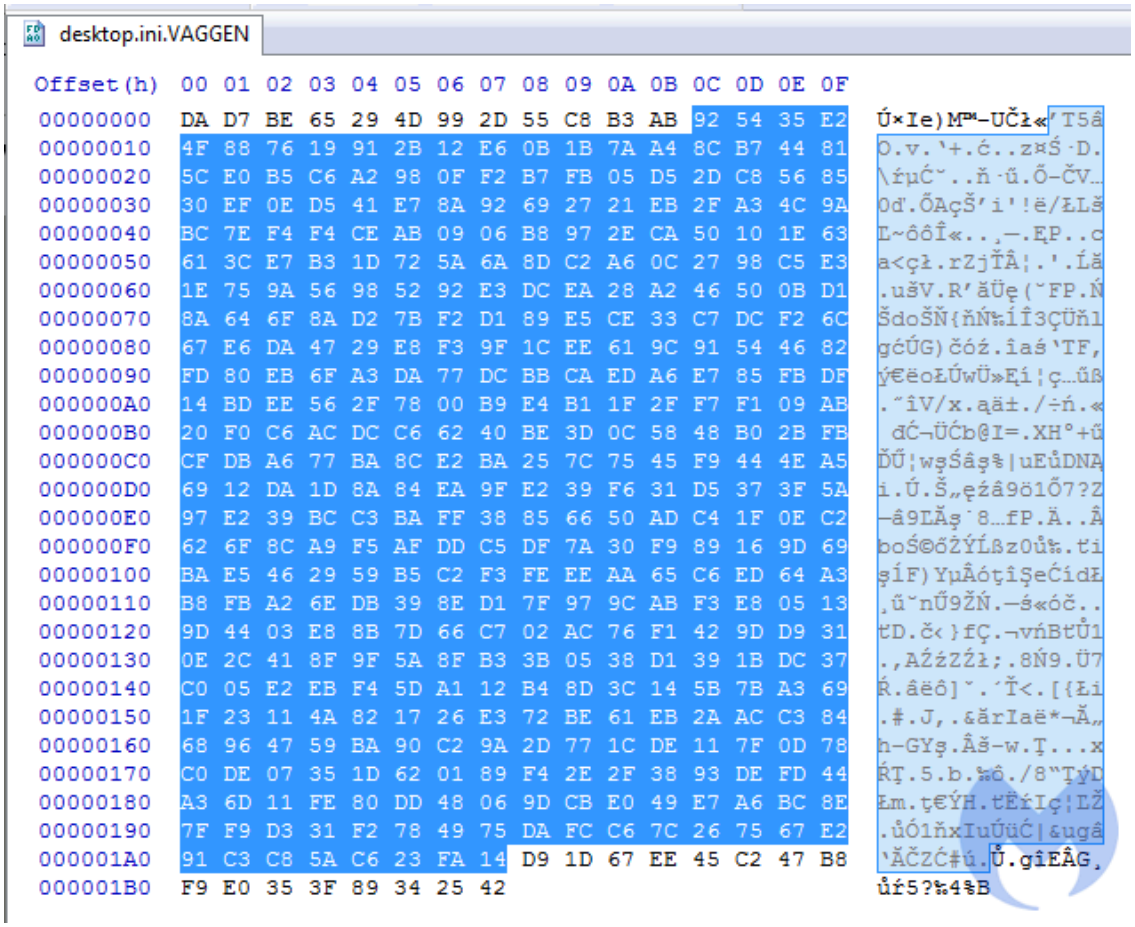


The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

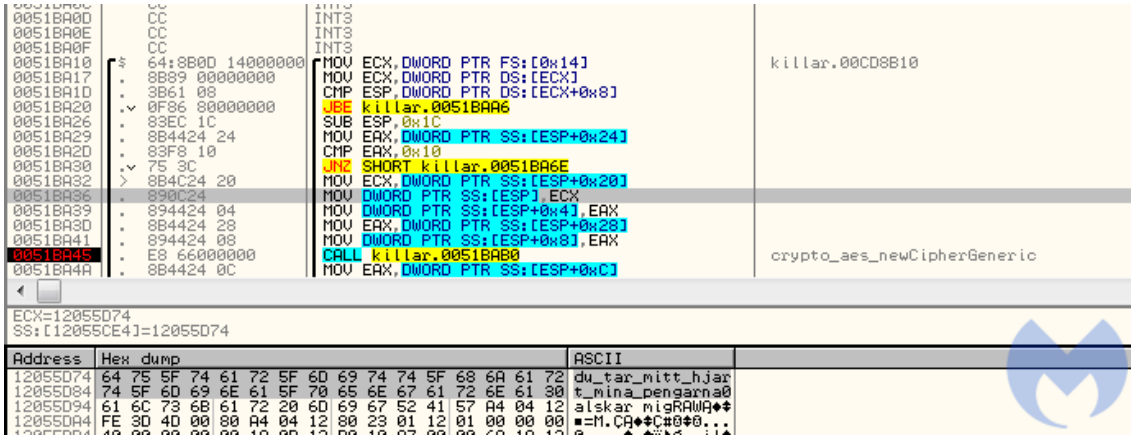


The content of the output file (with .VAGGEN extension) contains:

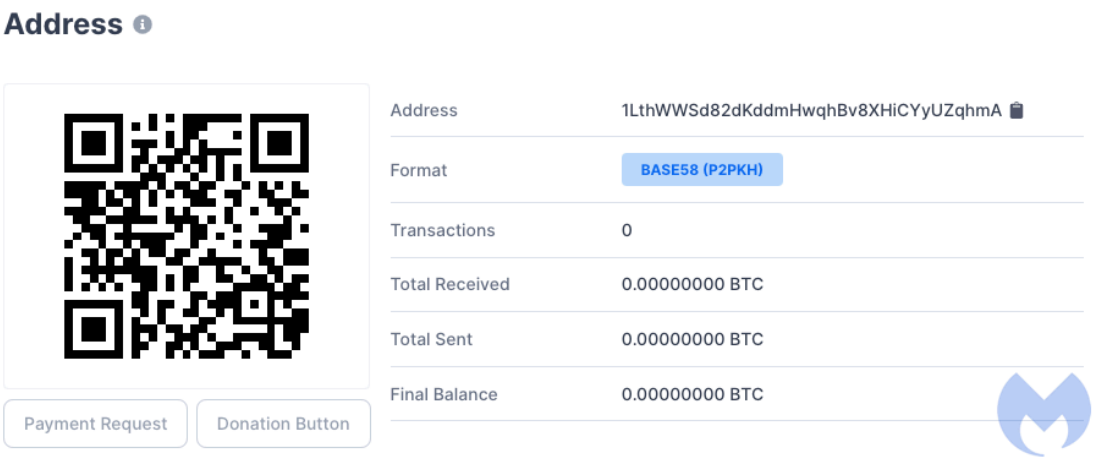
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

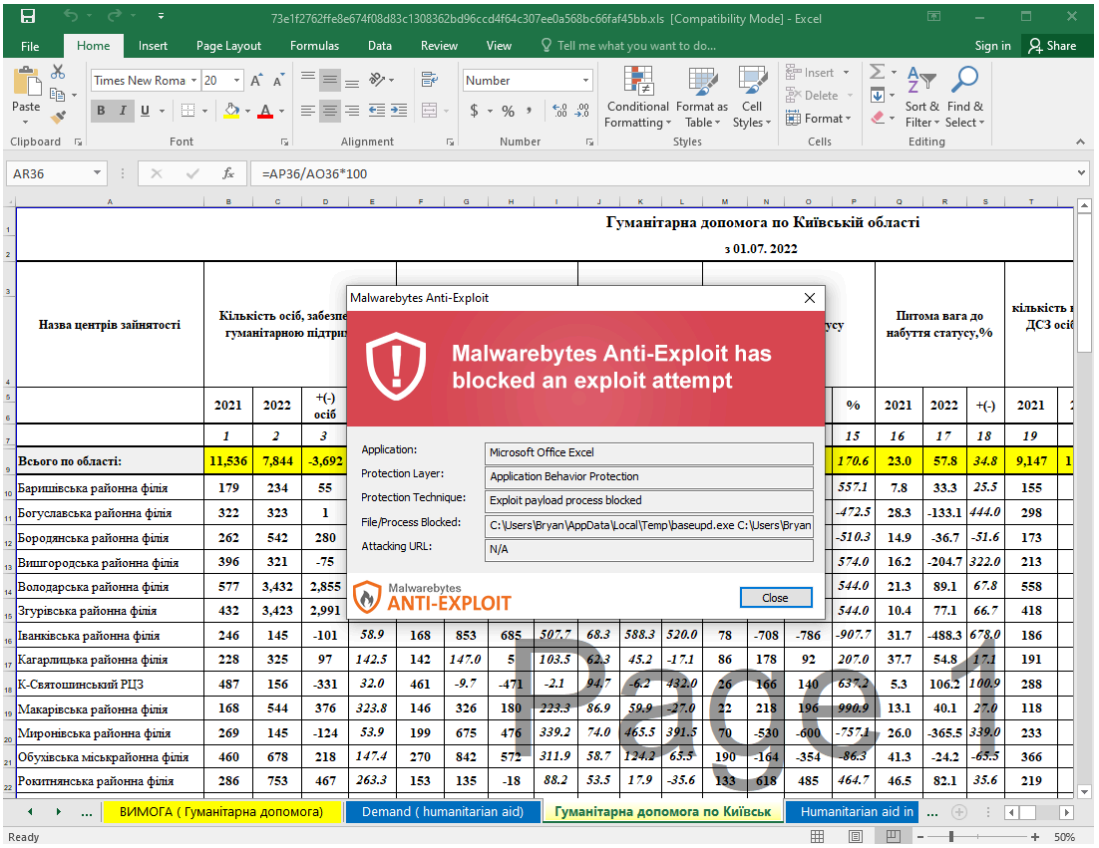
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

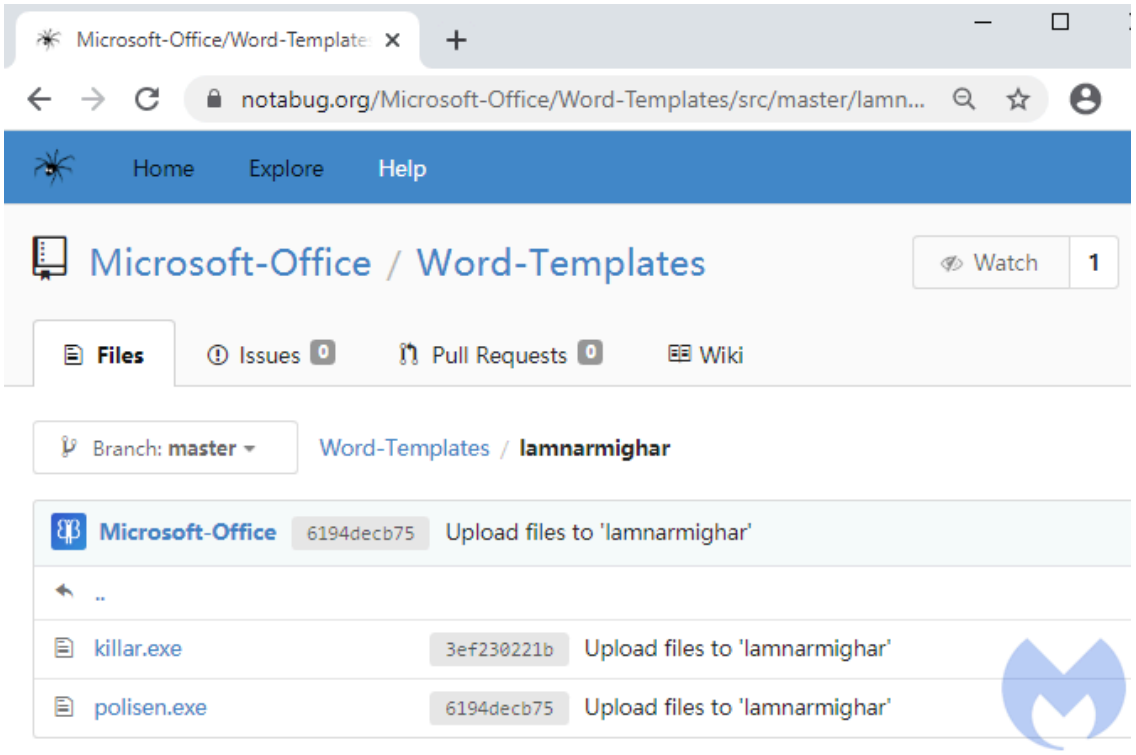
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



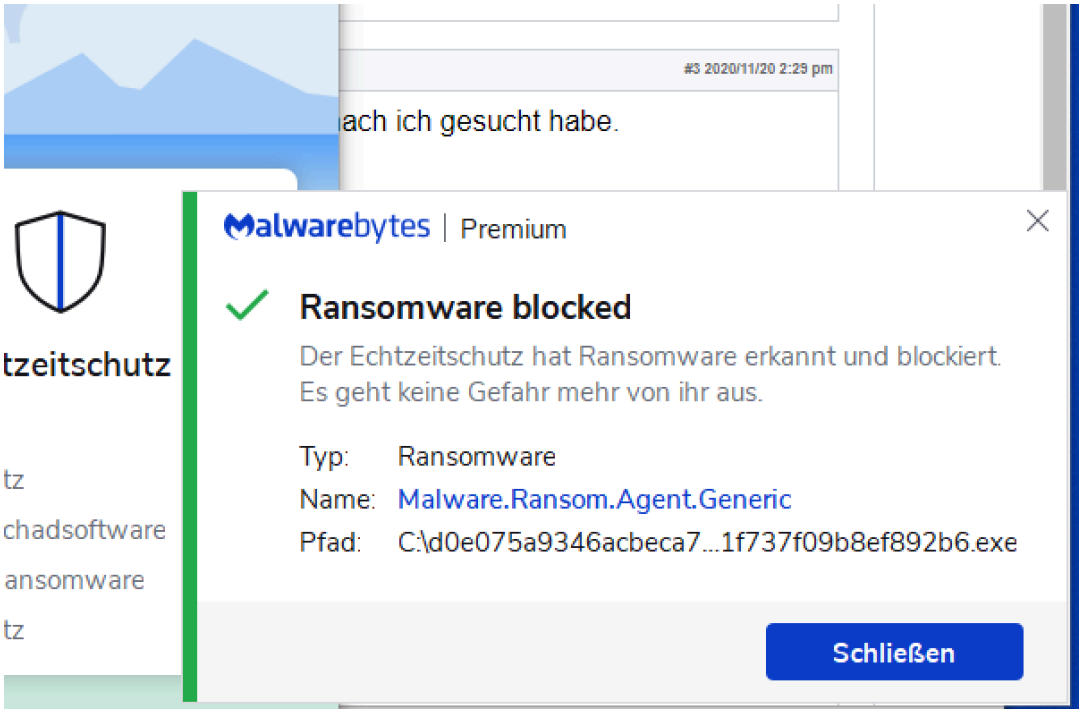
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

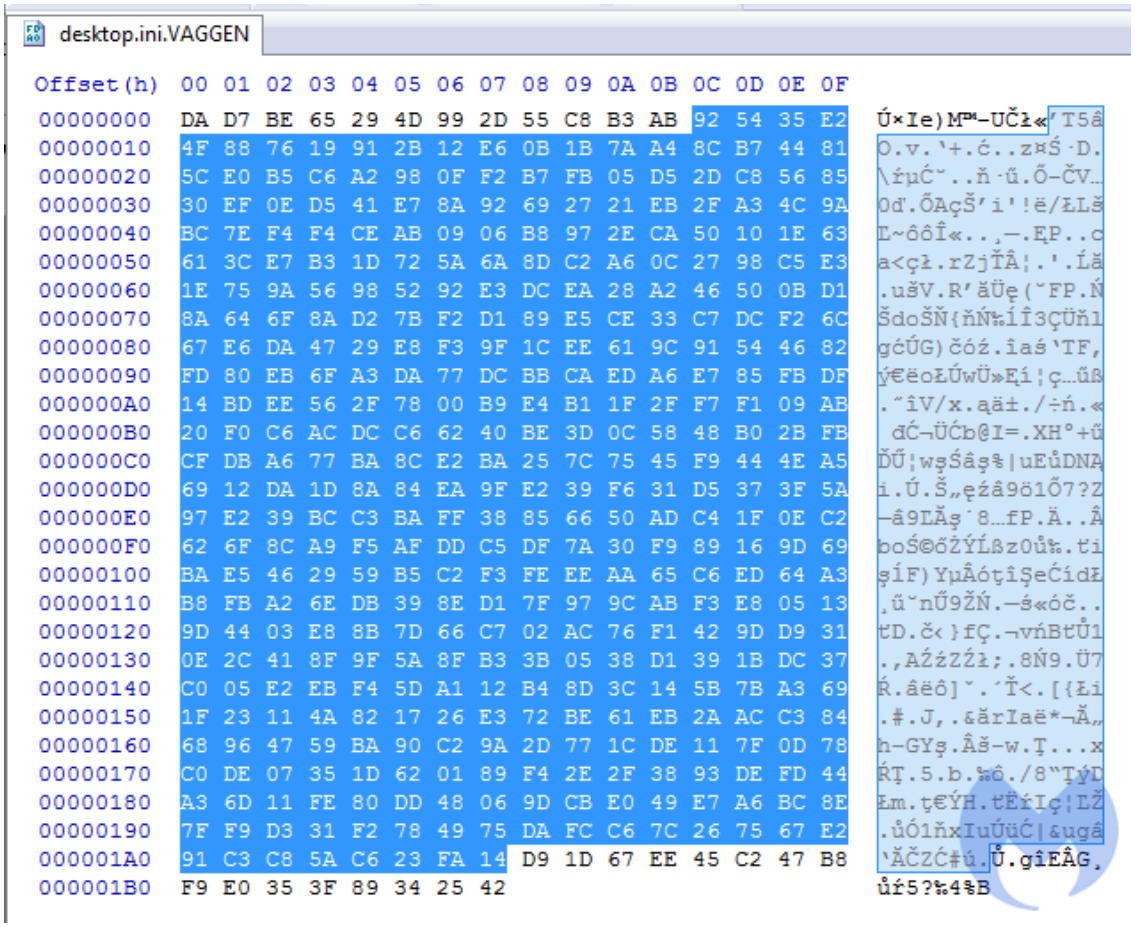
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

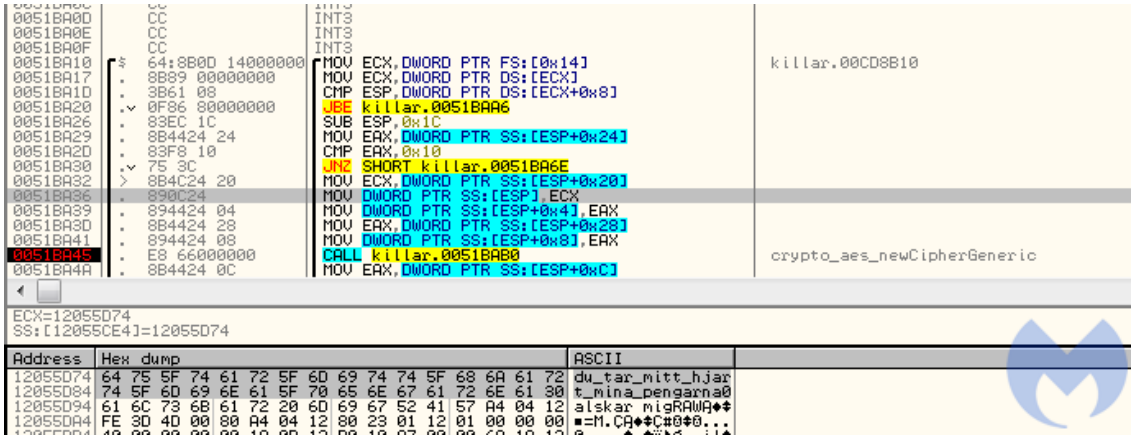


The content of the output file (with .VAGGEN extension) contains:

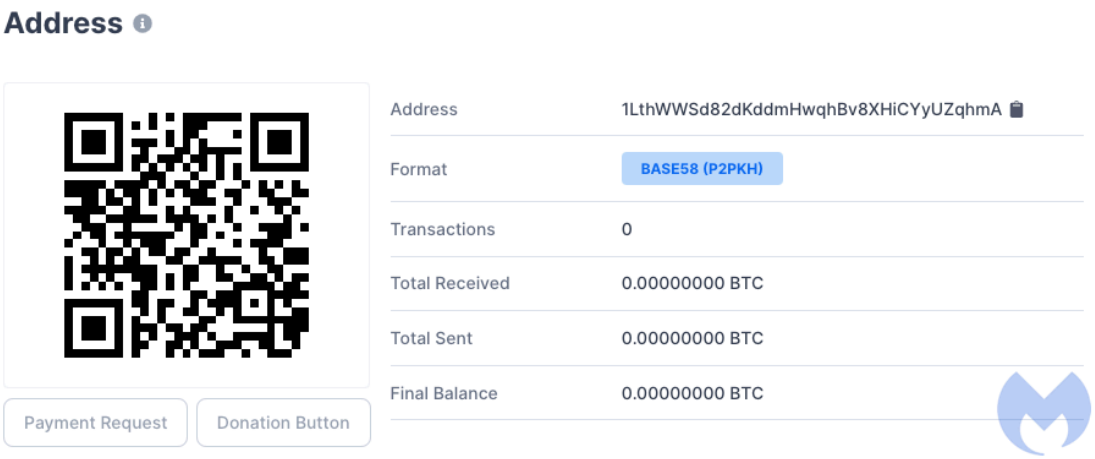
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

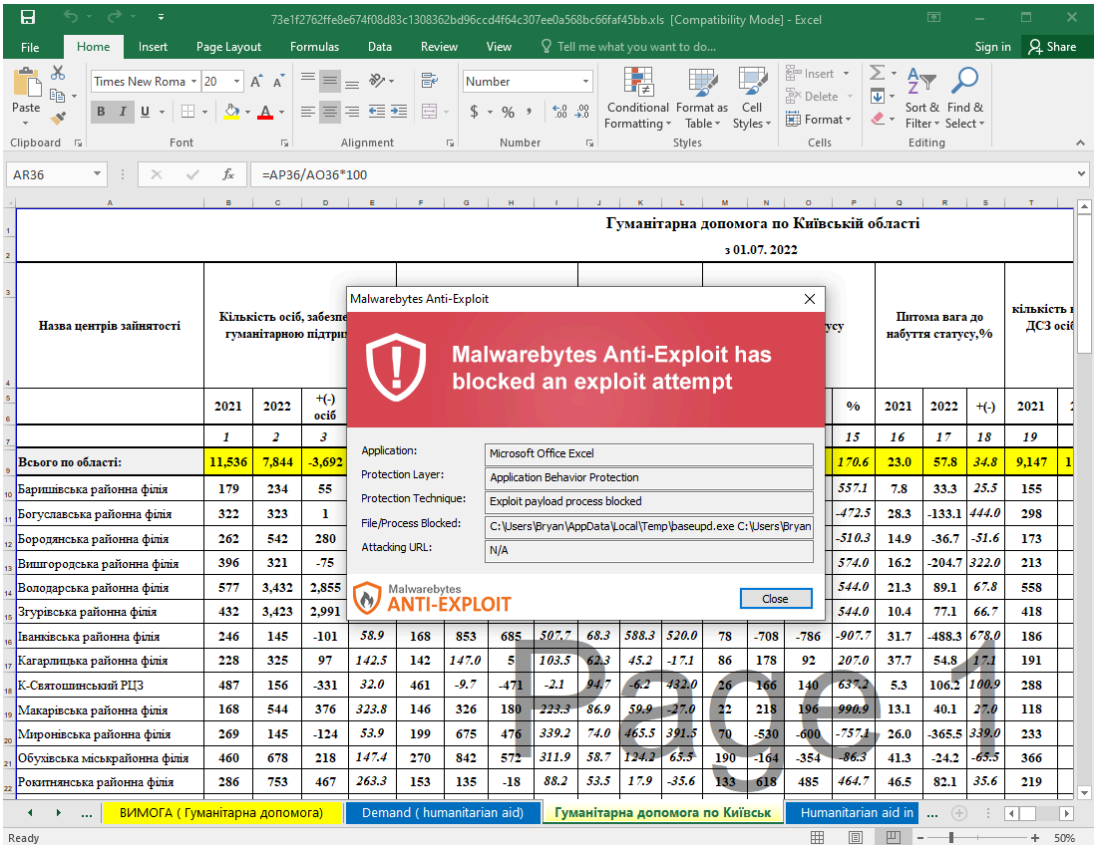
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

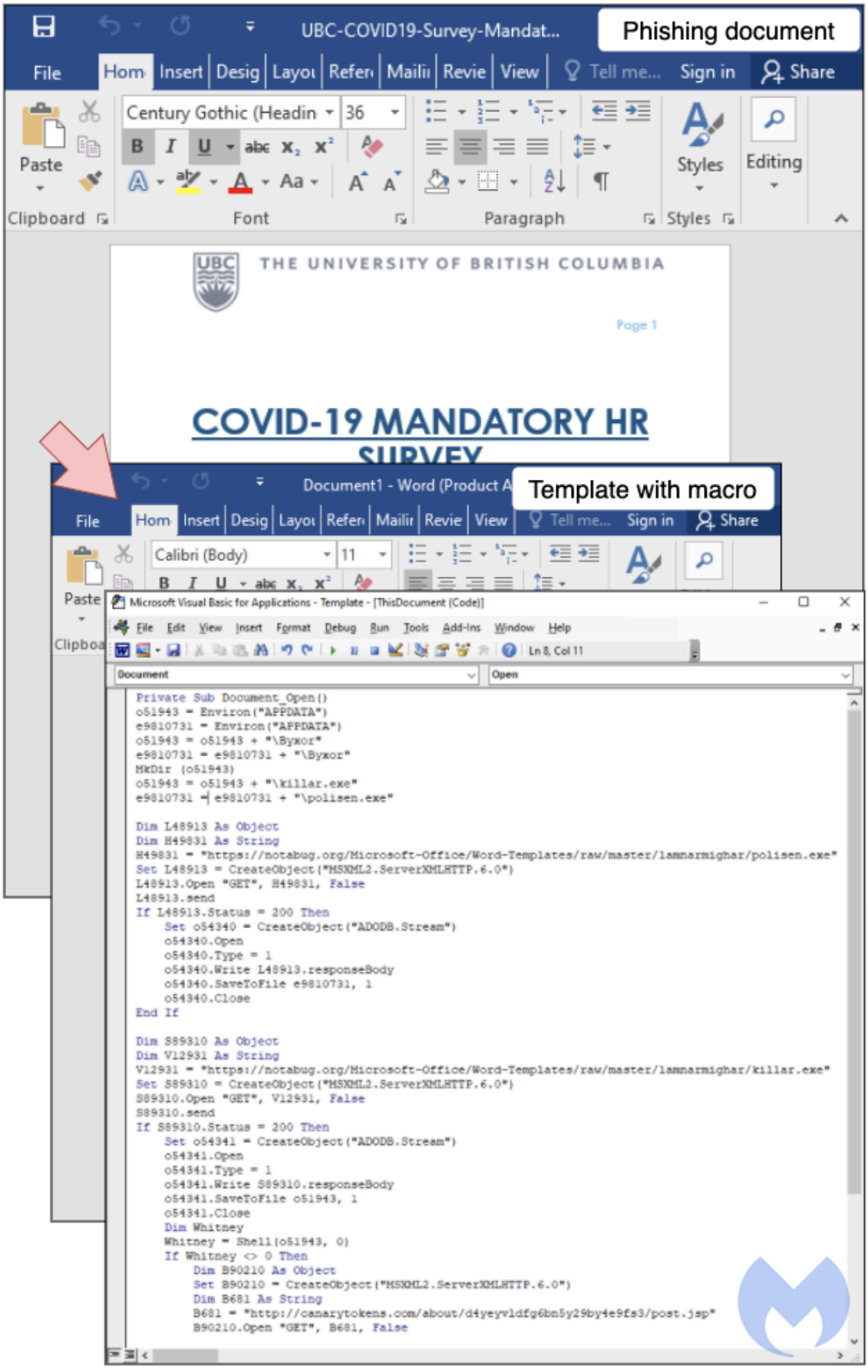
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

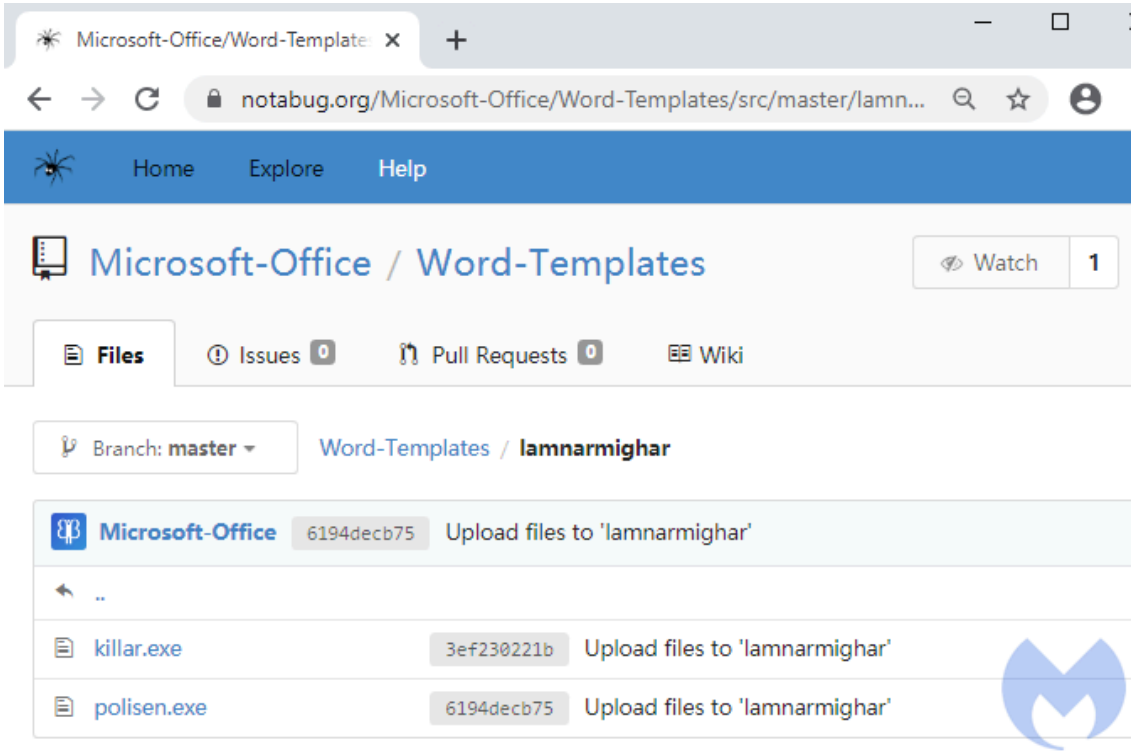
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



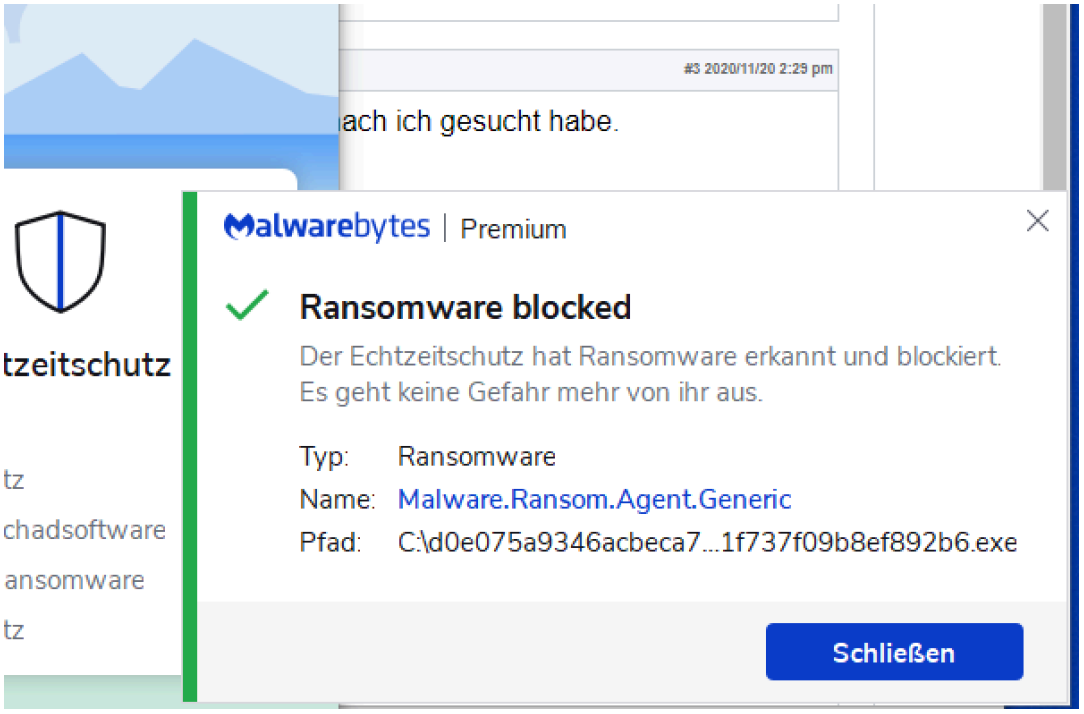
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

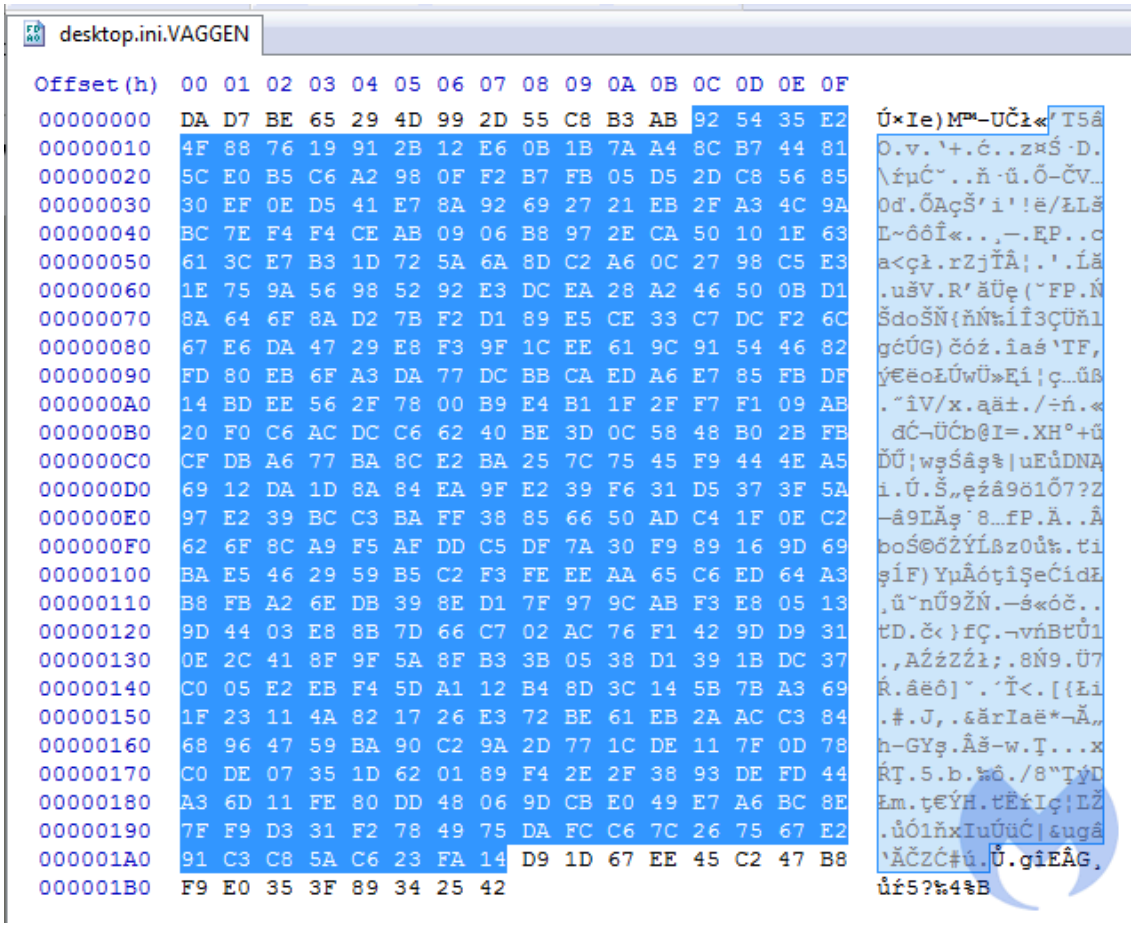
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

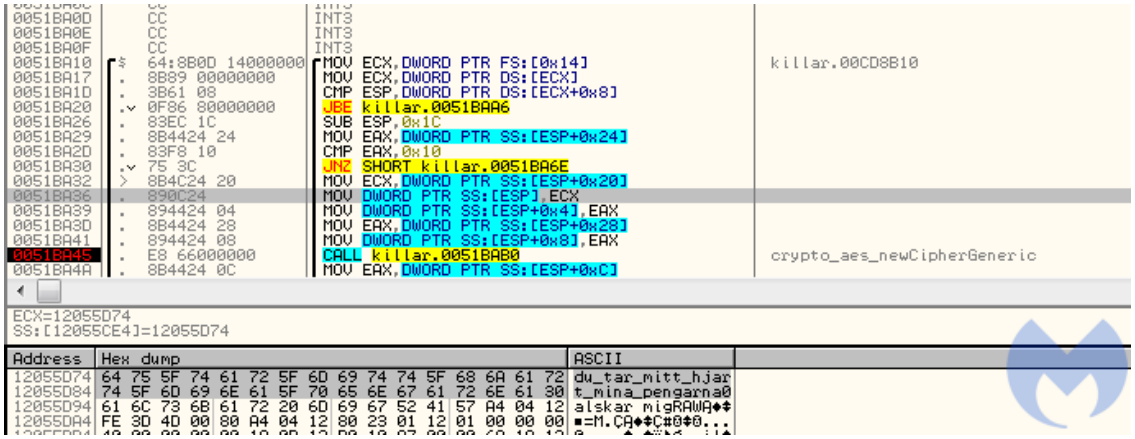


The content of the output file (with .VAGGEN extension) contains:

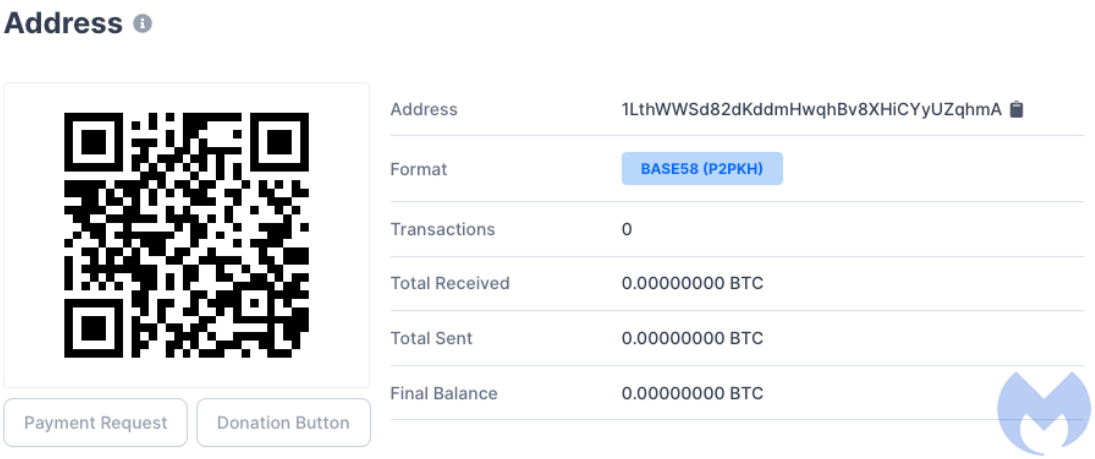
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

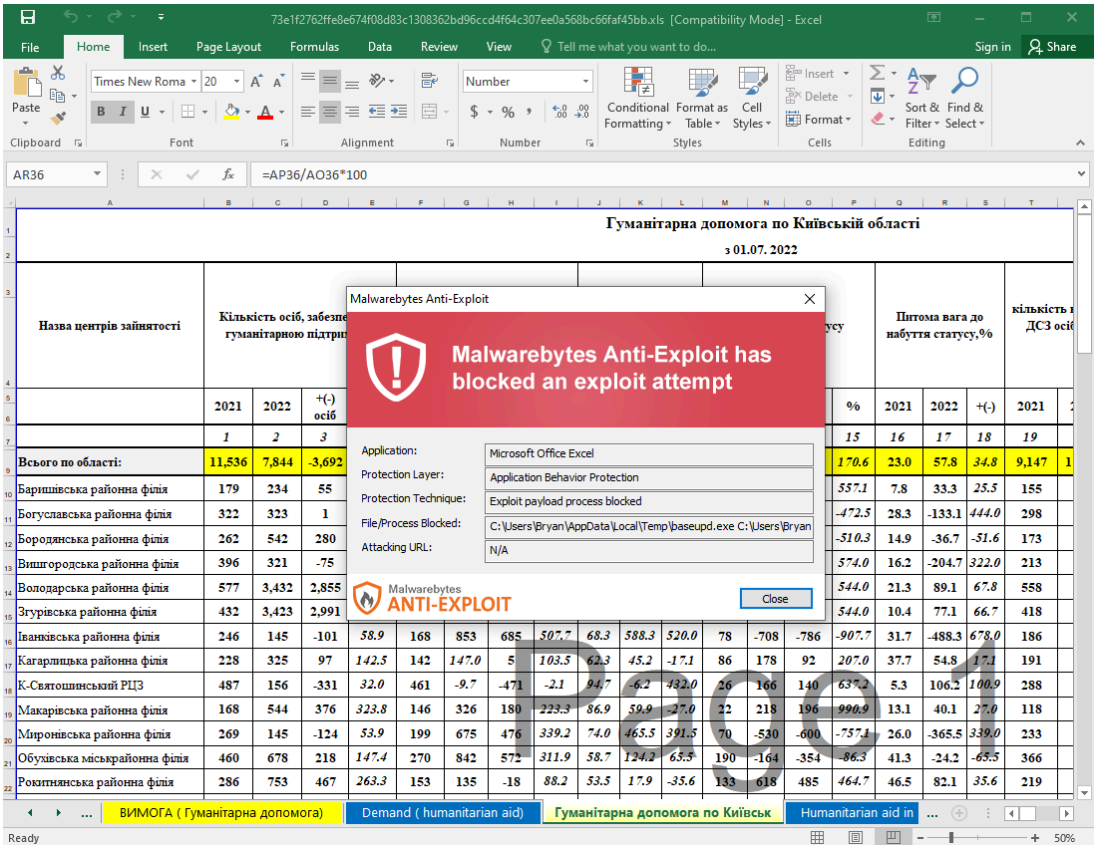
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

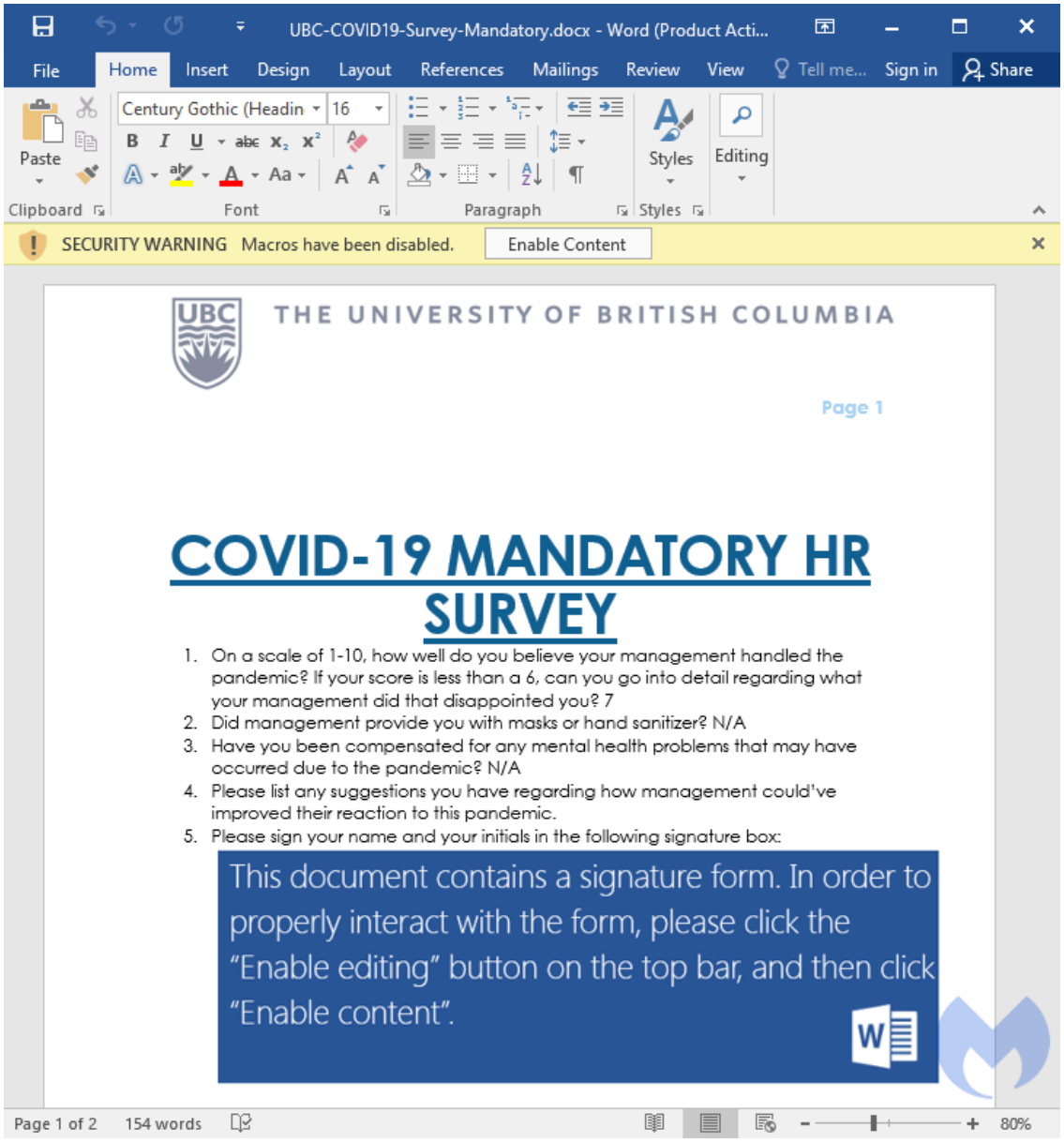
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

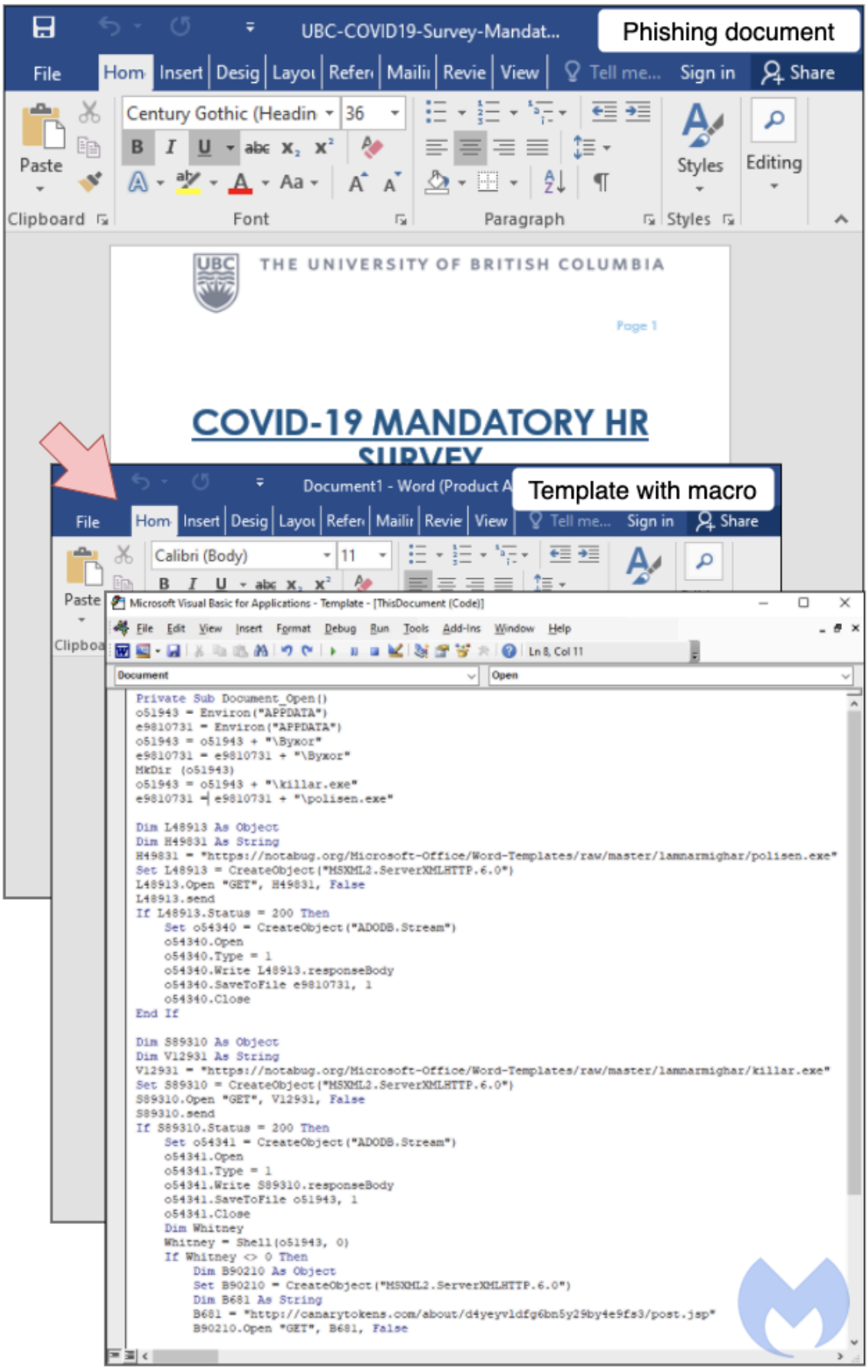
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

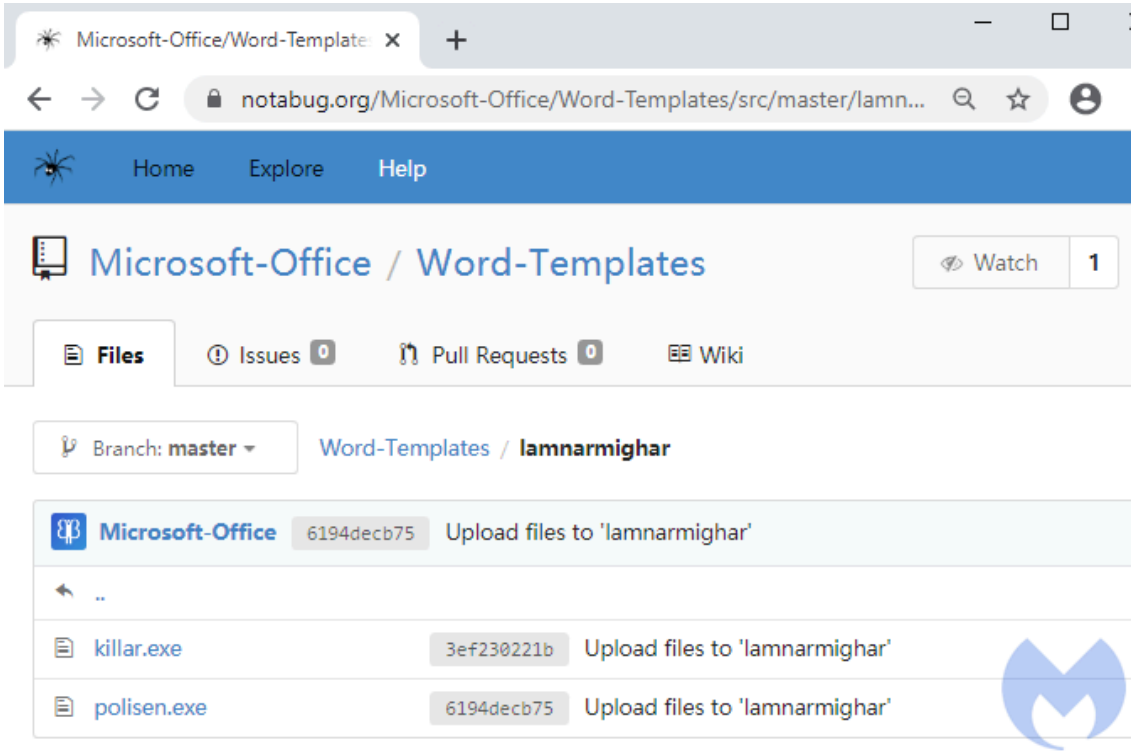
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



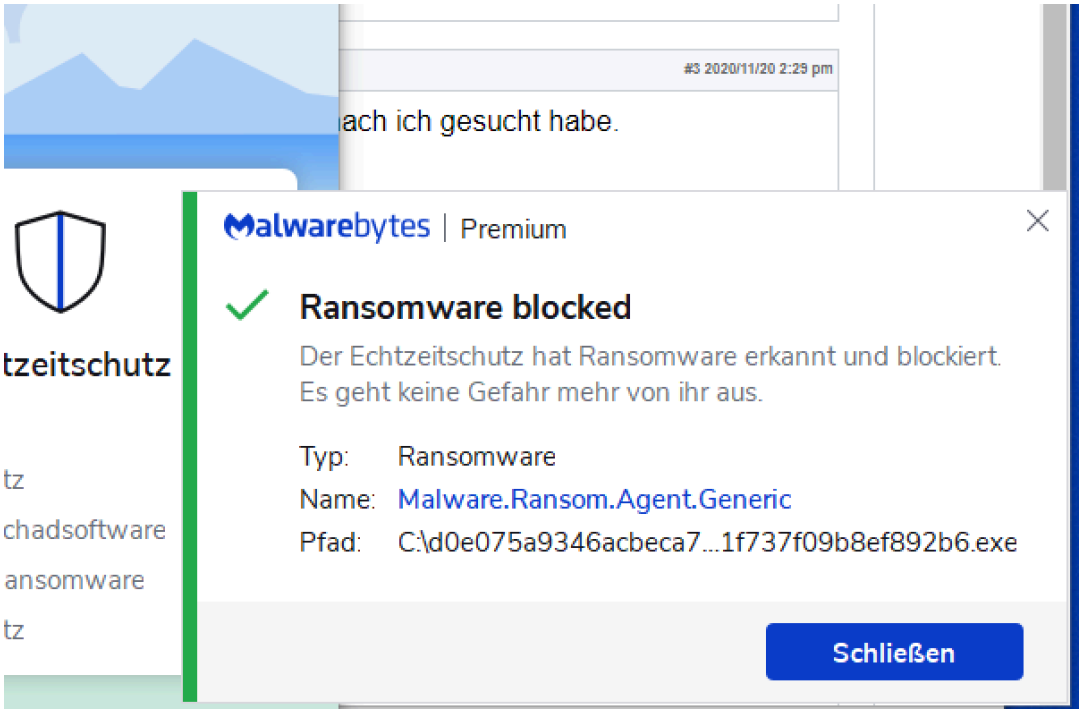
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

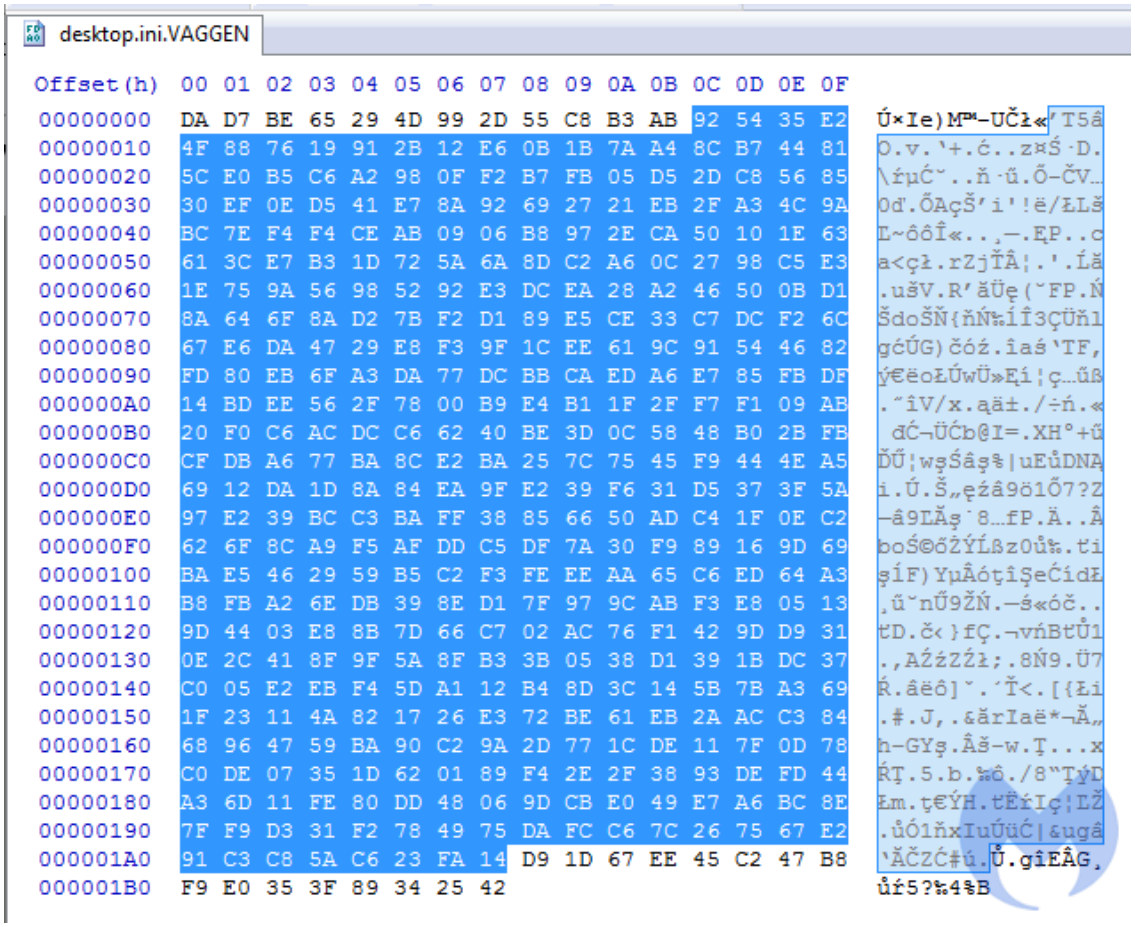
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



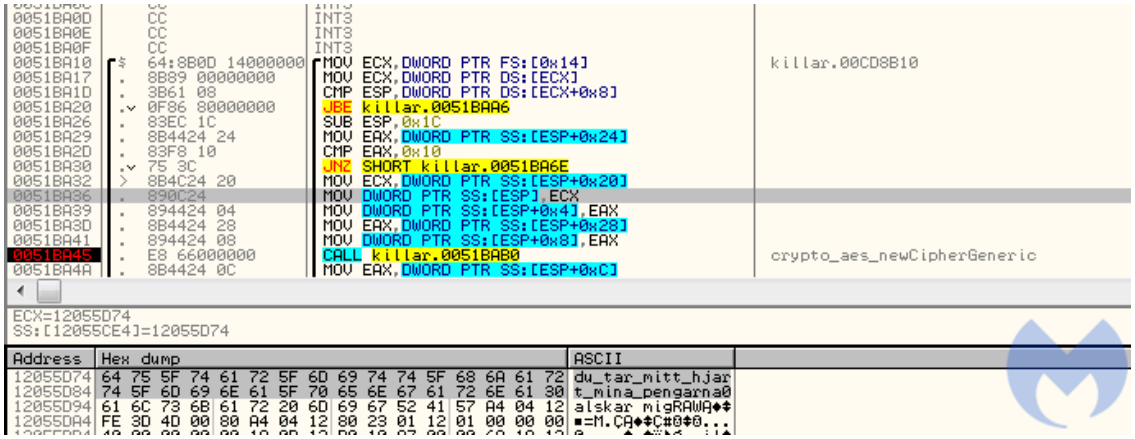
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

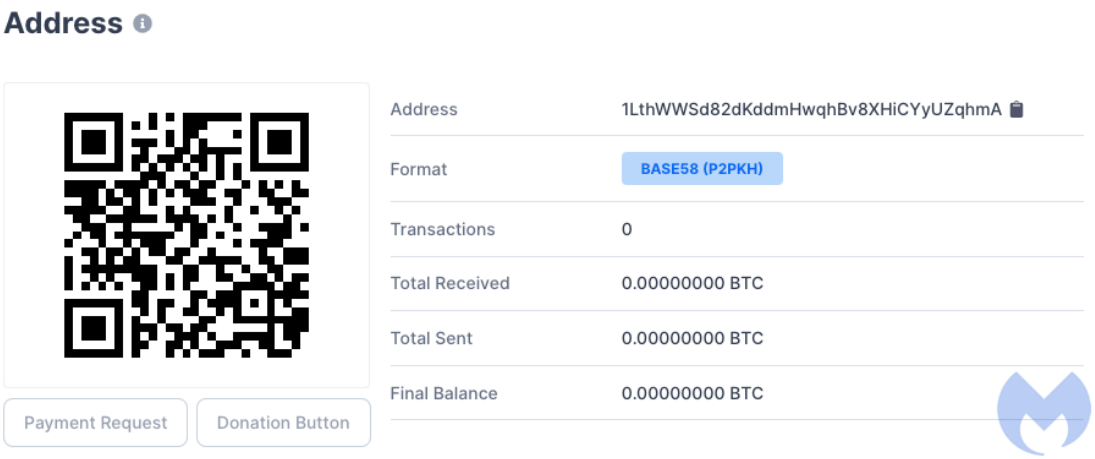
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

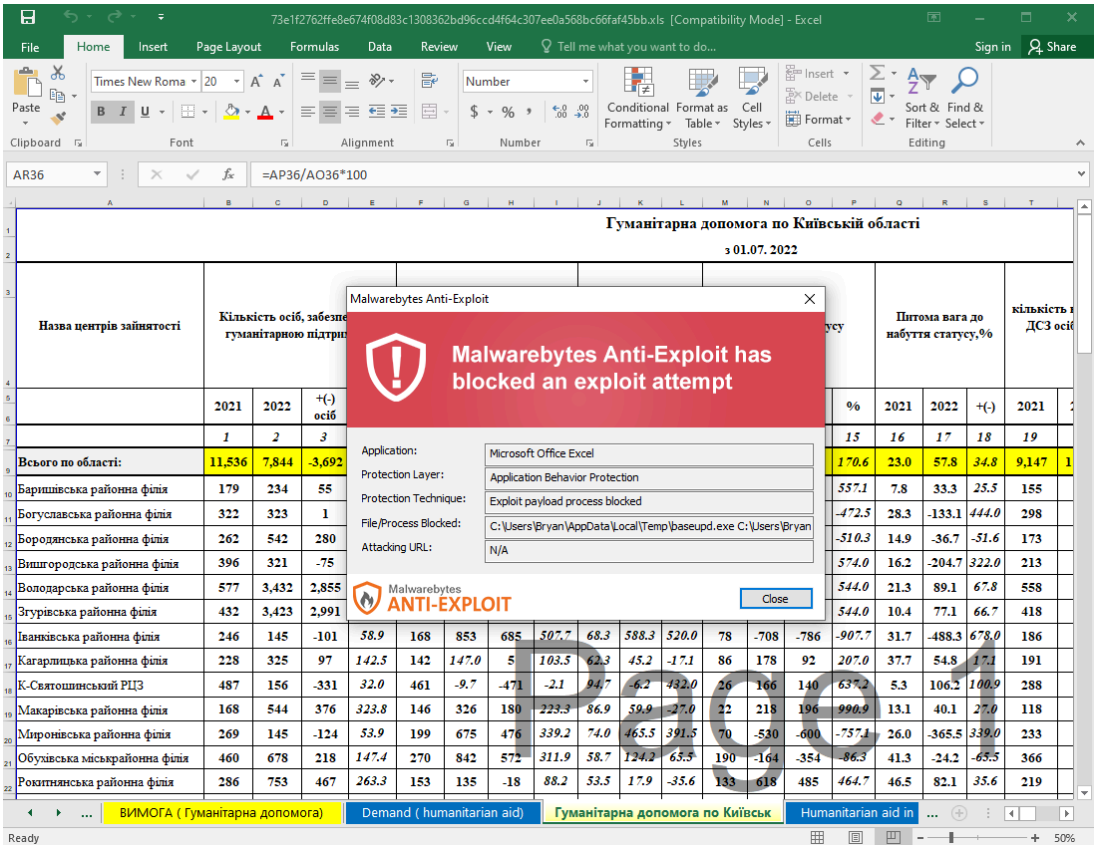
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

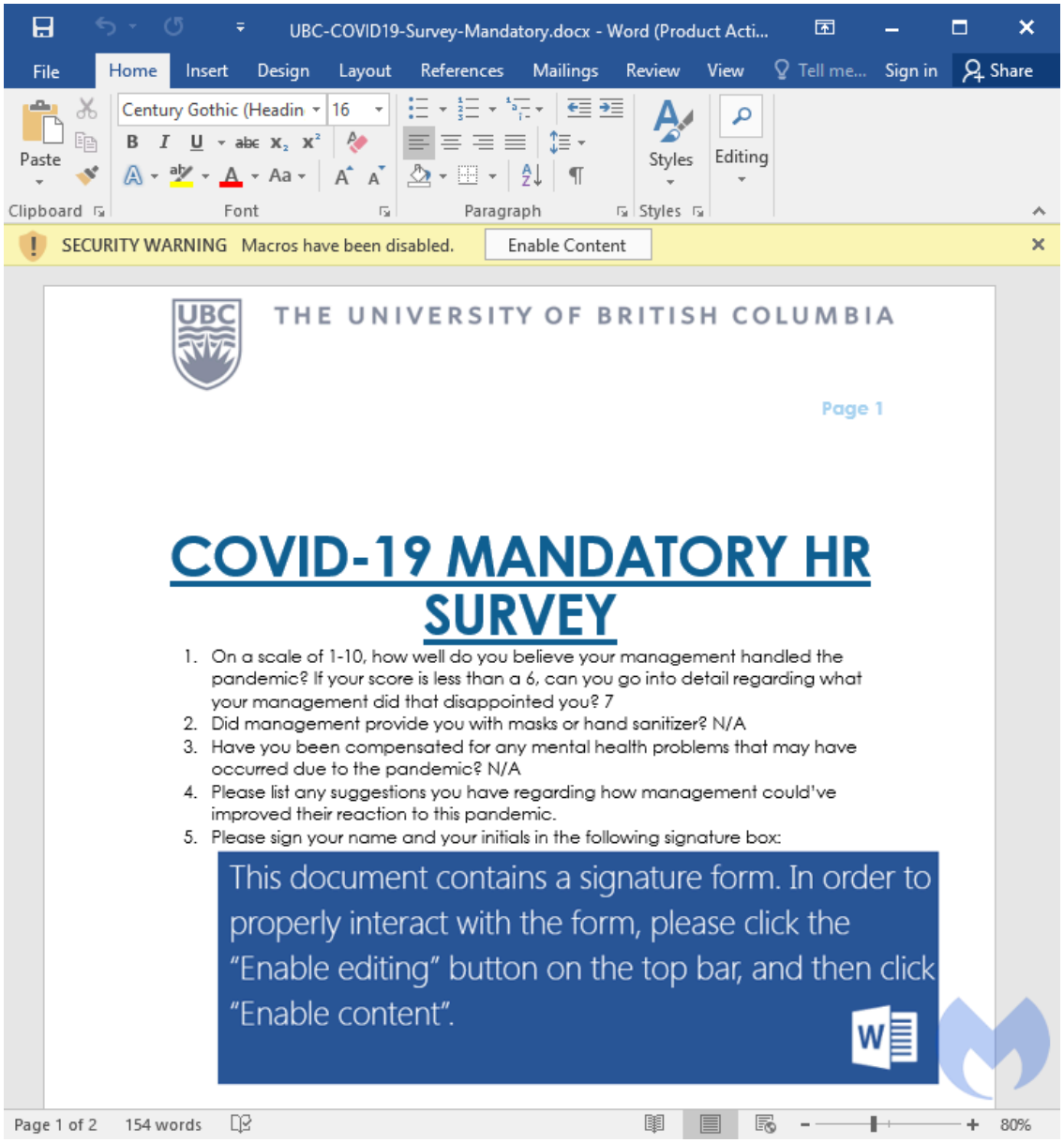
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

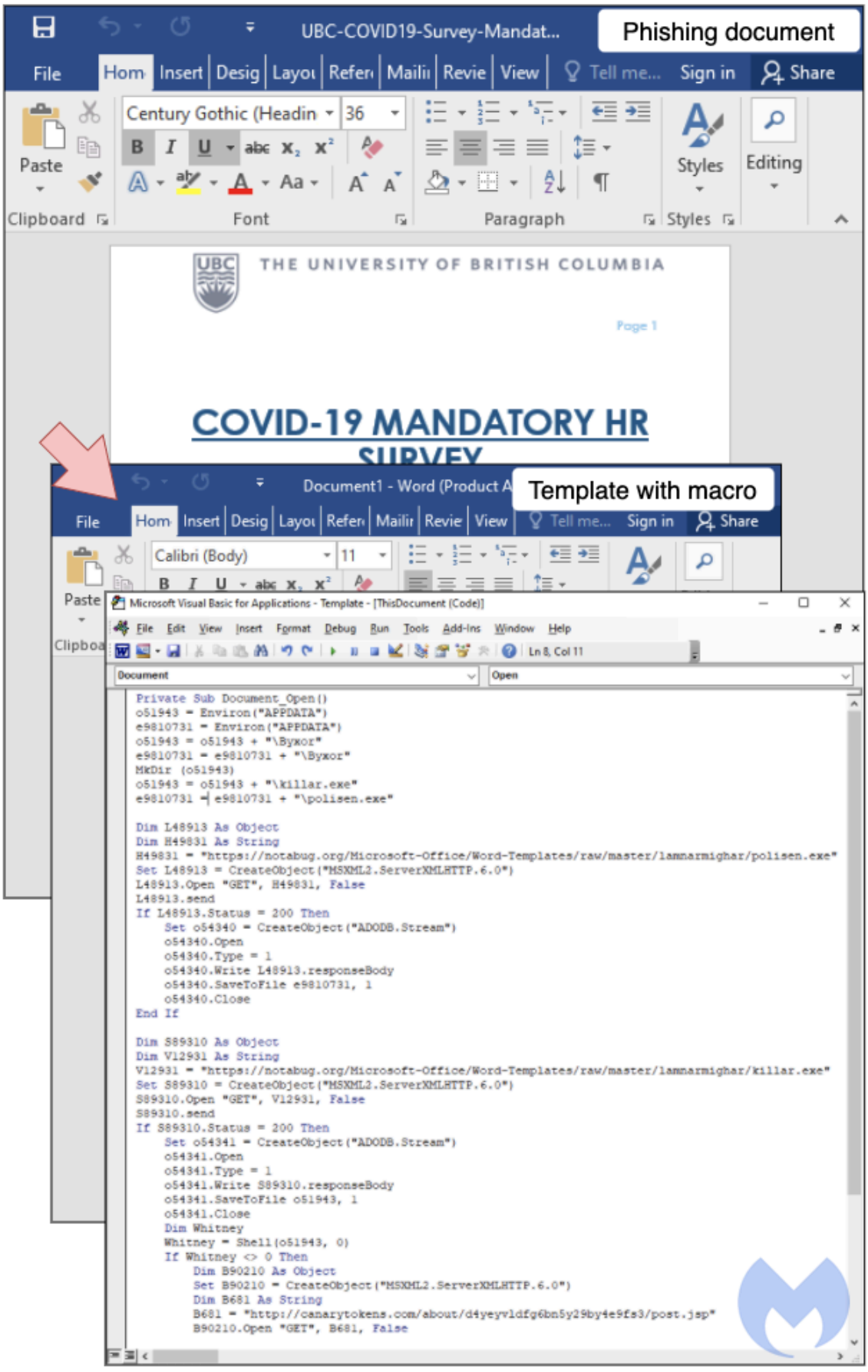
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

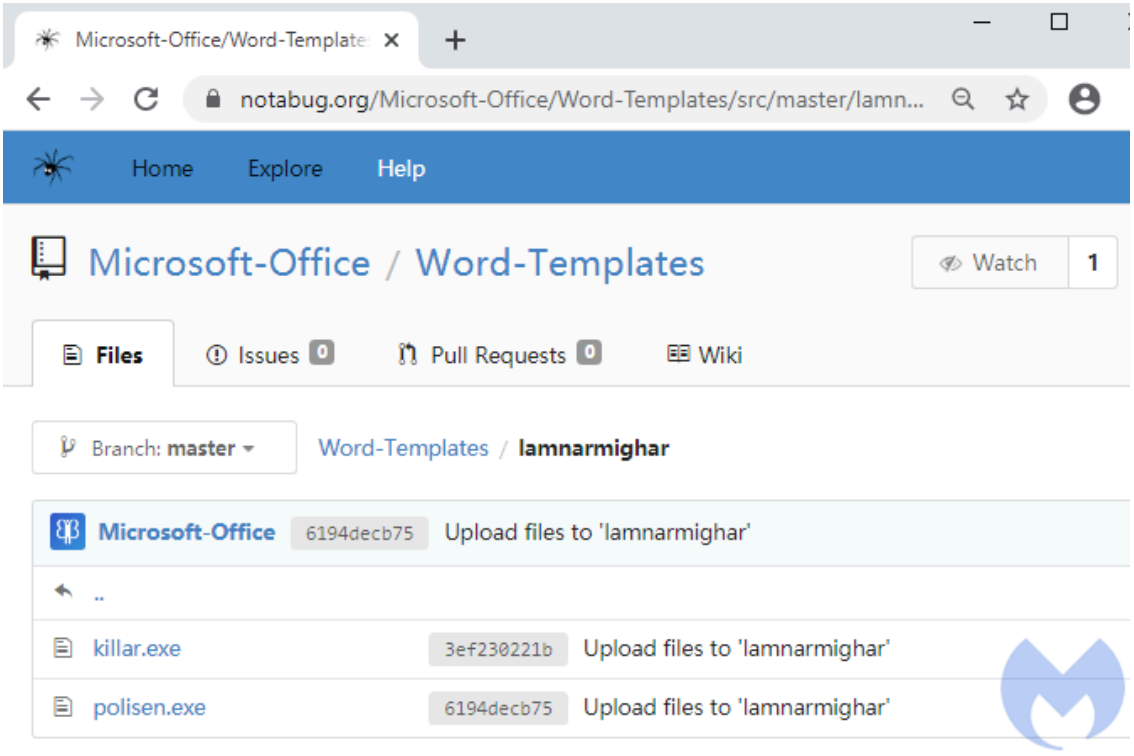
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



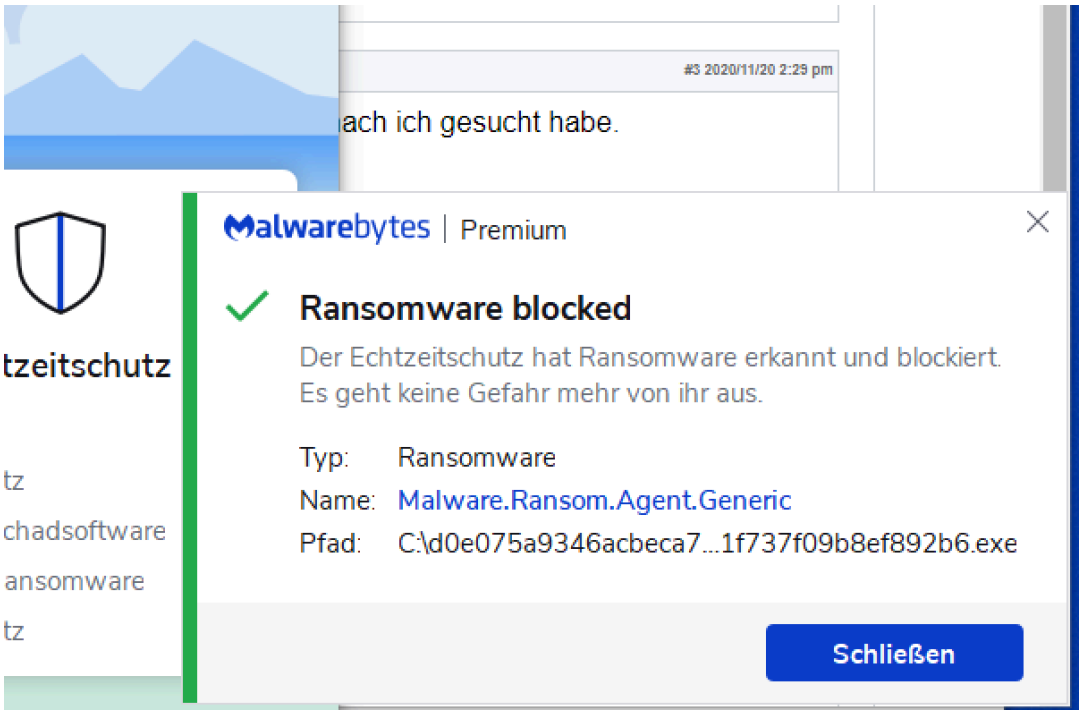
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

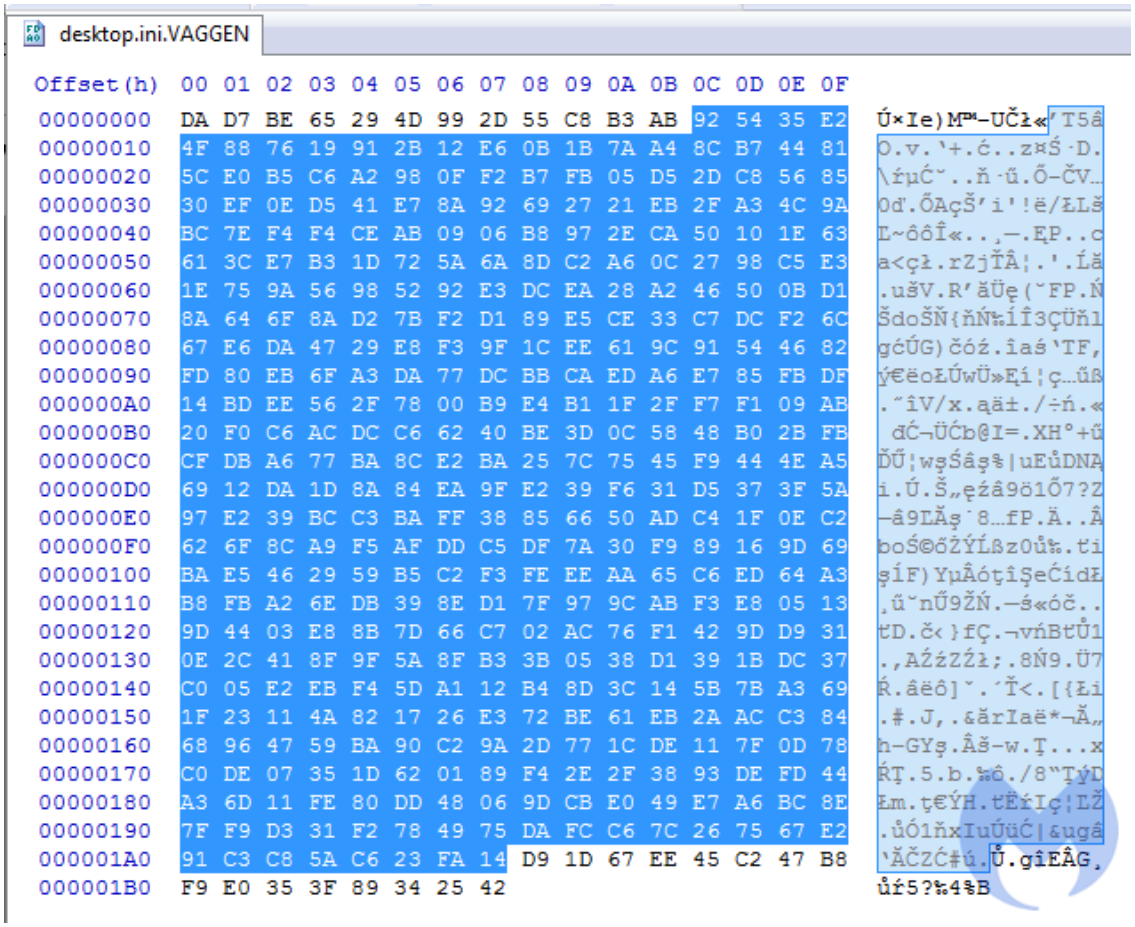
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

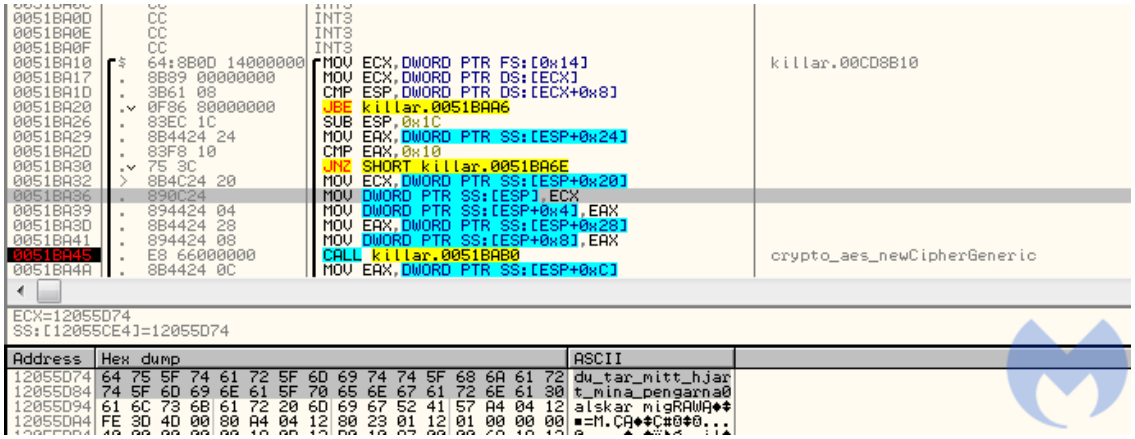


The content of the output file (with .VAGGEN extension) contains:

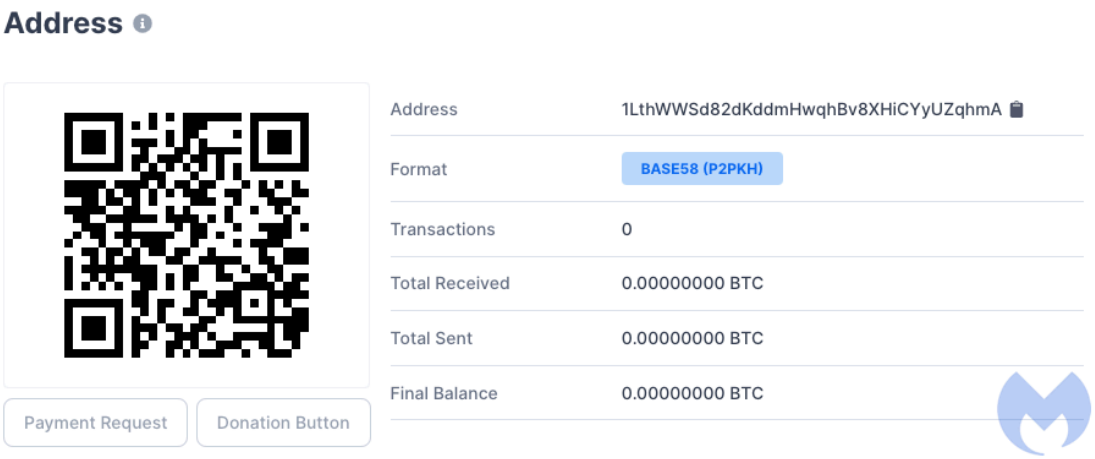
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

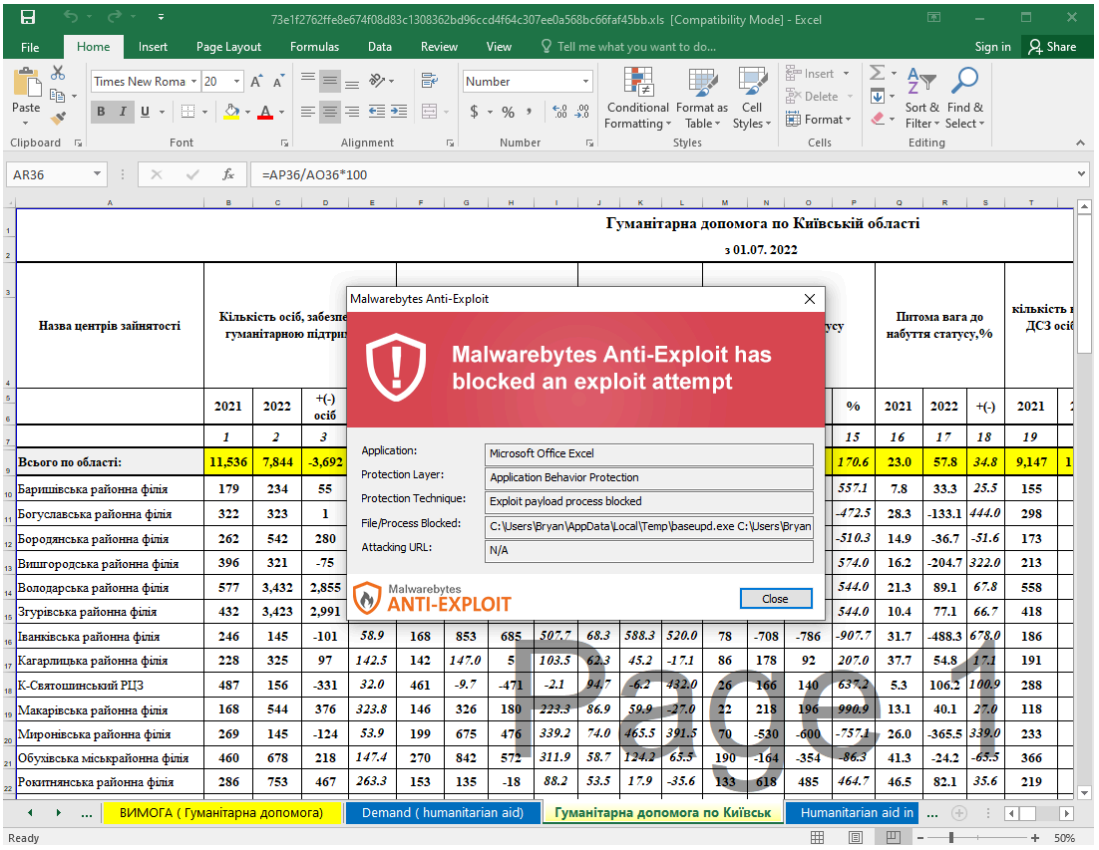
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

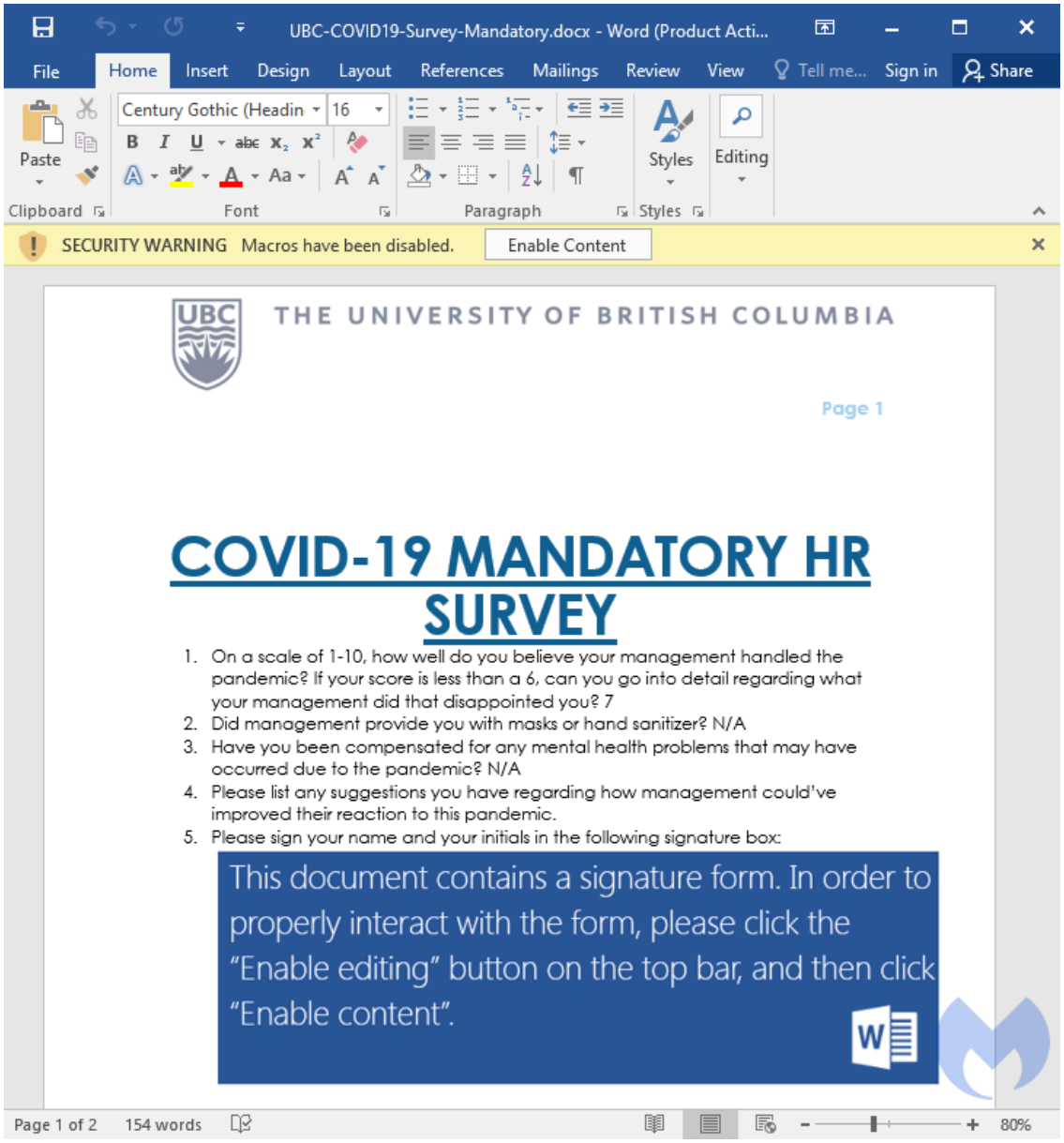
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

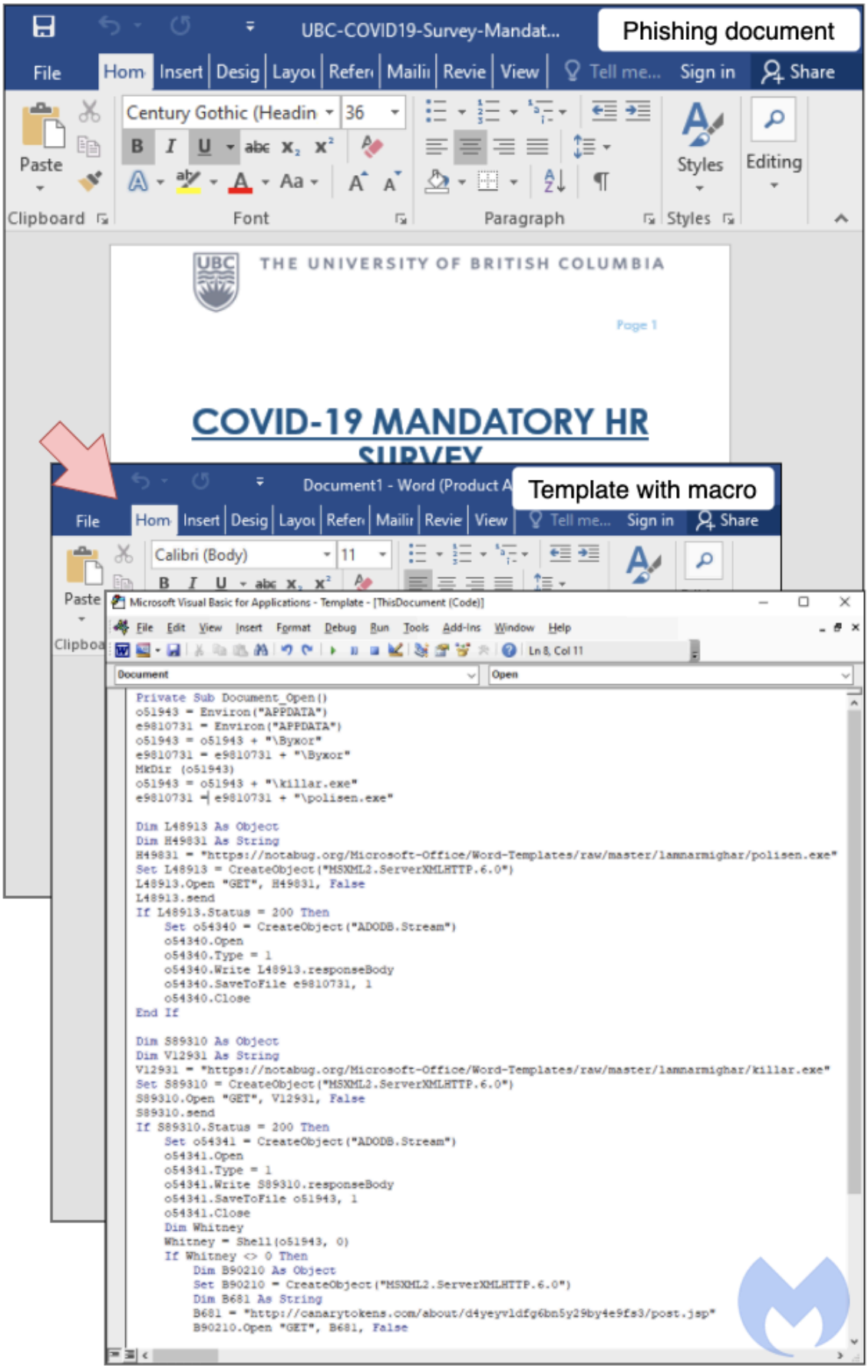
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

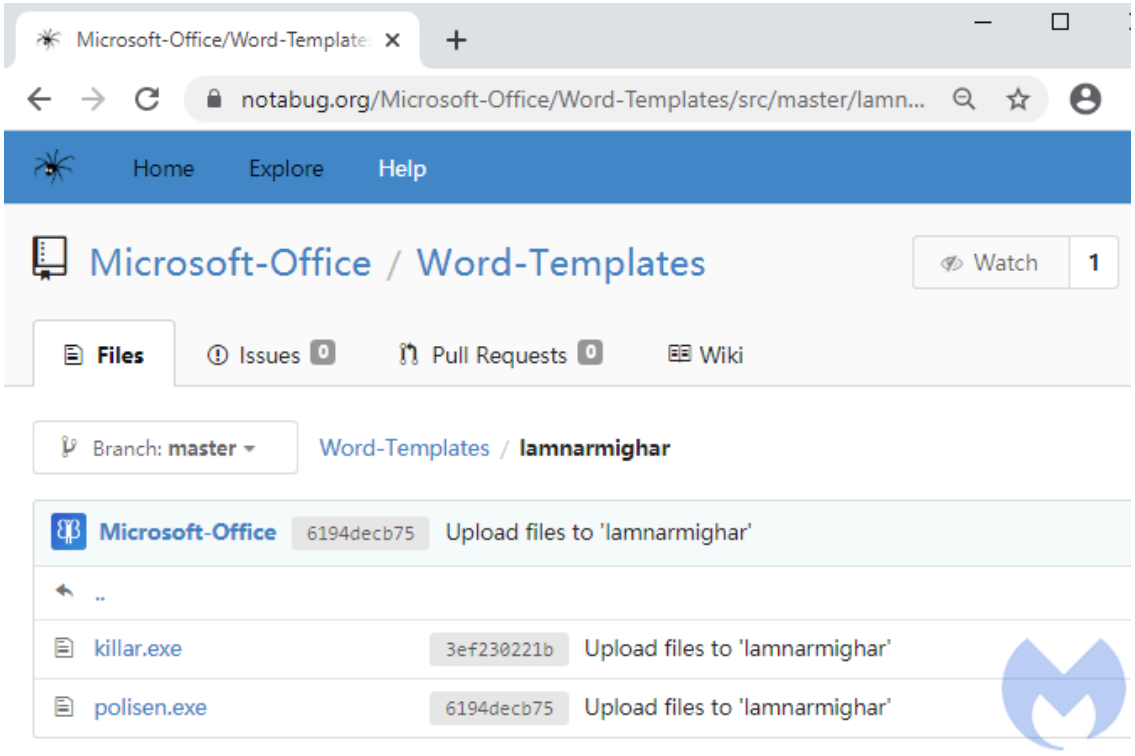
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



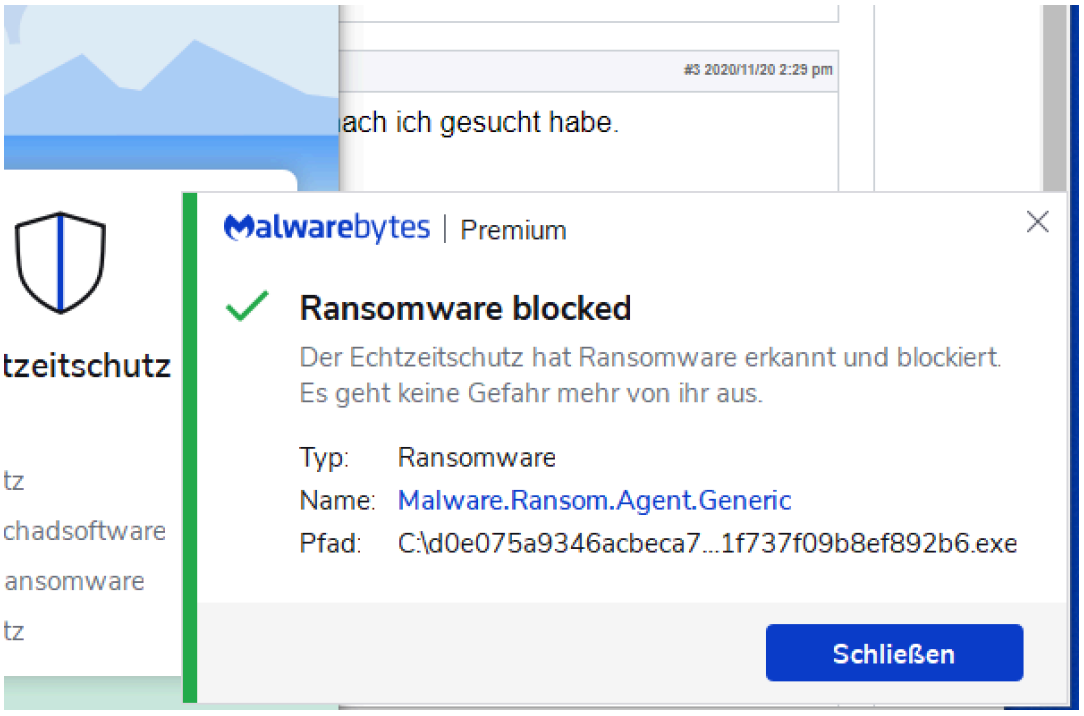
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

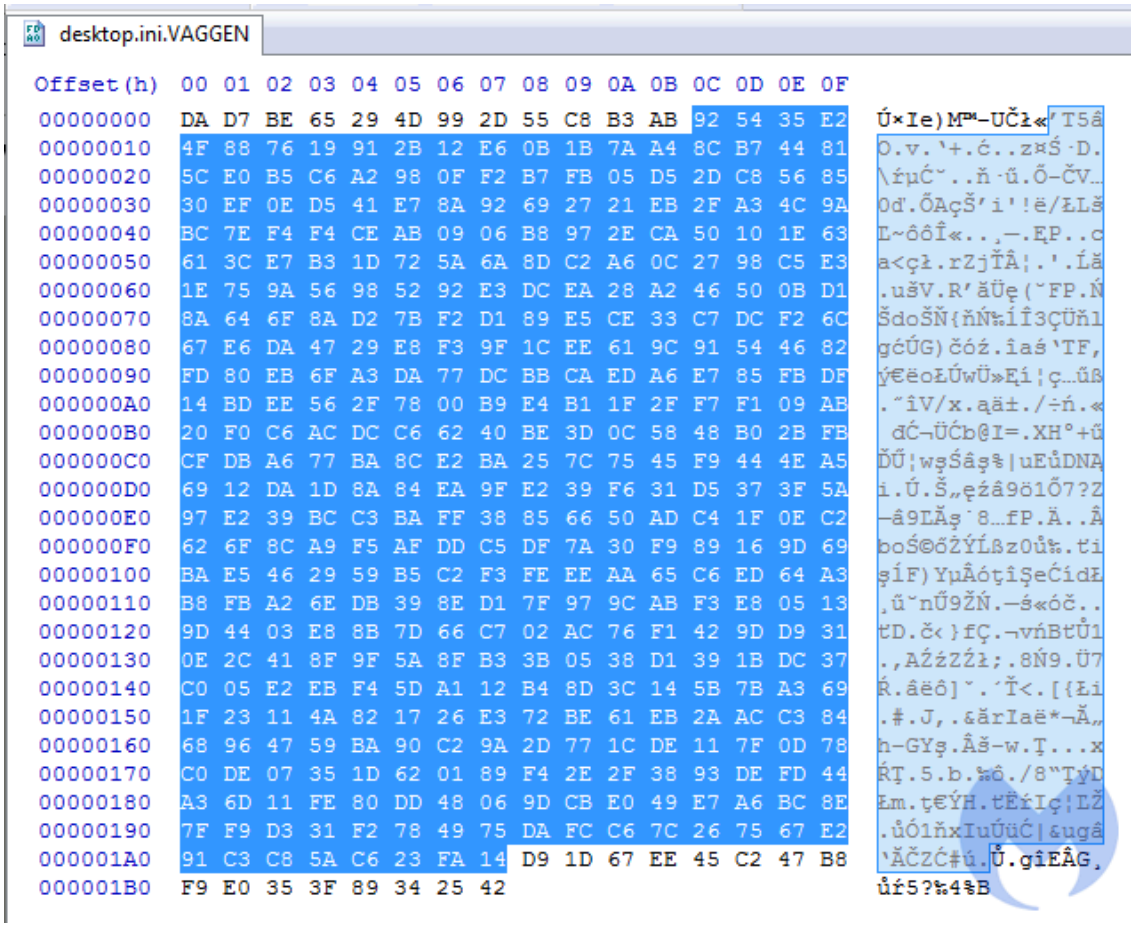
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

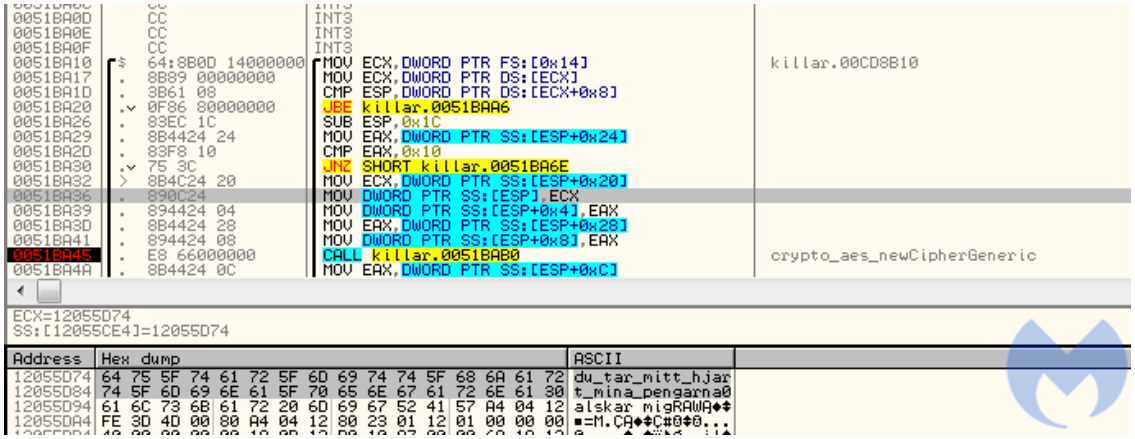


The content of the output file (with .VAGGEN extension) contains:

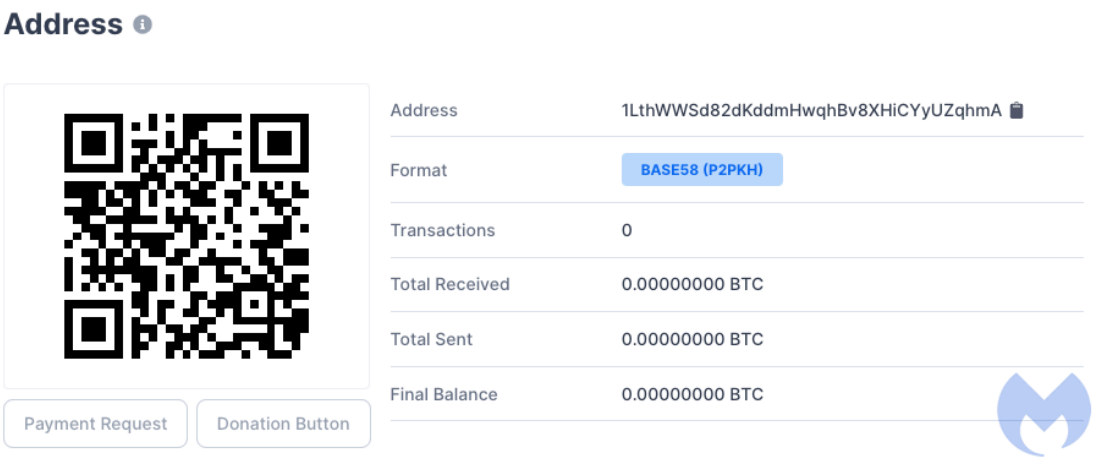
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

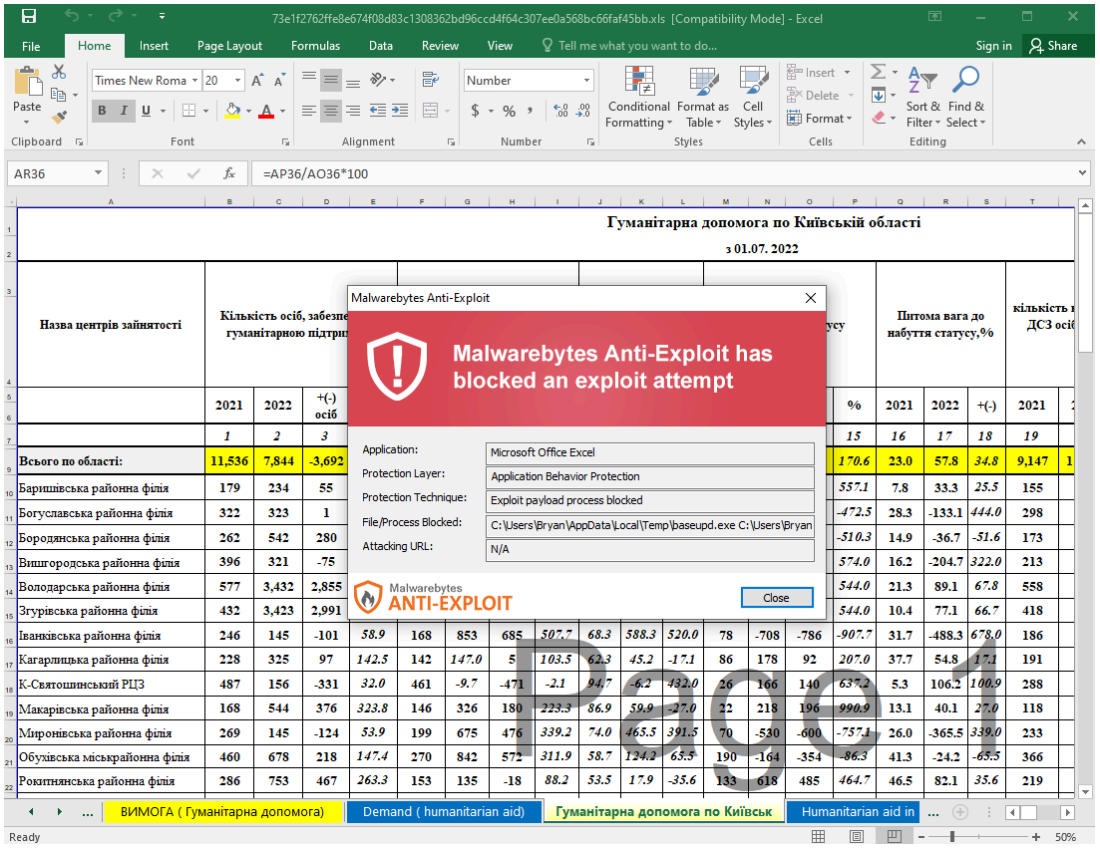
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

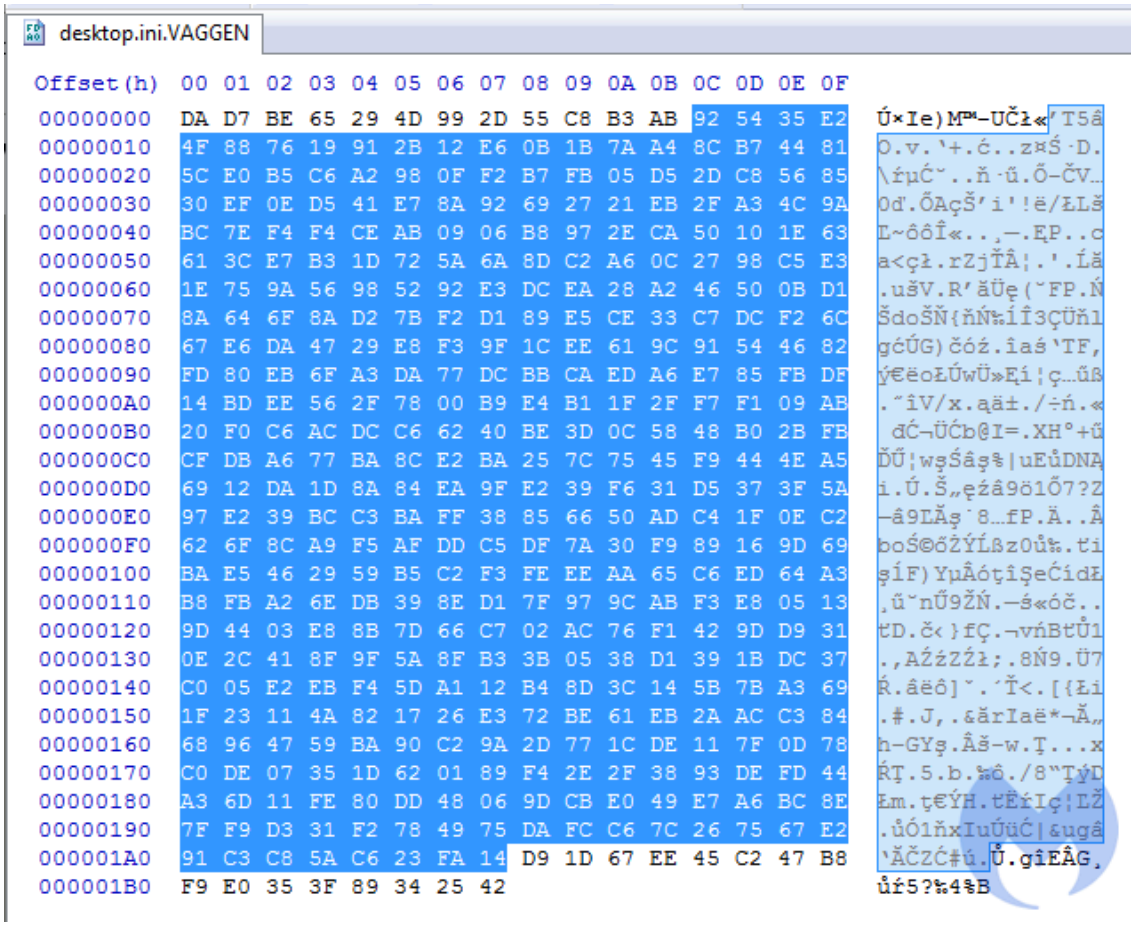
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

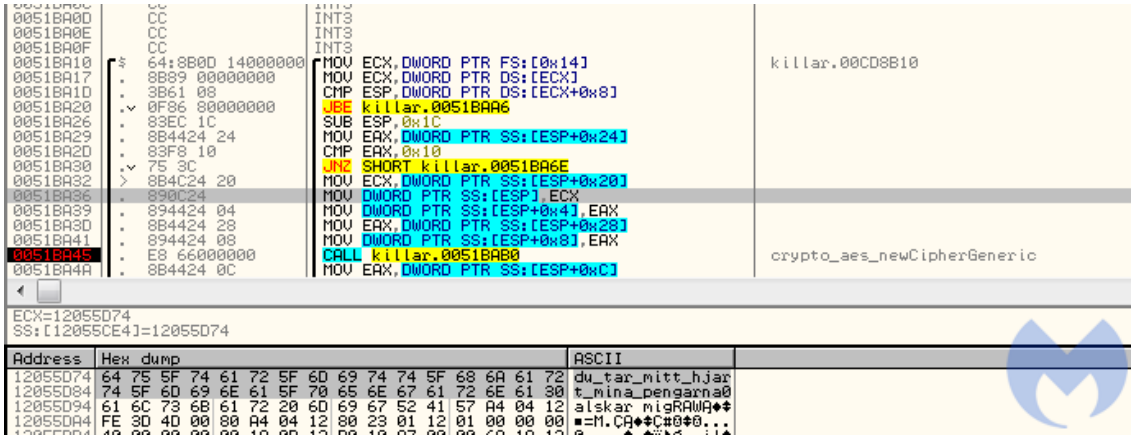


The content of the output file (with .VAGGEN extension) contains:

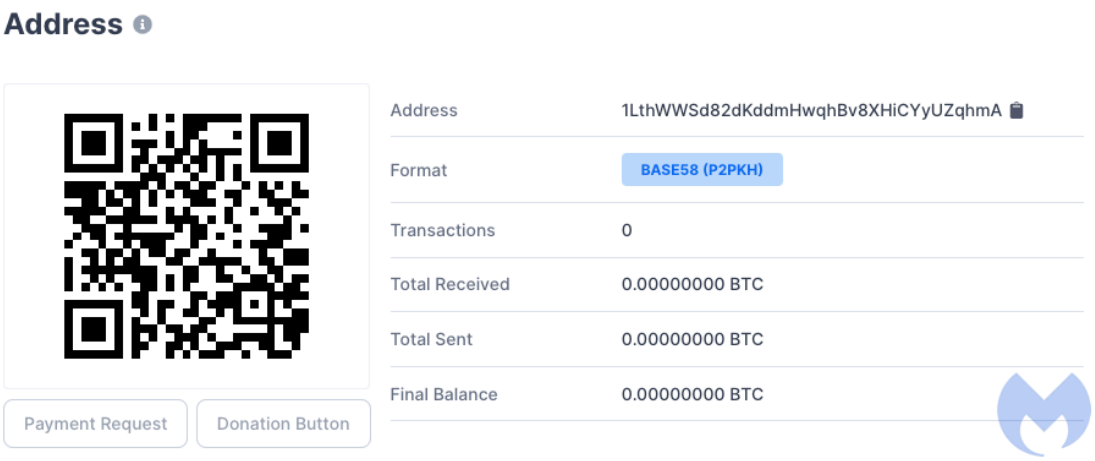
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

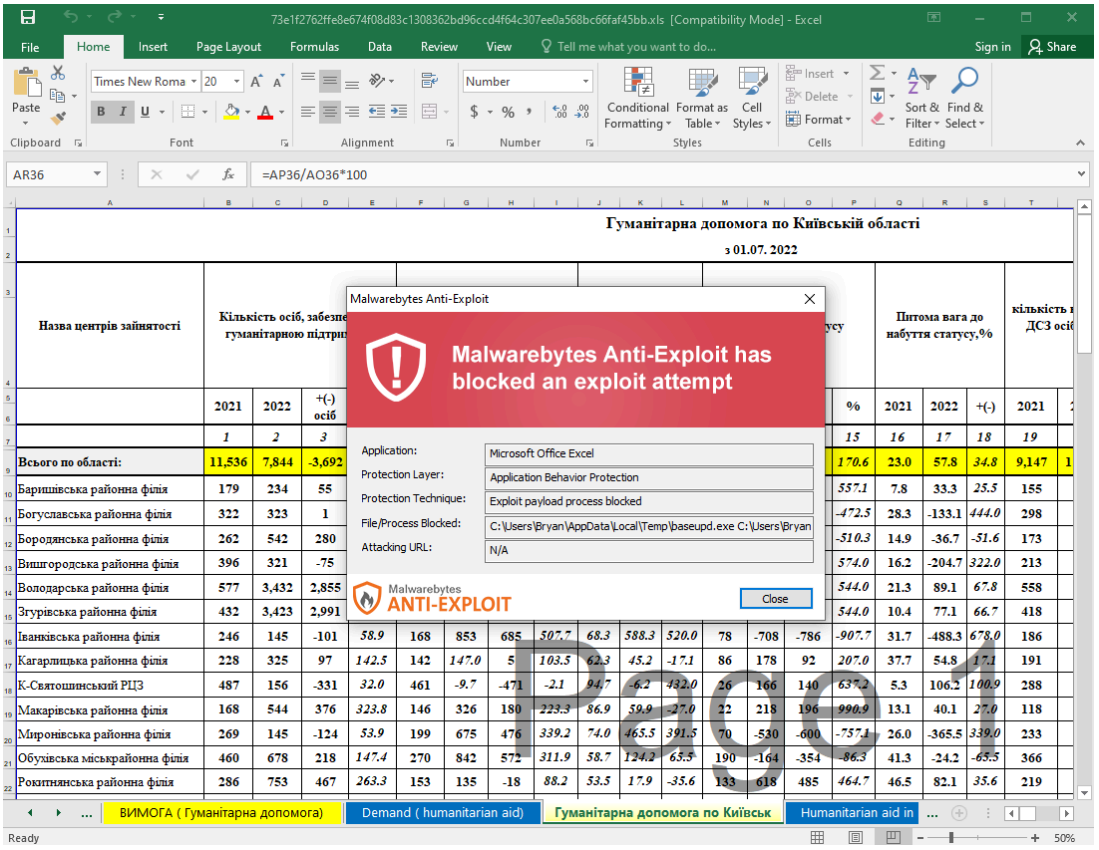
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
46  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
47    runtime_morestack_noctxt();
48  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
49  ((void (*)(void))loc_45CDA0)();
50  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
51  v38 = v4;
52  v30 = v5;
53  v31 = v6;
54  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
55  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
56  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
57  v32 = v10;
58  v11 = *(void (__cdecl **)(int))(v10 + 16);
59  v39 = v12;
60  v11(v12);
61  v29 = v13;
62  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
63  v40 = v15;
64  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
65  v17 = v27;
66  if ( v27 )
67  {
68    if ( v27 )
69      v17 = *(_DWORD *)(v27 + 4);
70    v43 = v17;
71    v44 = v28;
72    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
73  }
```

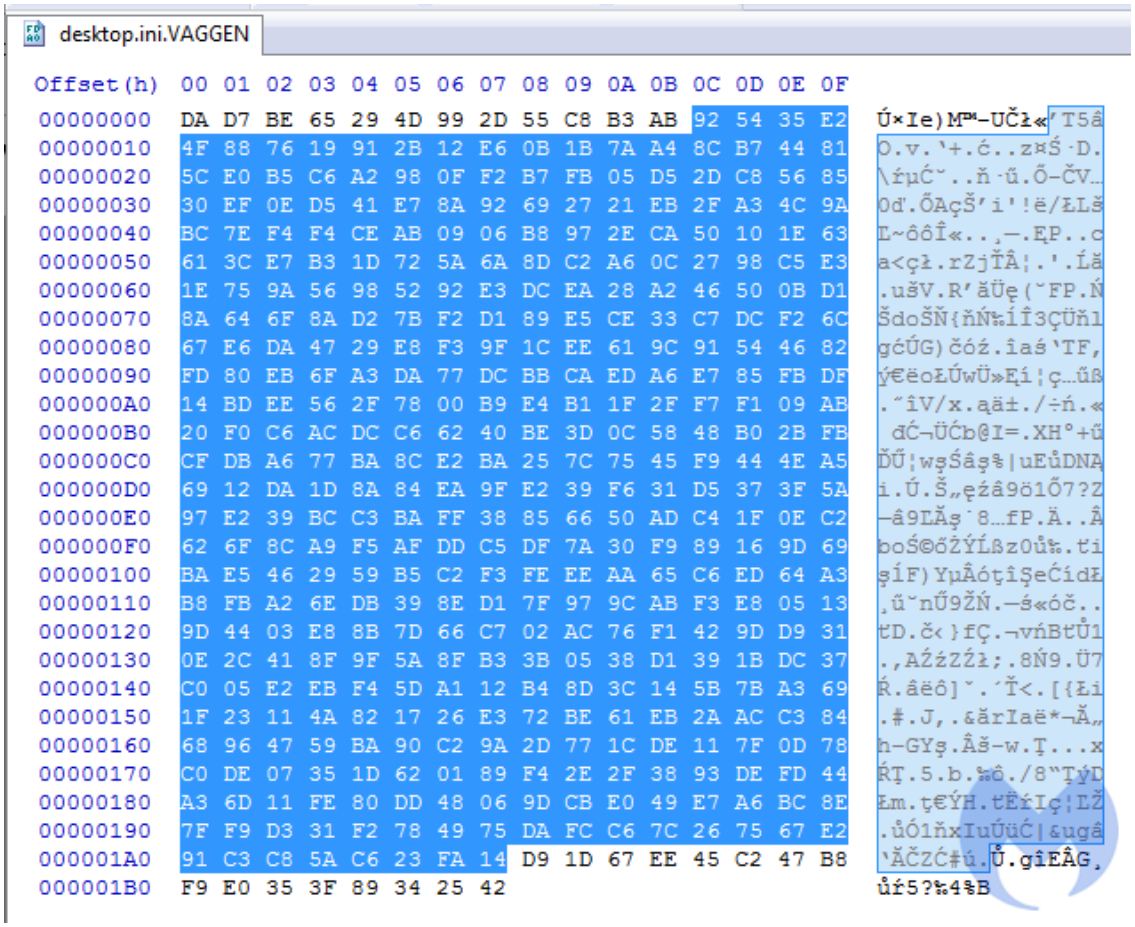


The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

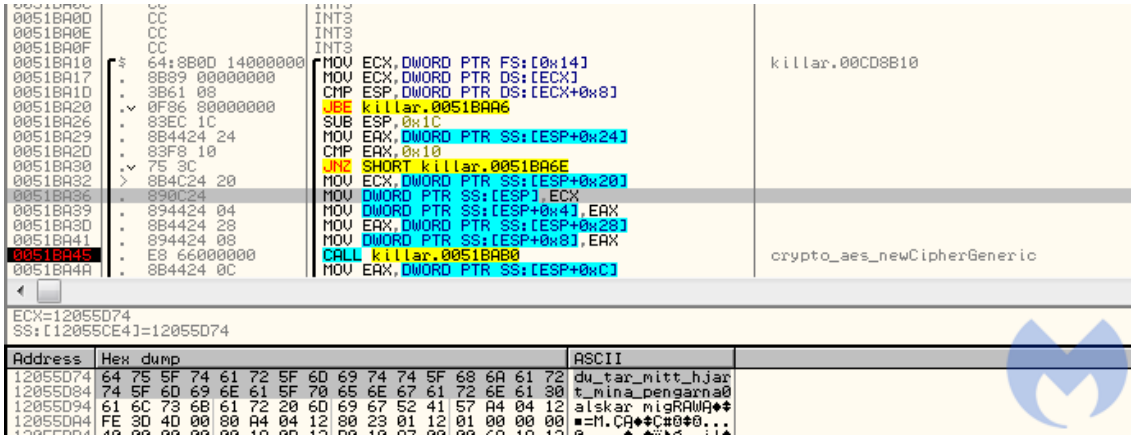


The content of the output file (with .VAGGEN extension) contains:

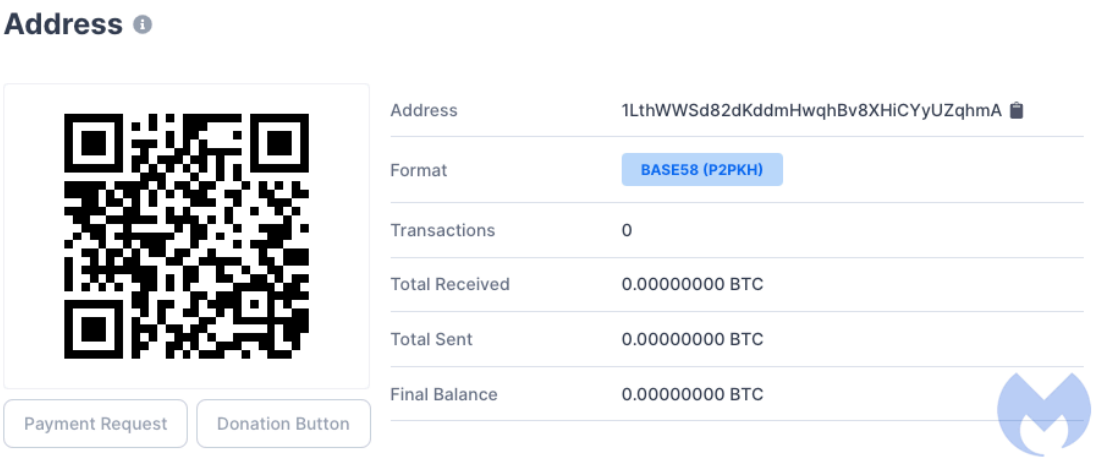
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

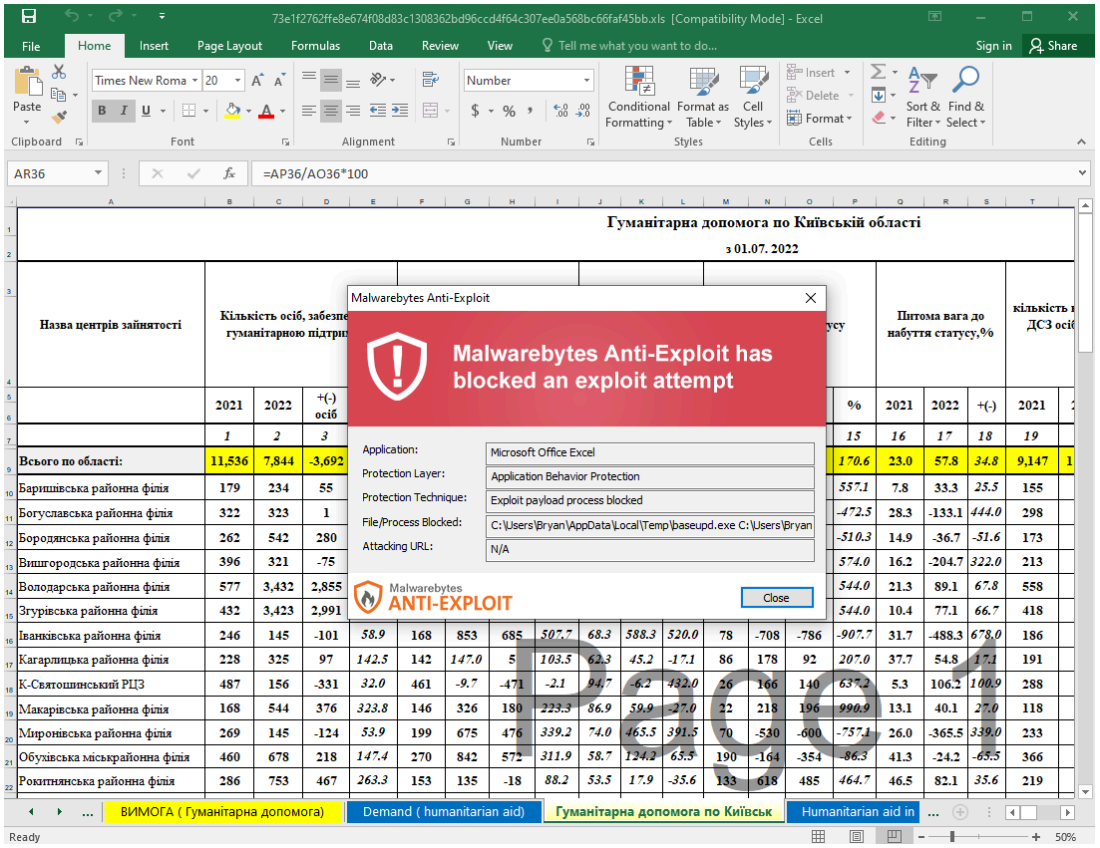
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

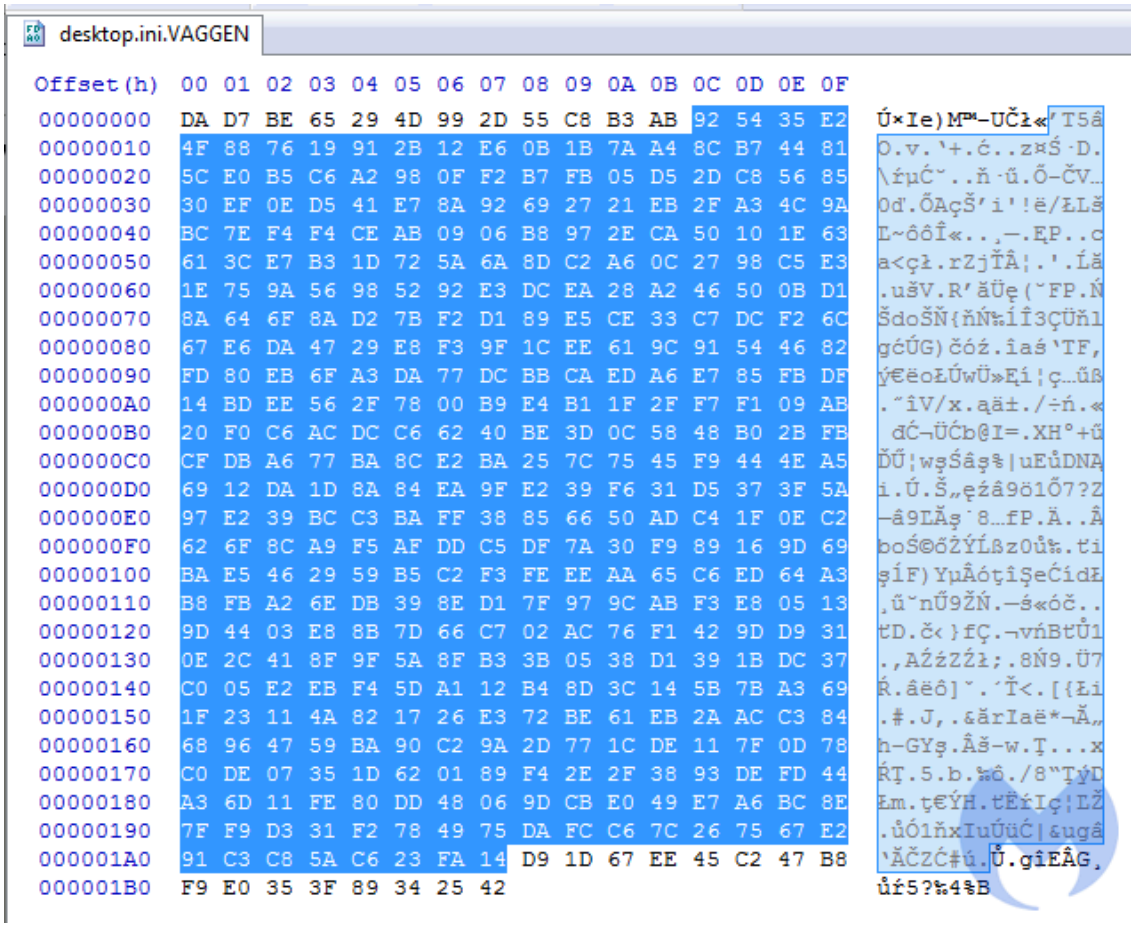
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

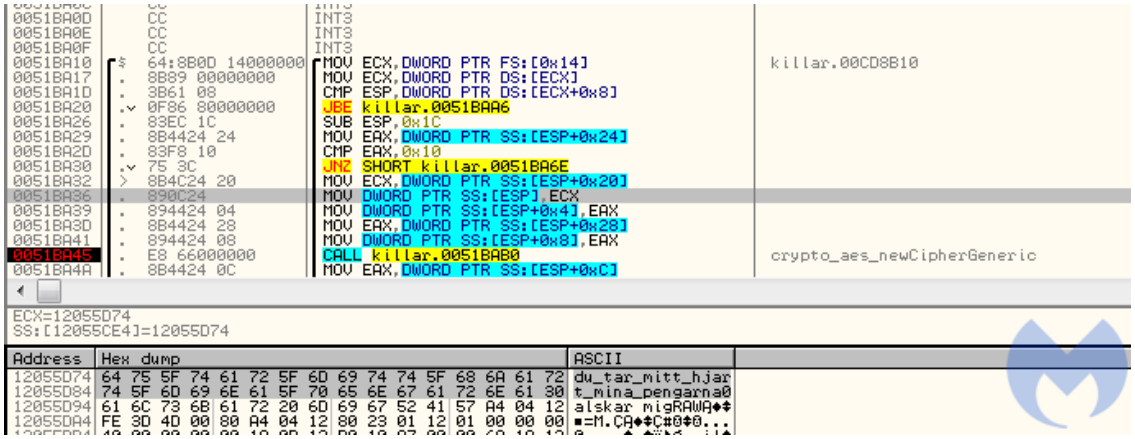


The content of the output file (with .VAGGEN extension) contains:

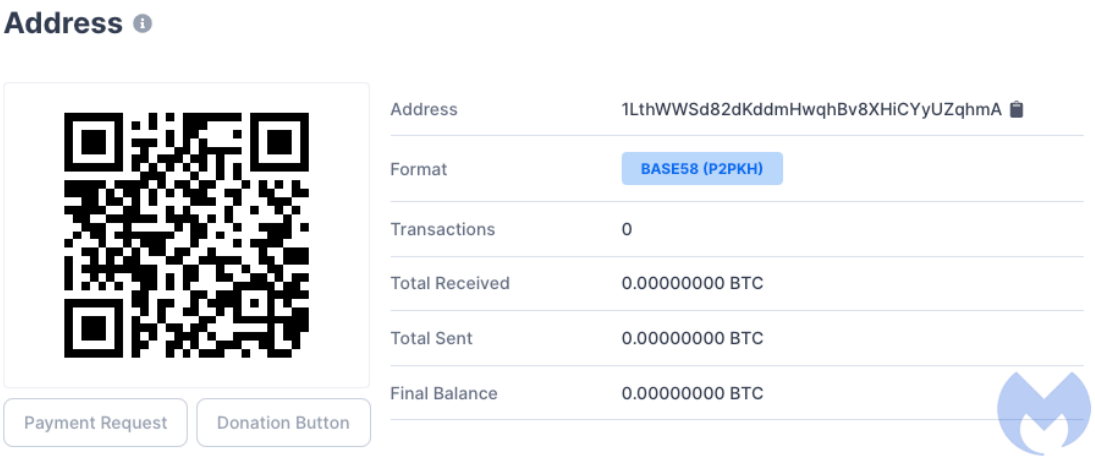
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

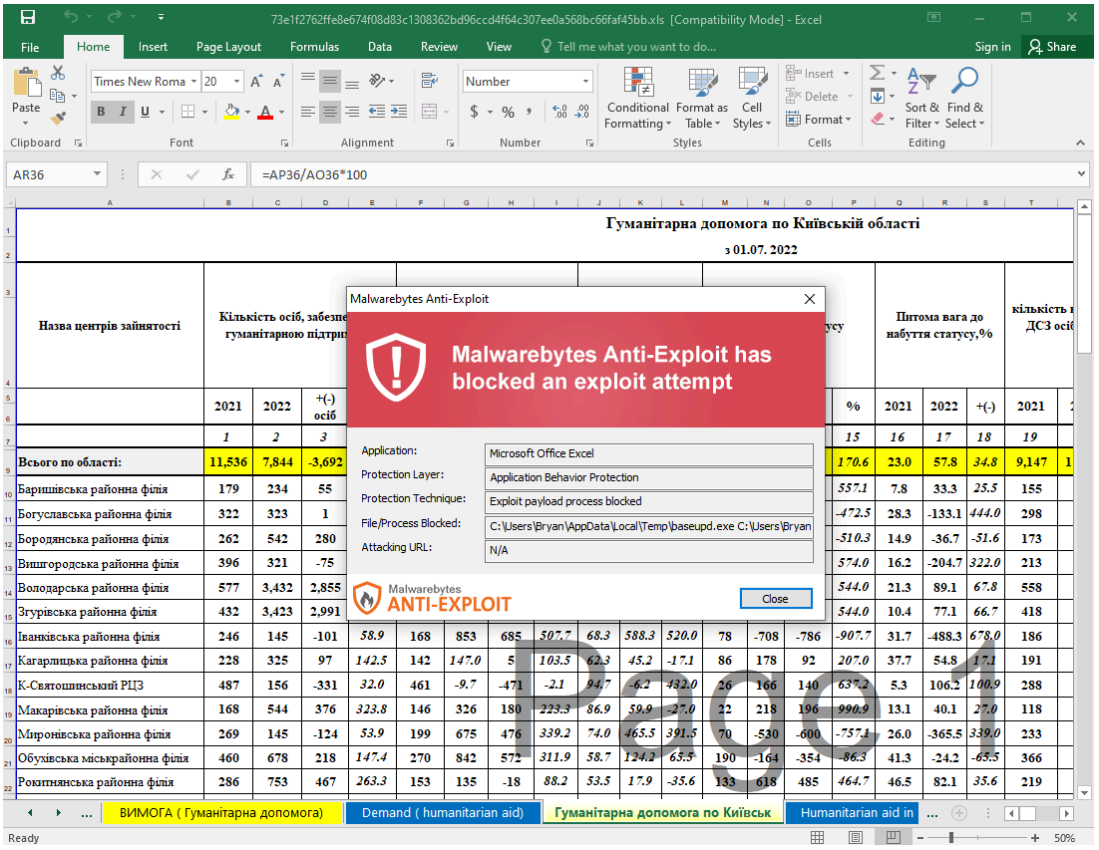
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

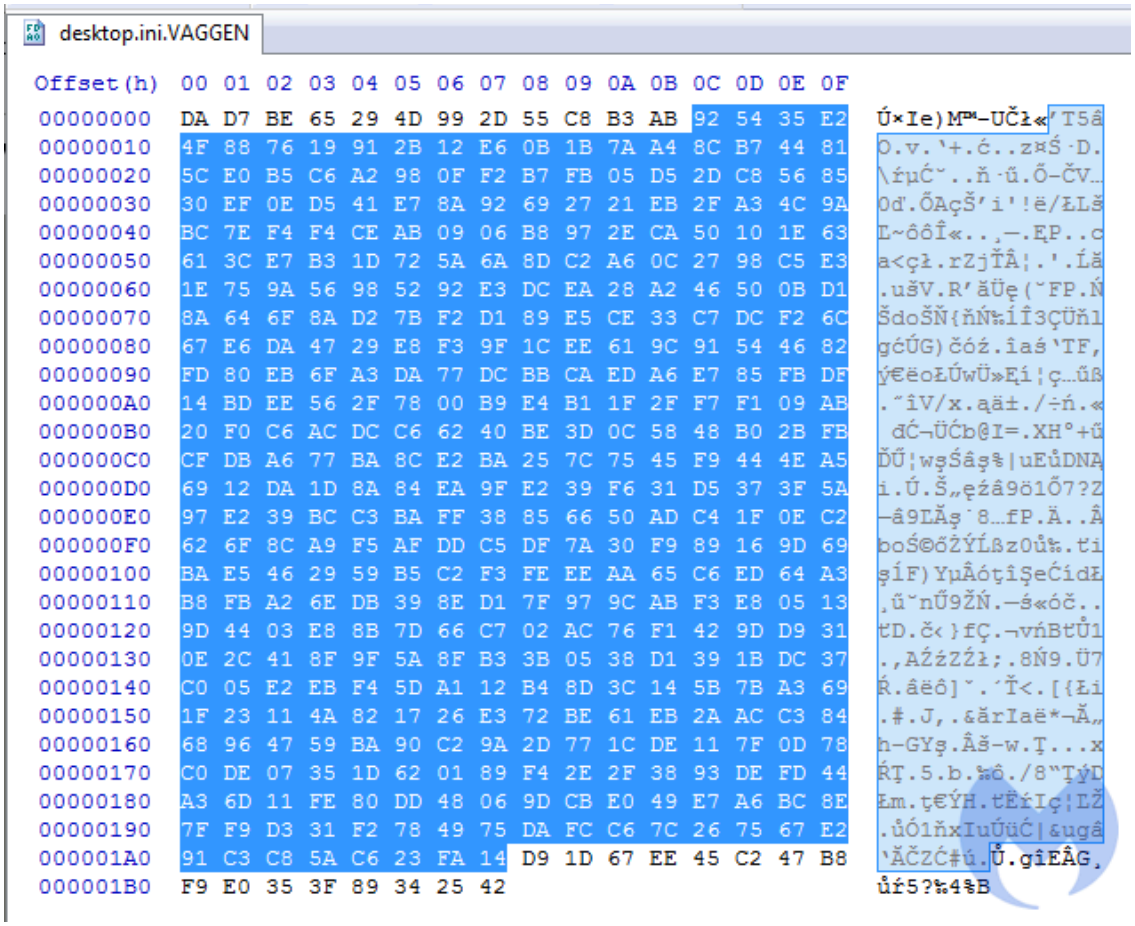
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



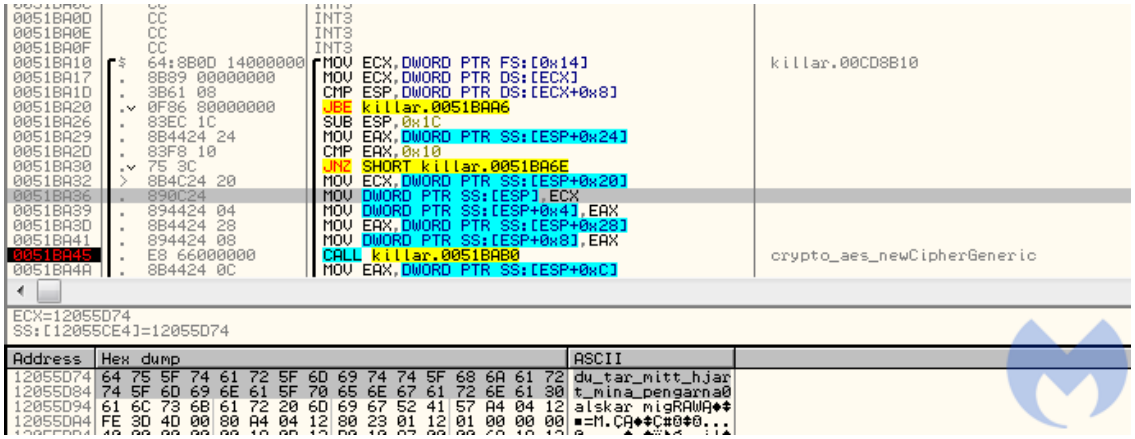
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

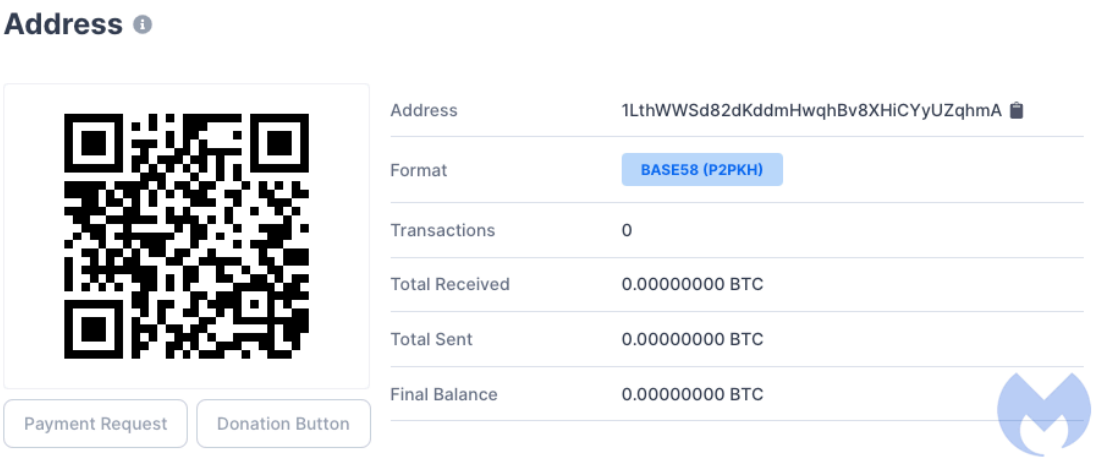
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

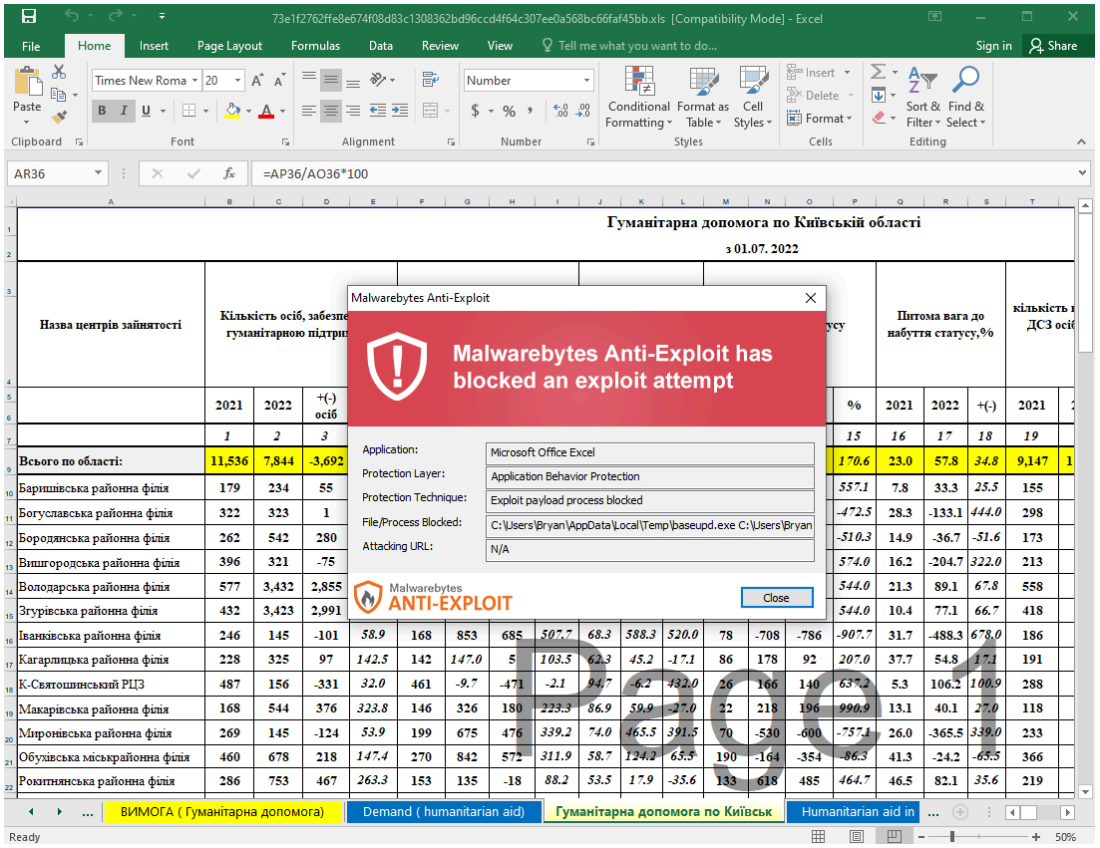
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

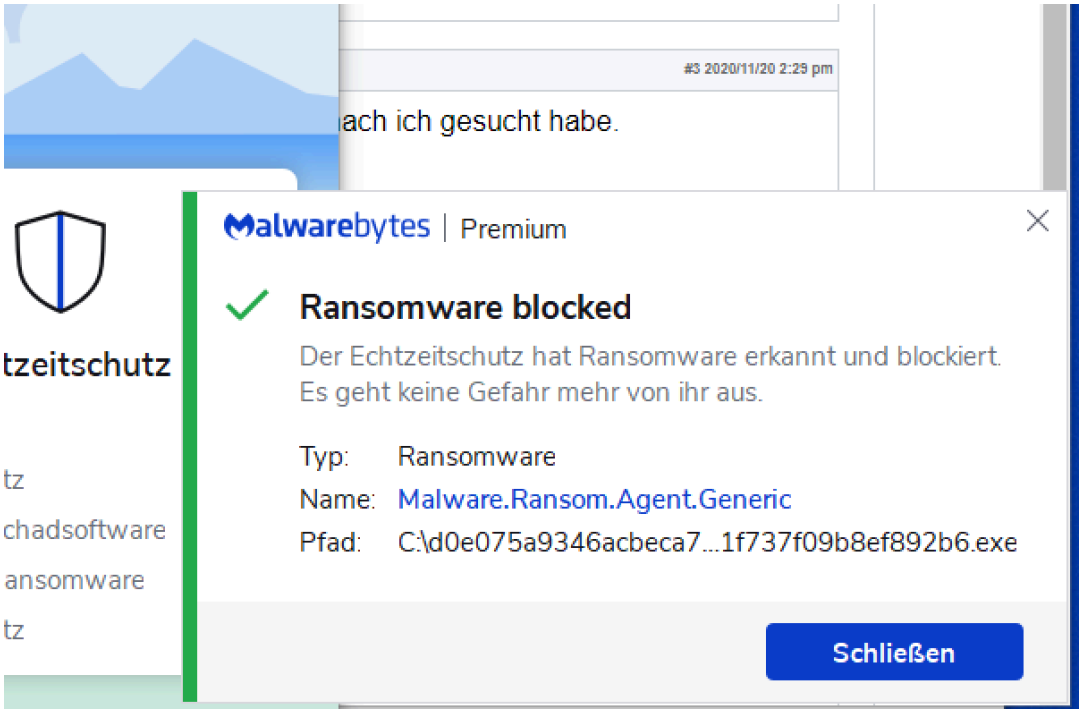
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

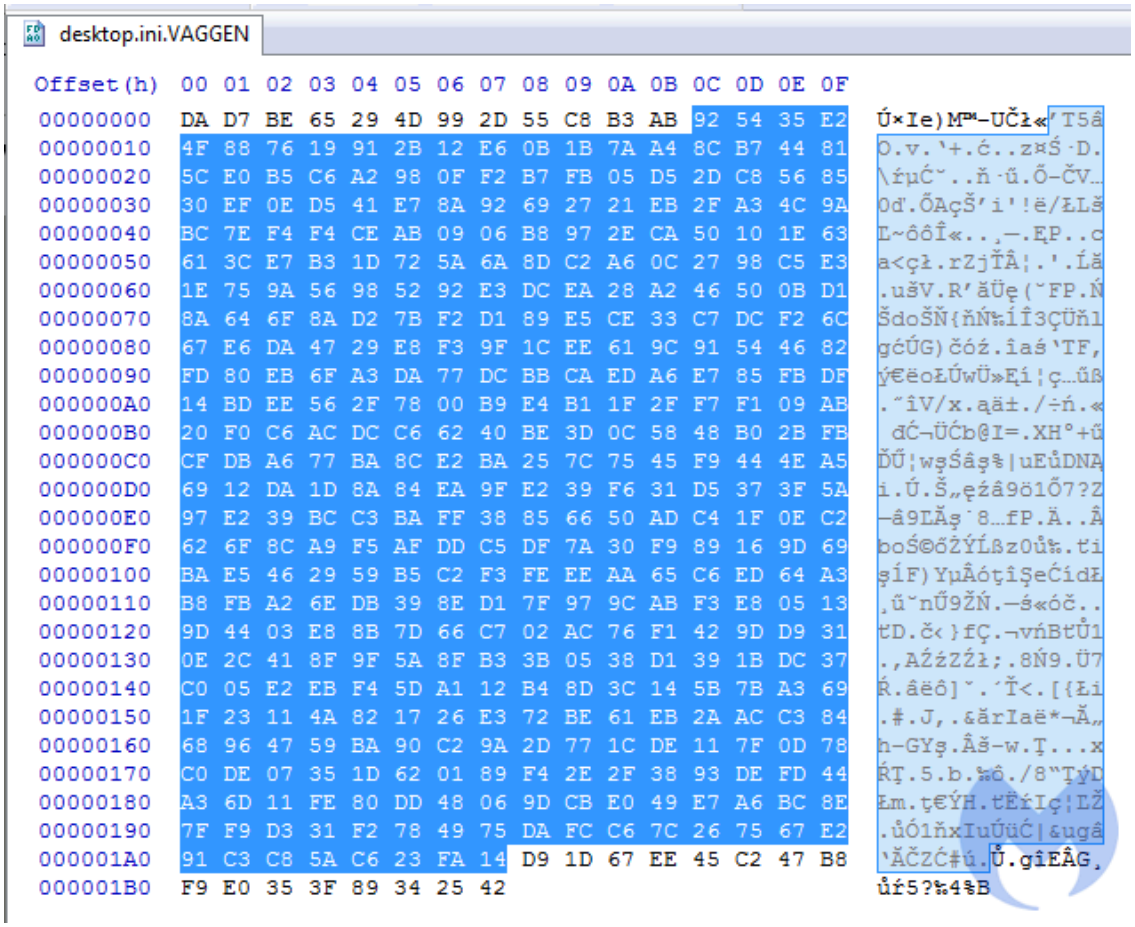
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

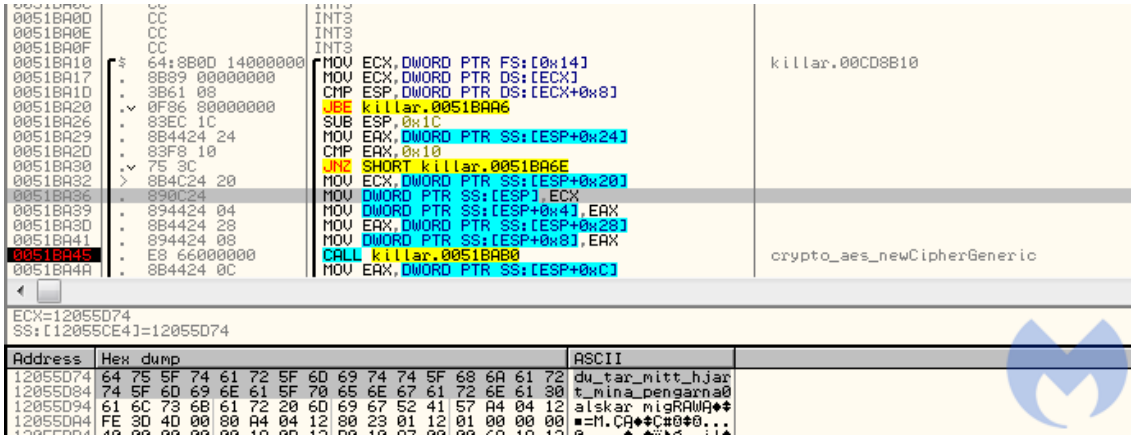


The content of the output file (with .VAGGEN extension) contains:

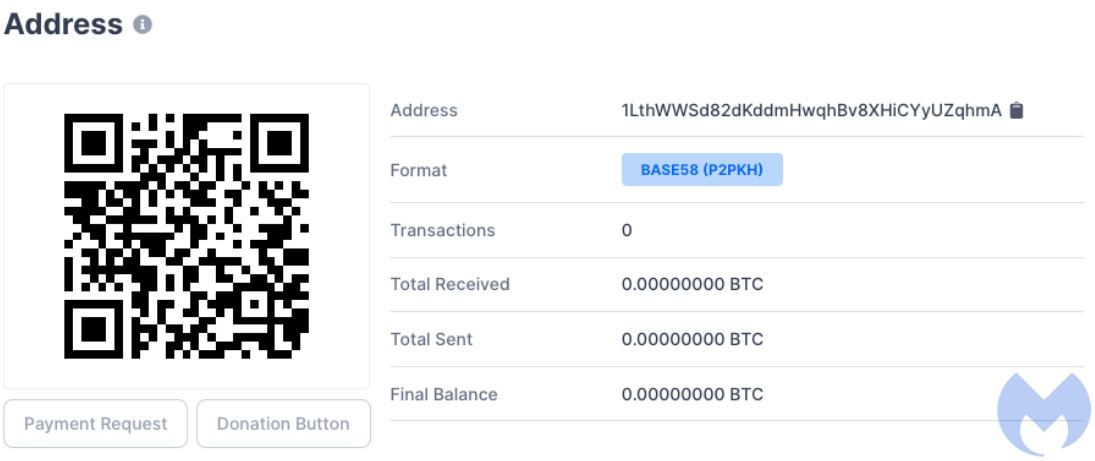
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

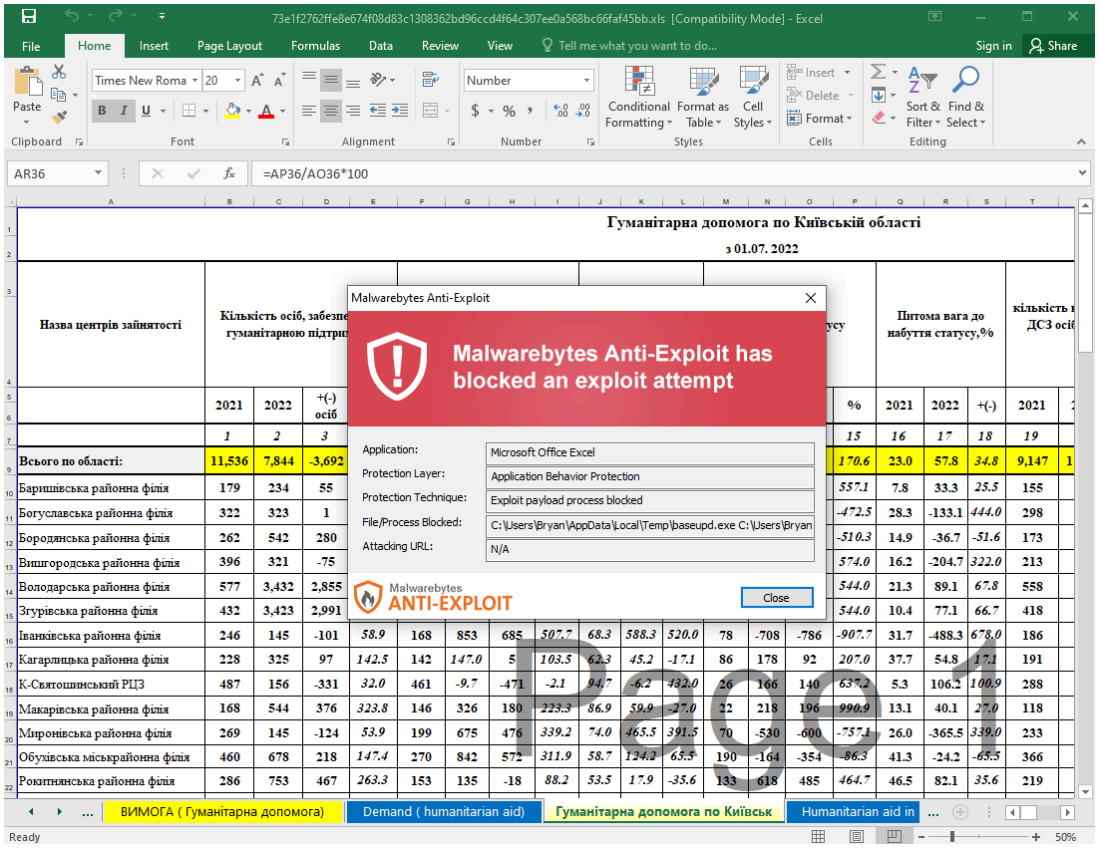
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

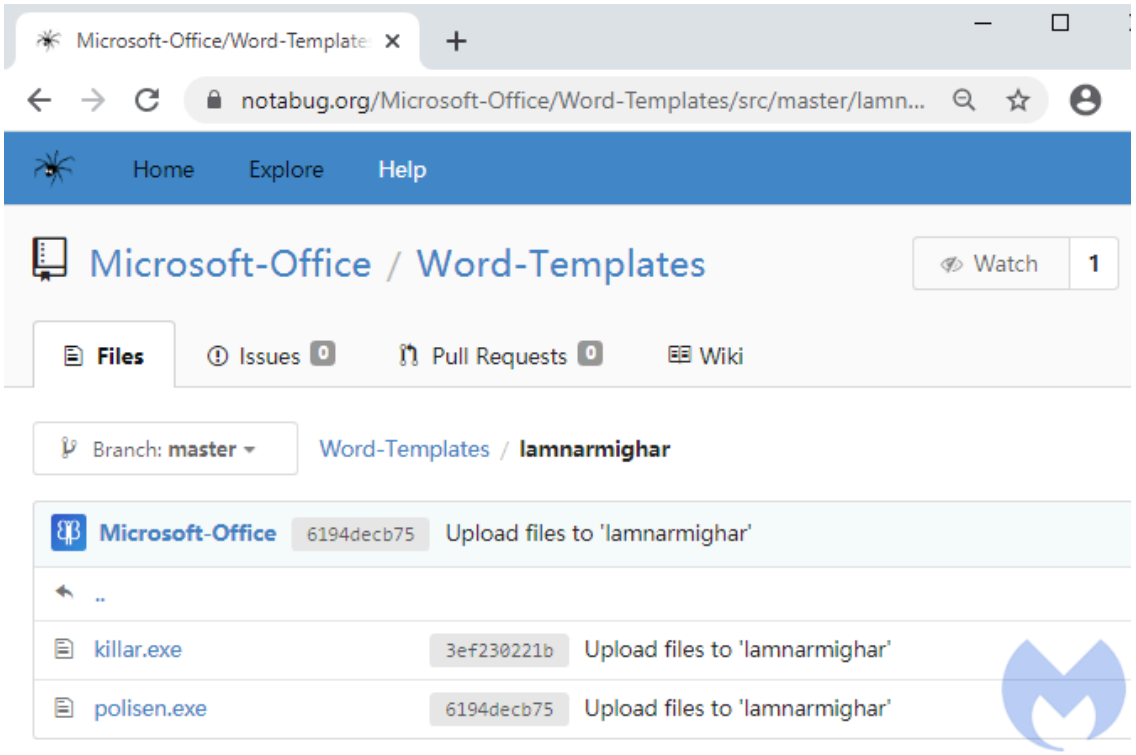
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



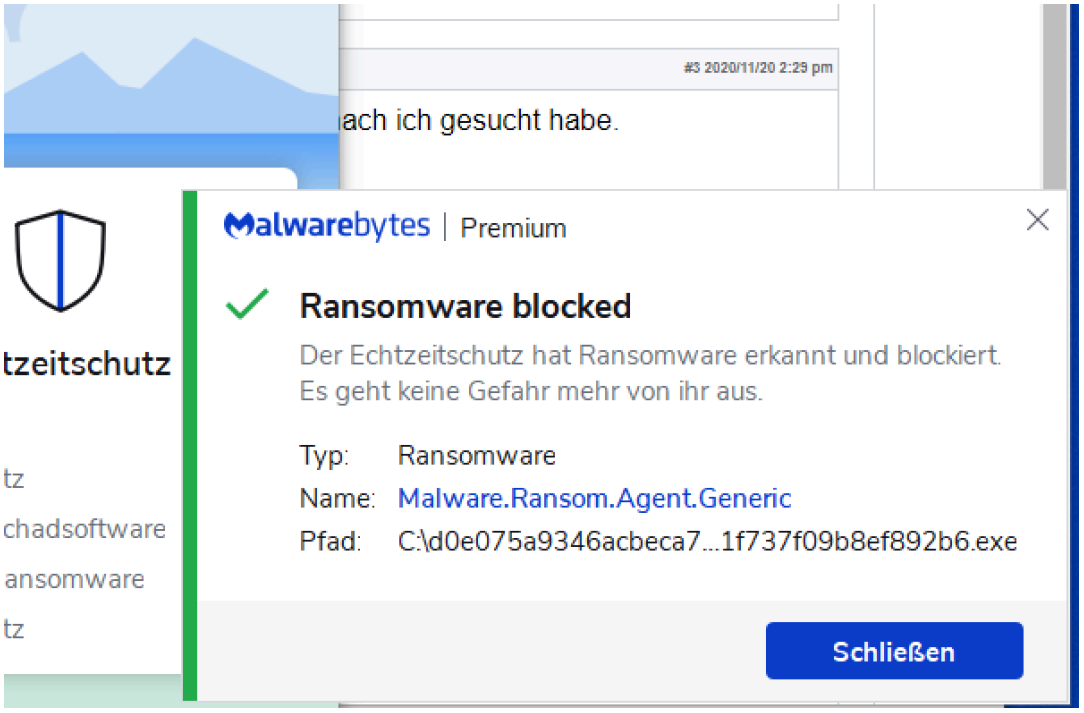
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

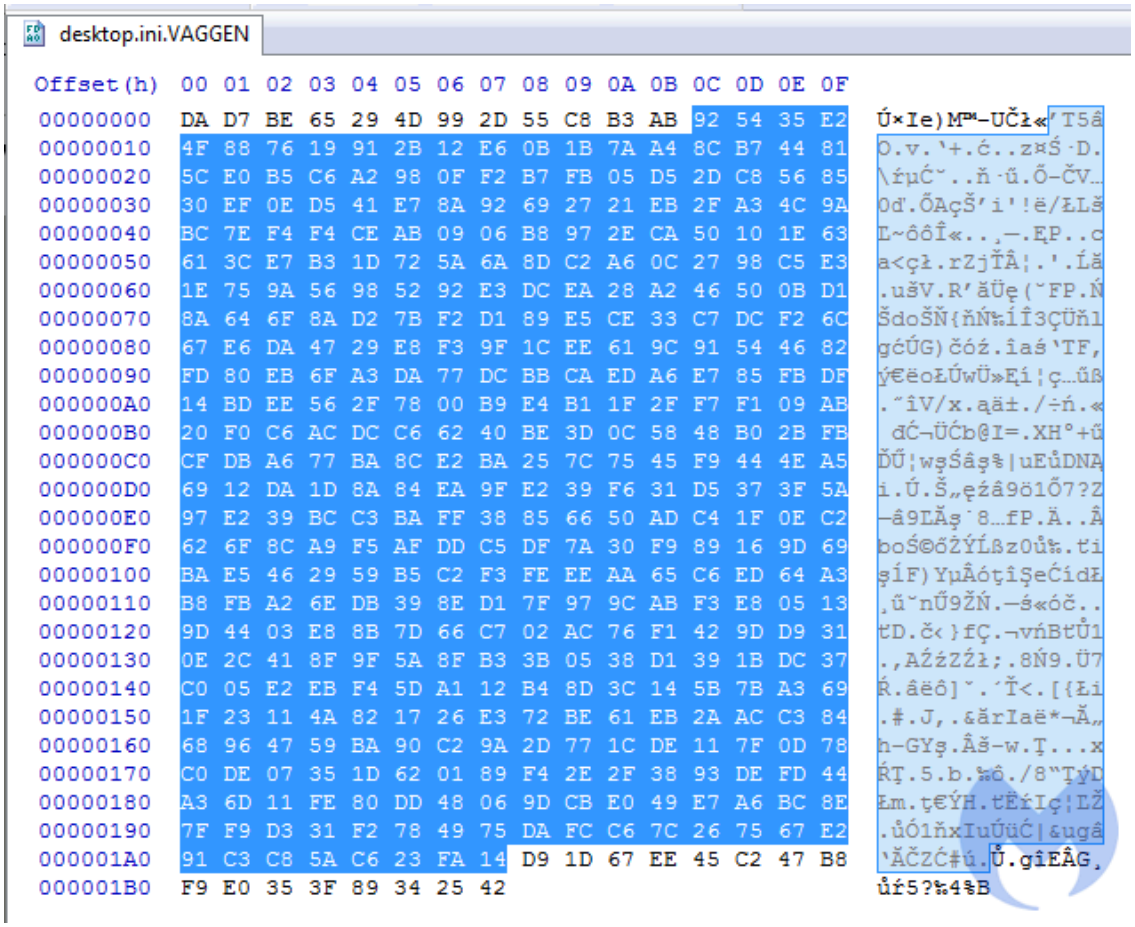
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

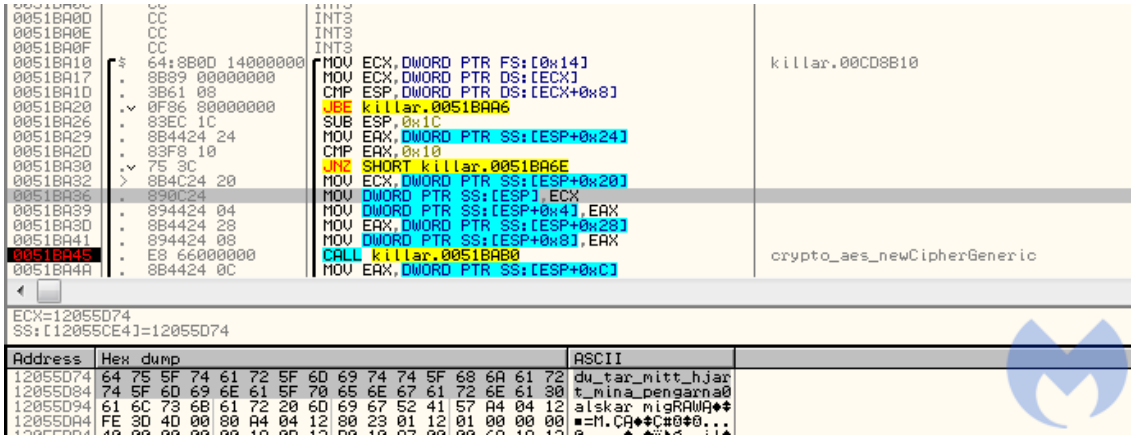


The content of the output file (with .VAGGEN extension) contains:

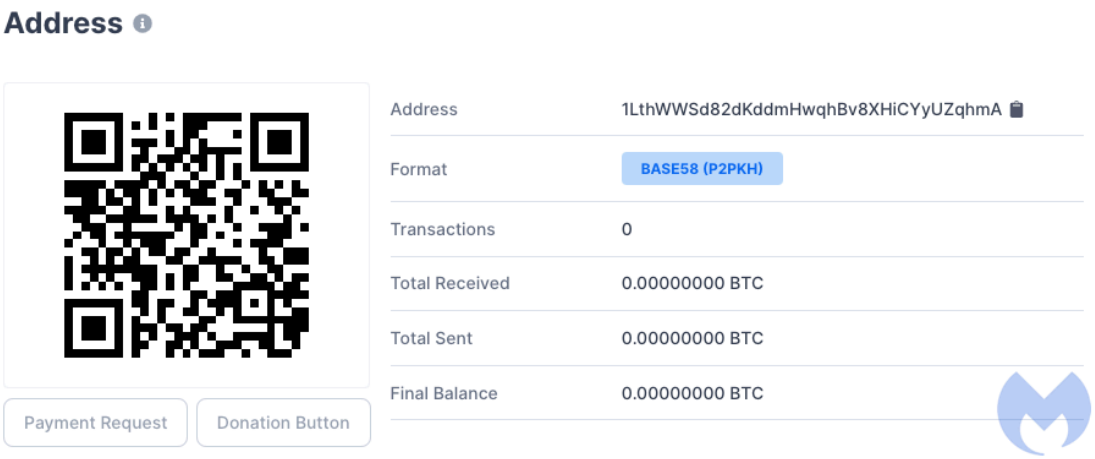
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

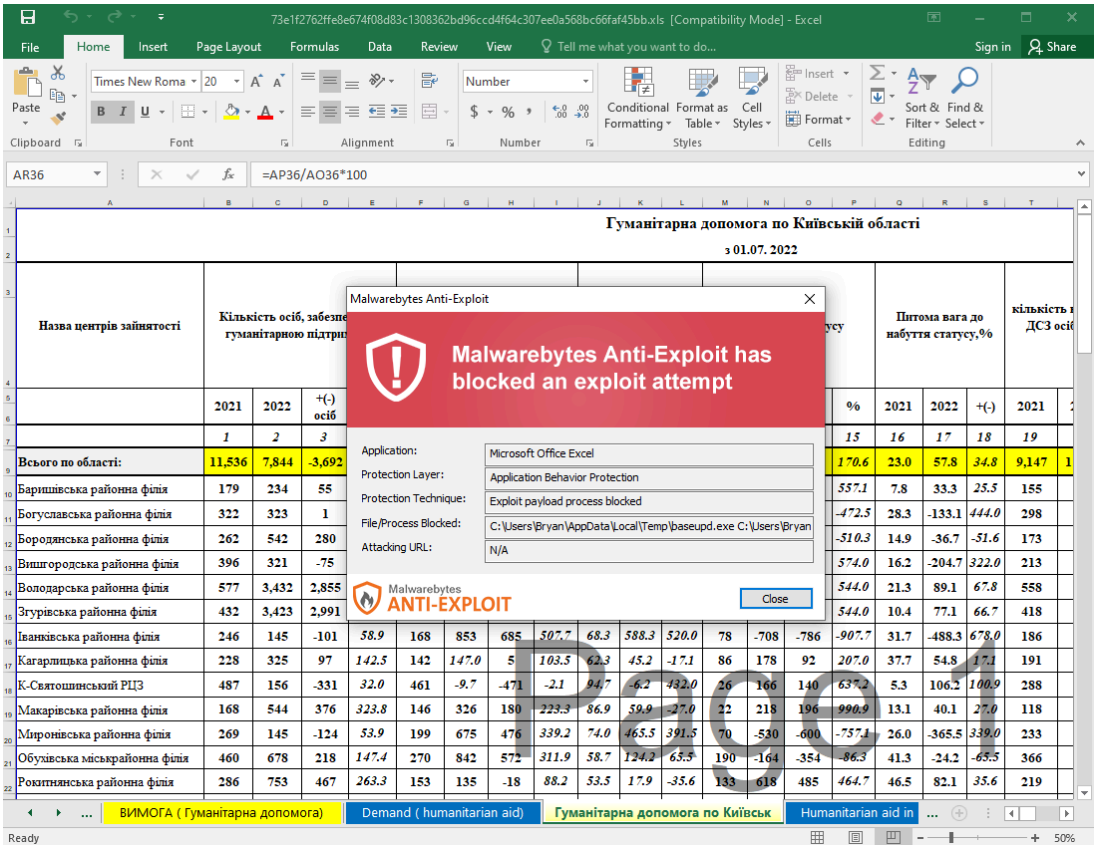
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

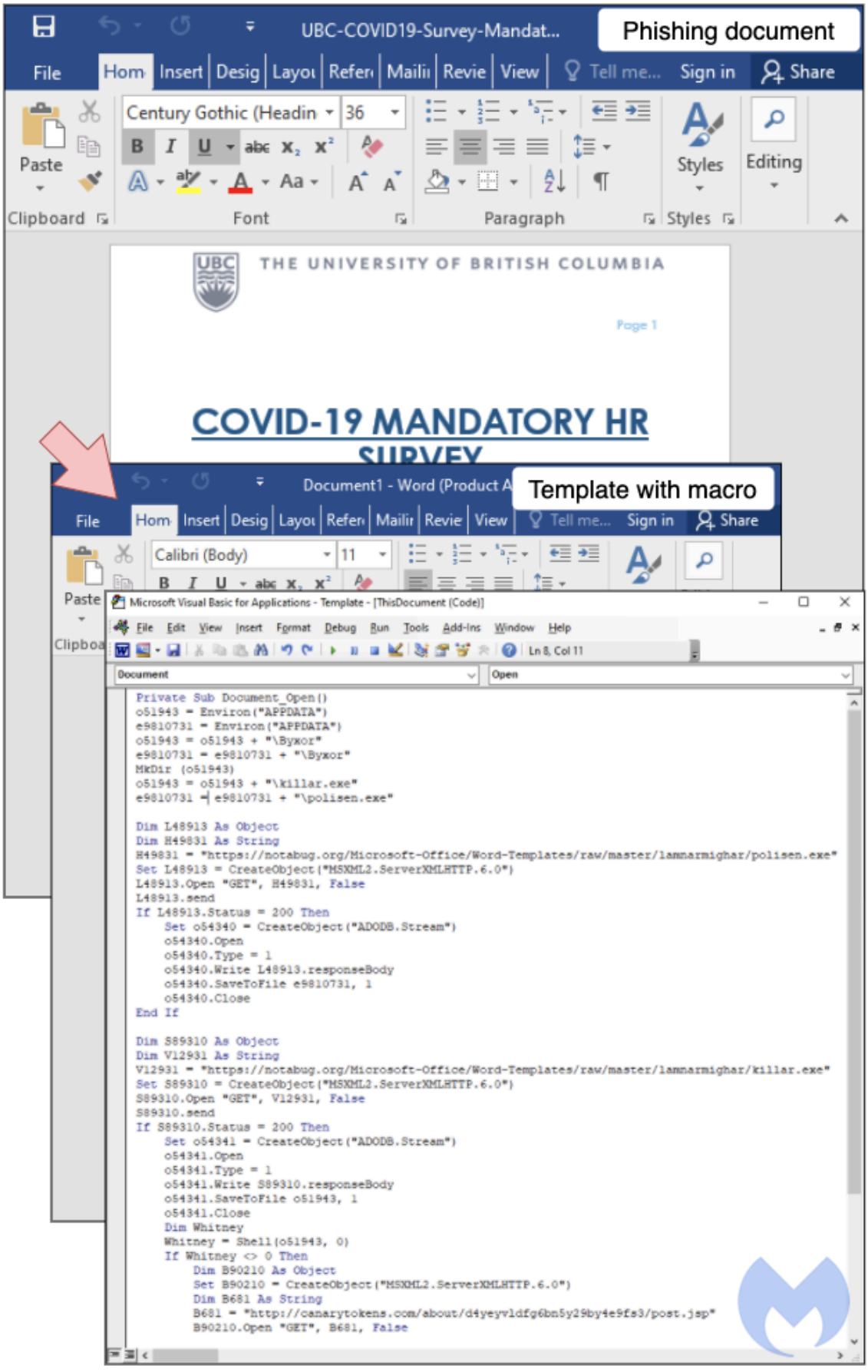
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

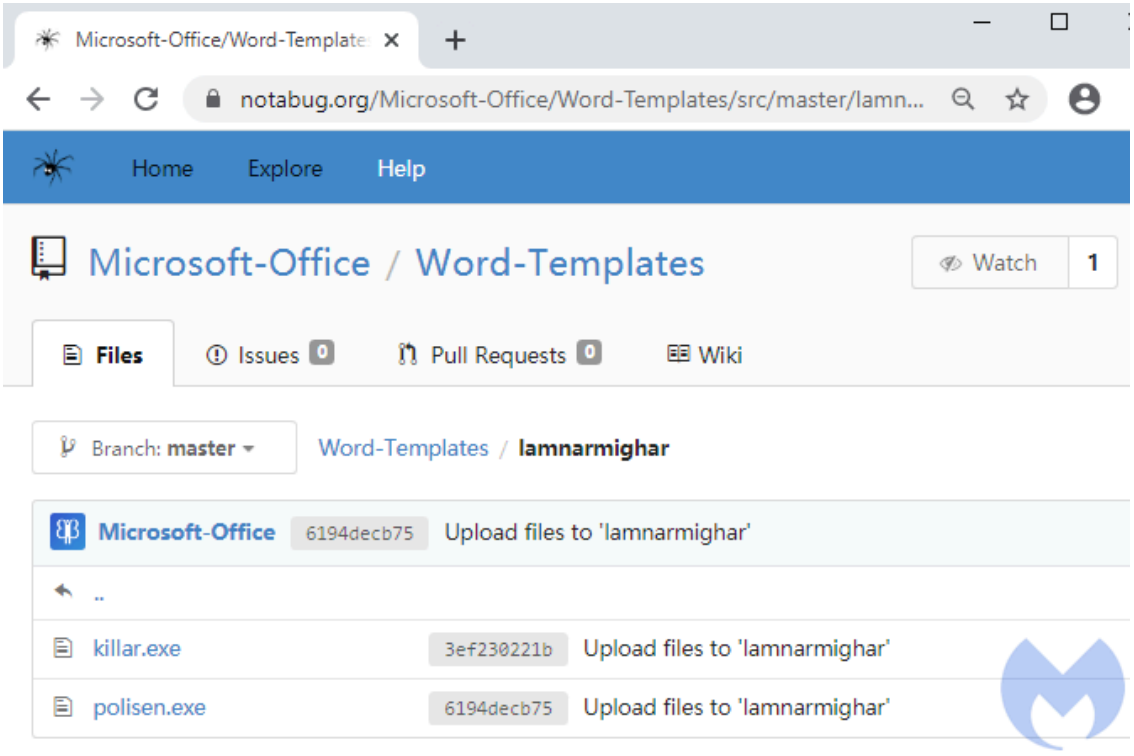
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



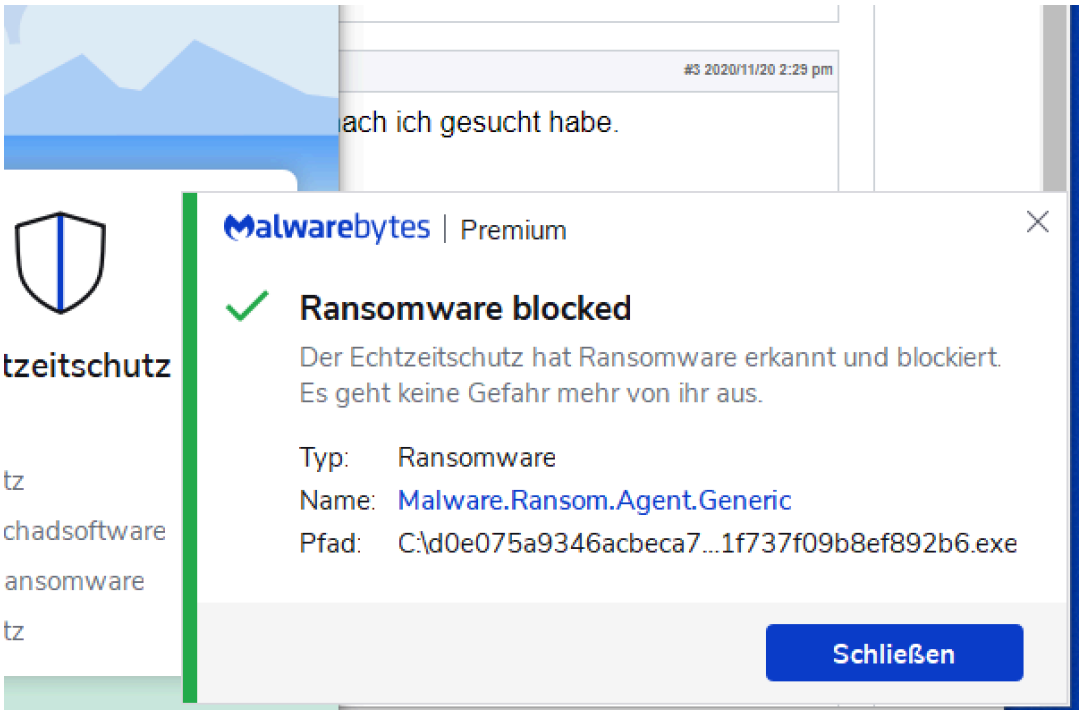
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

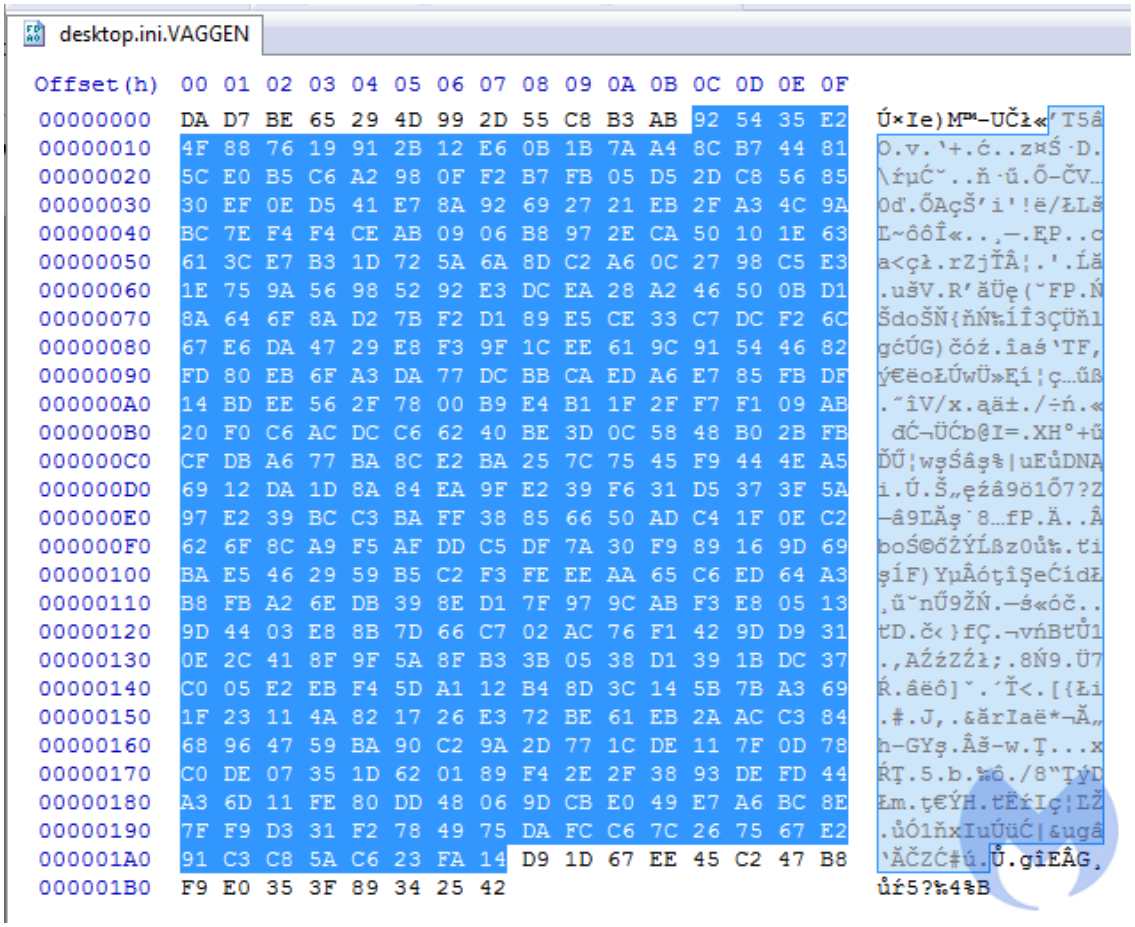
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

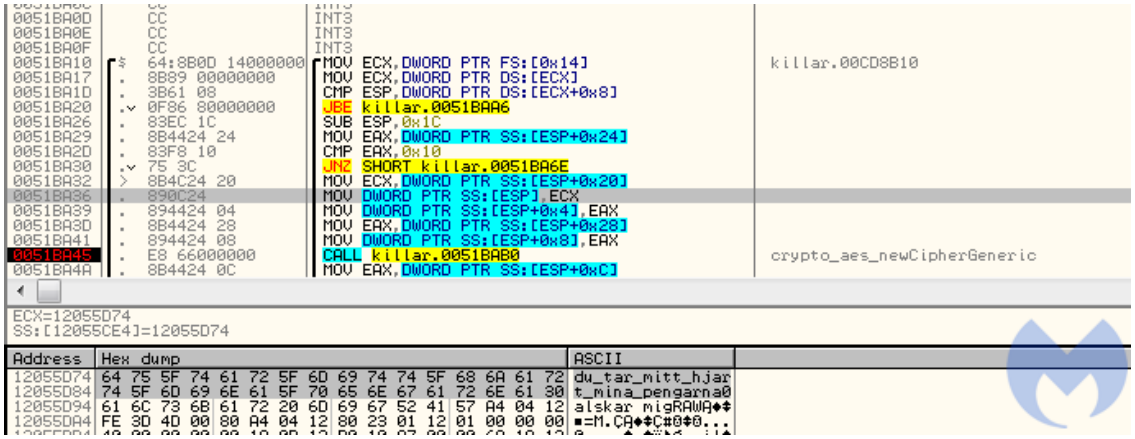


The content of the output file (with .VAGGEN extension) contains:

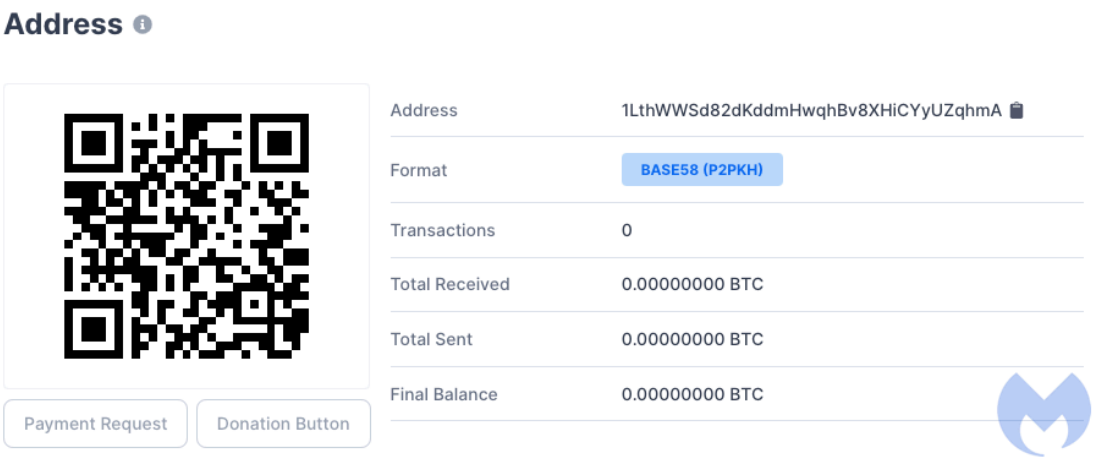
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

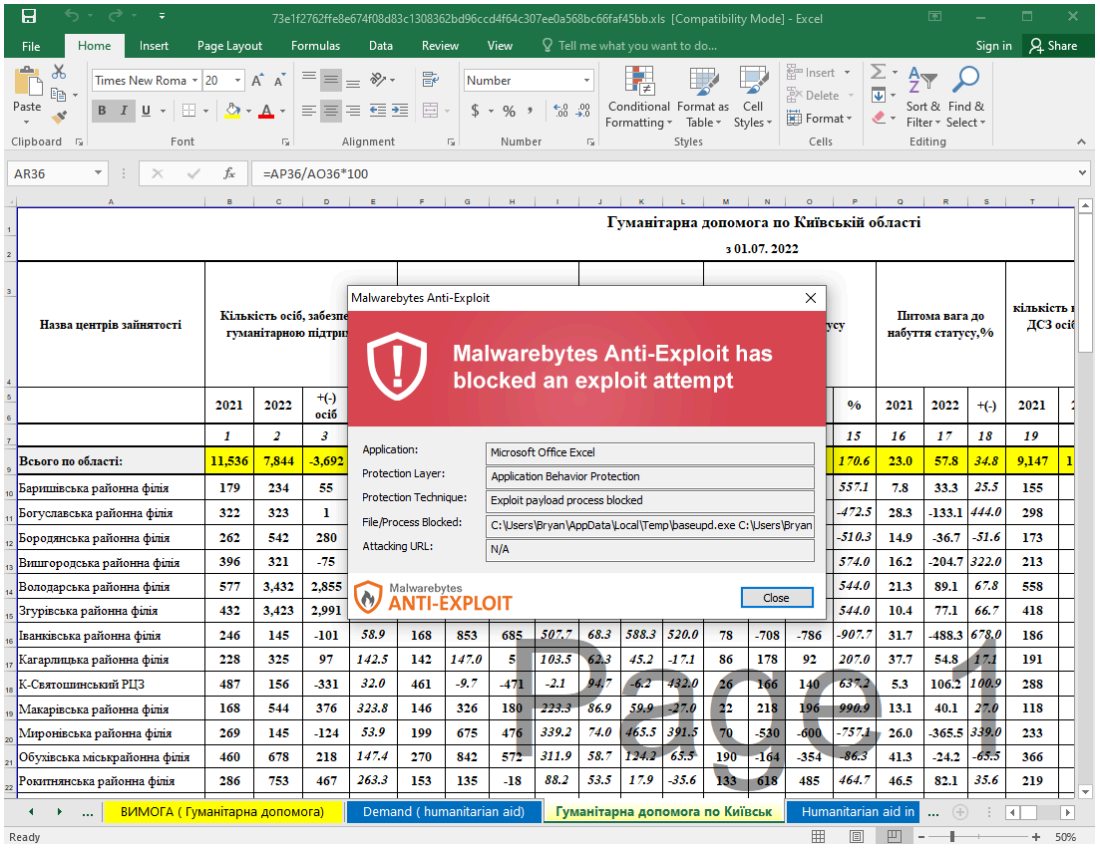
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

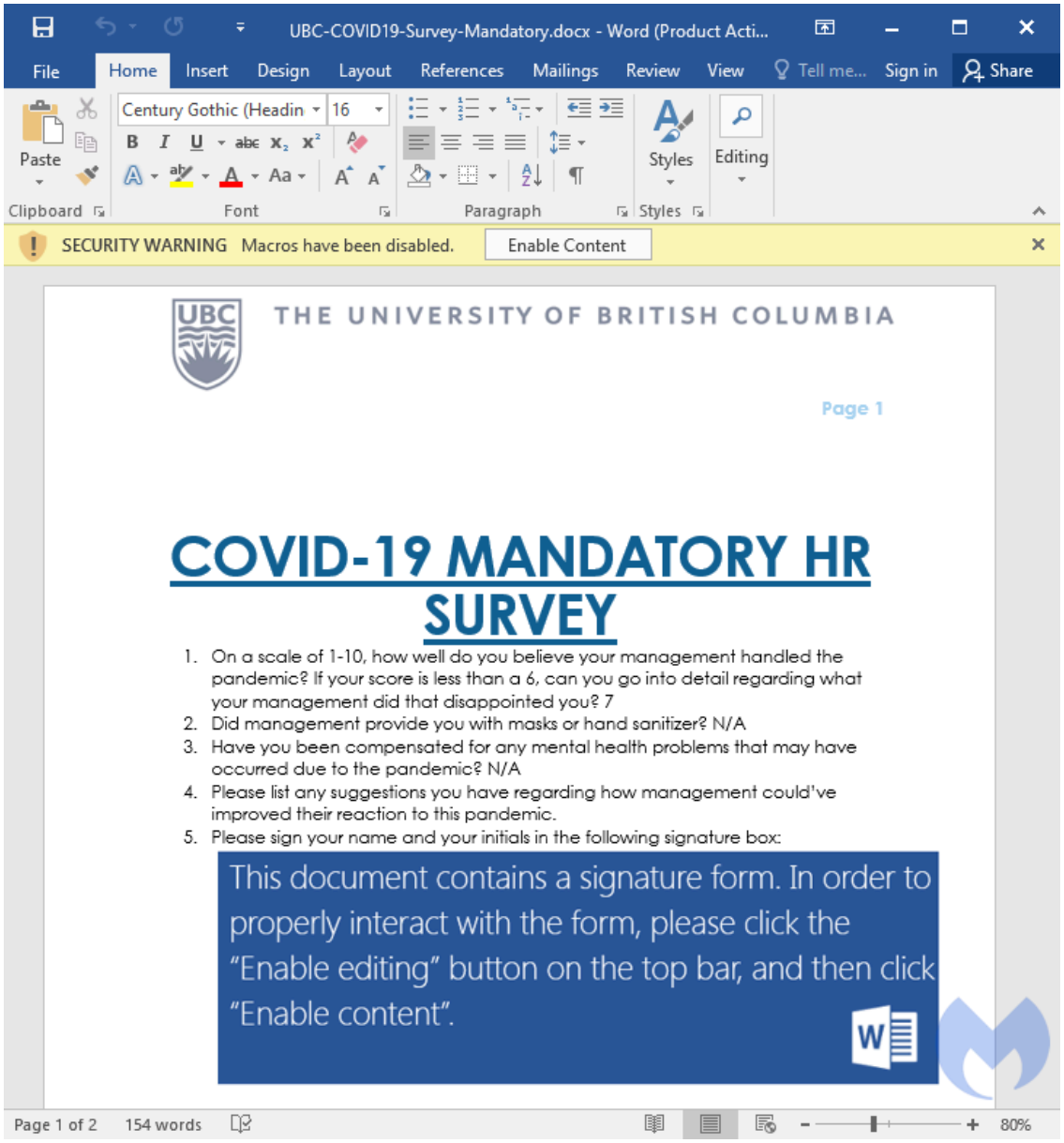
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

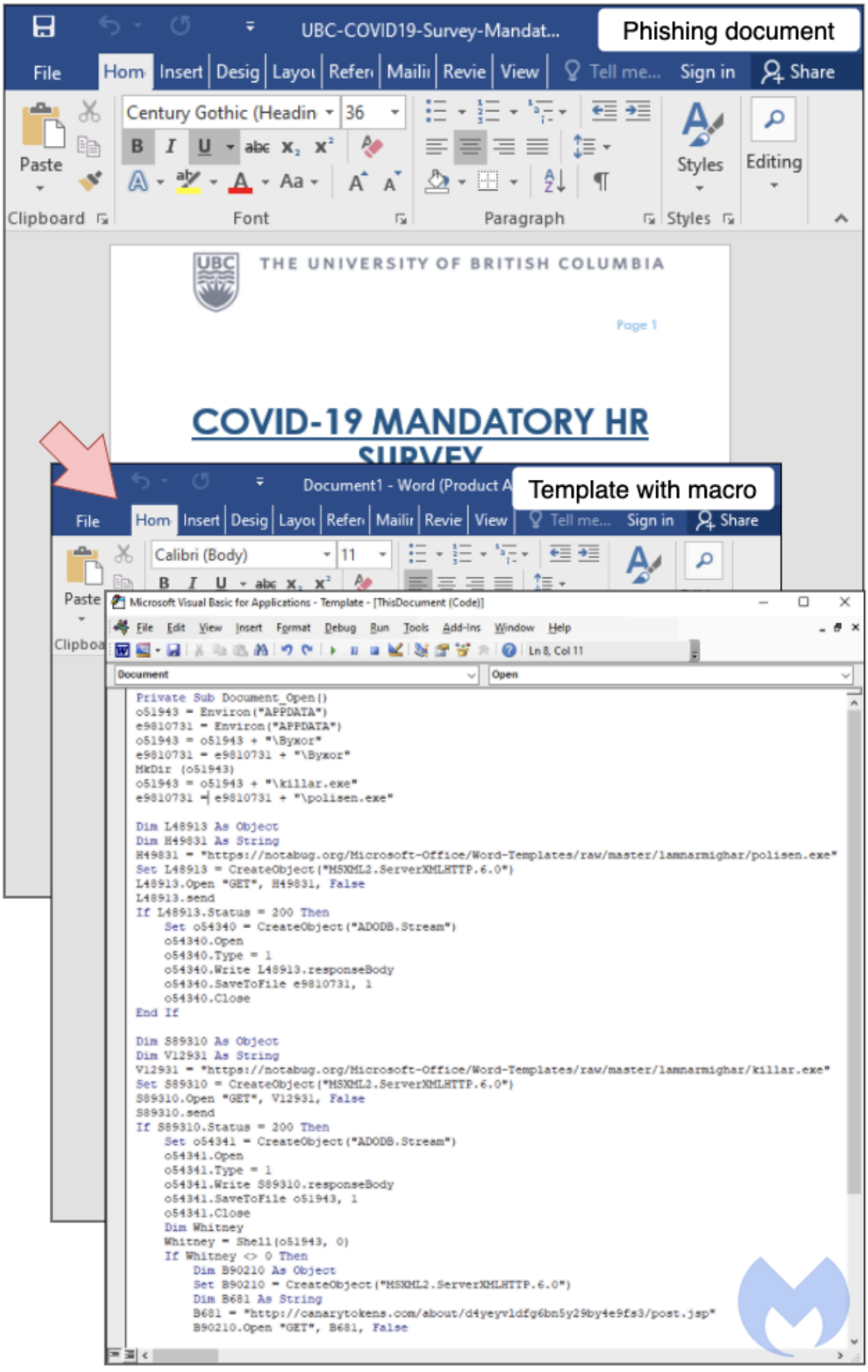
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

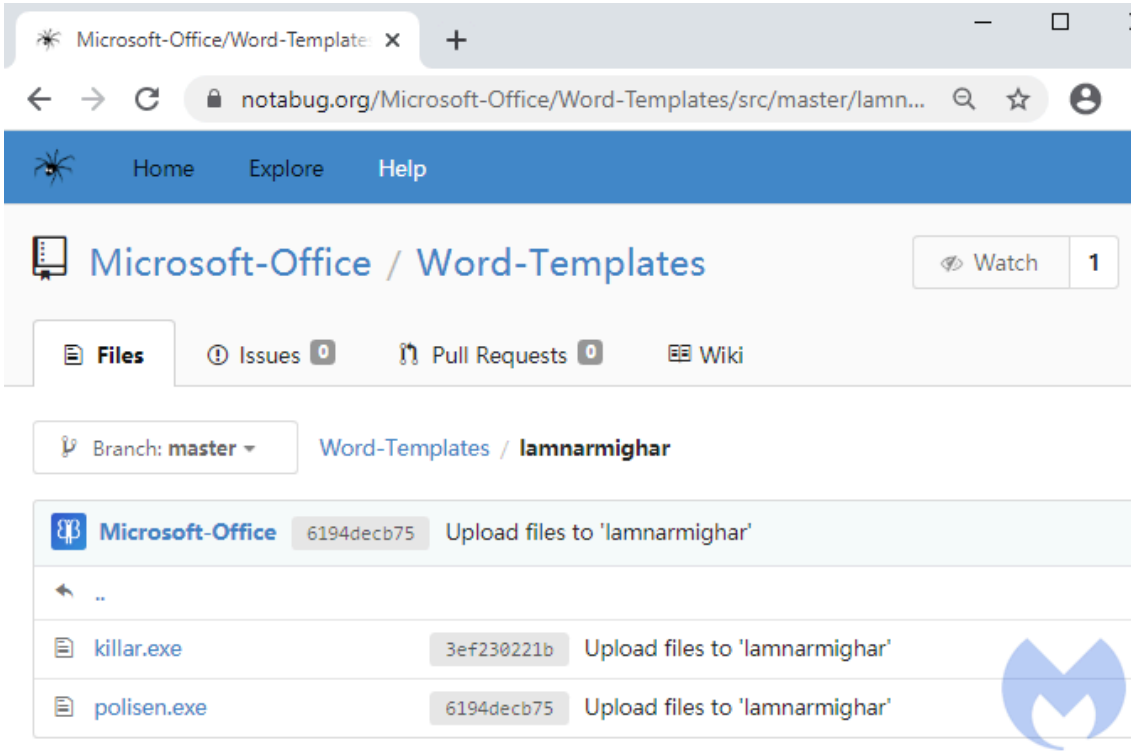
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



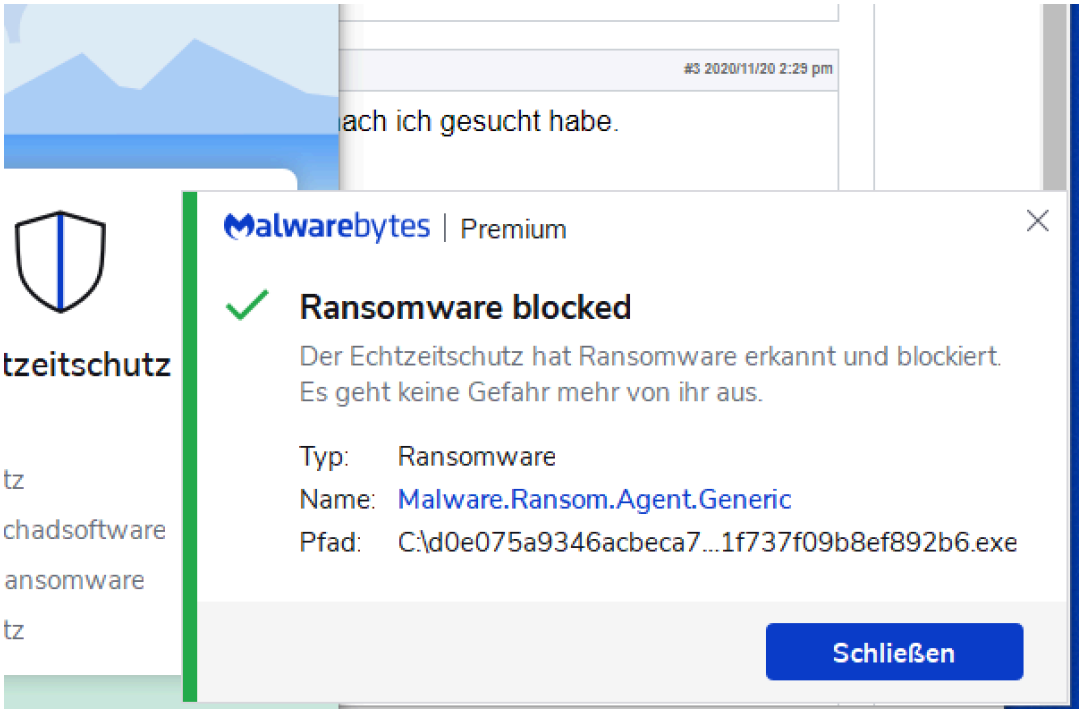
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

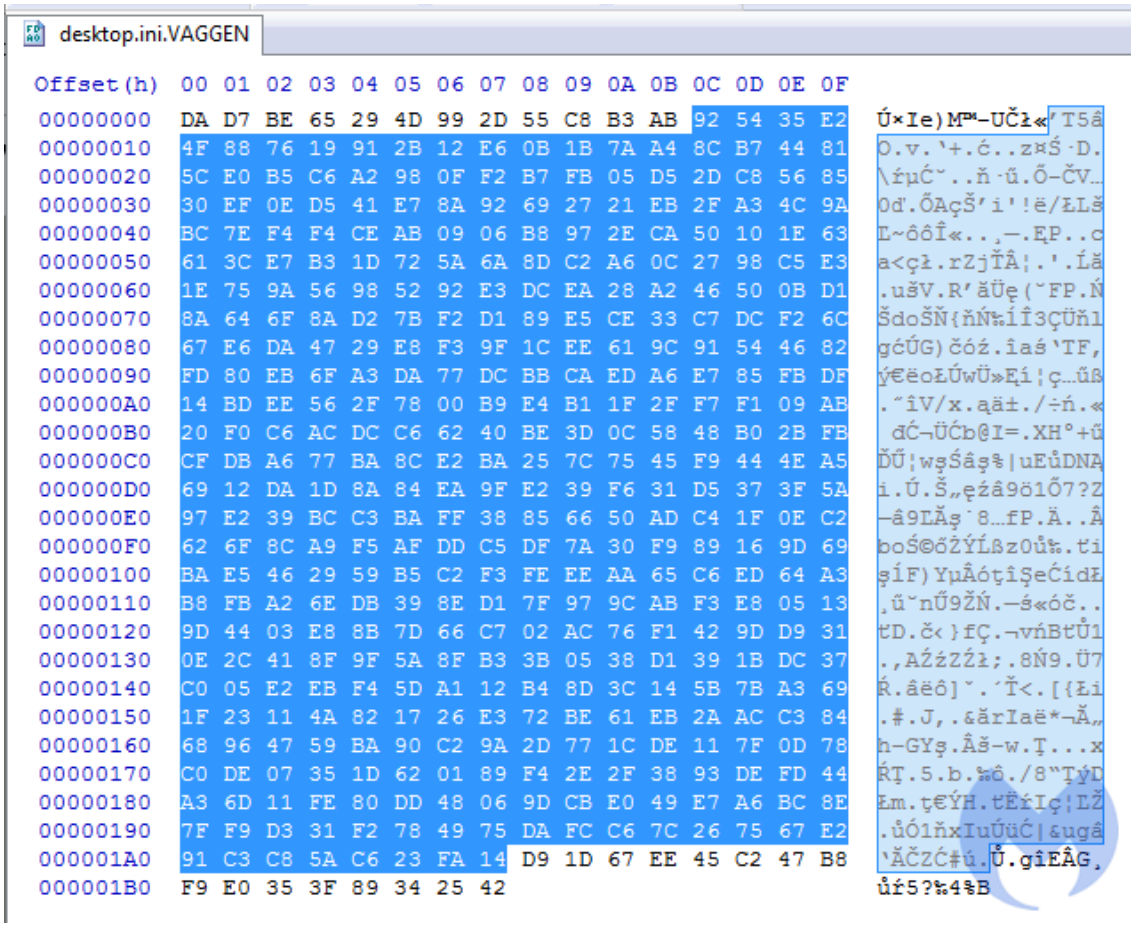
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

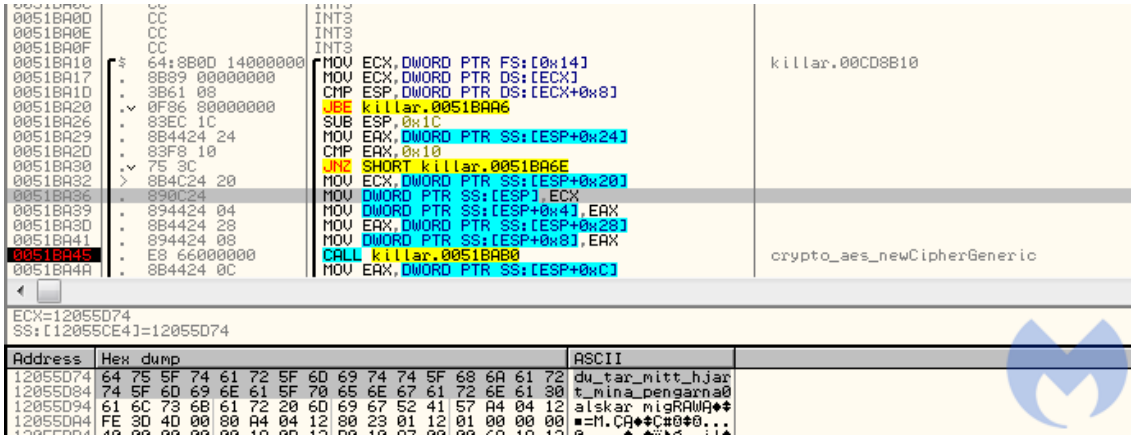


The content of the output file (with .VAGGEN extension) contains:

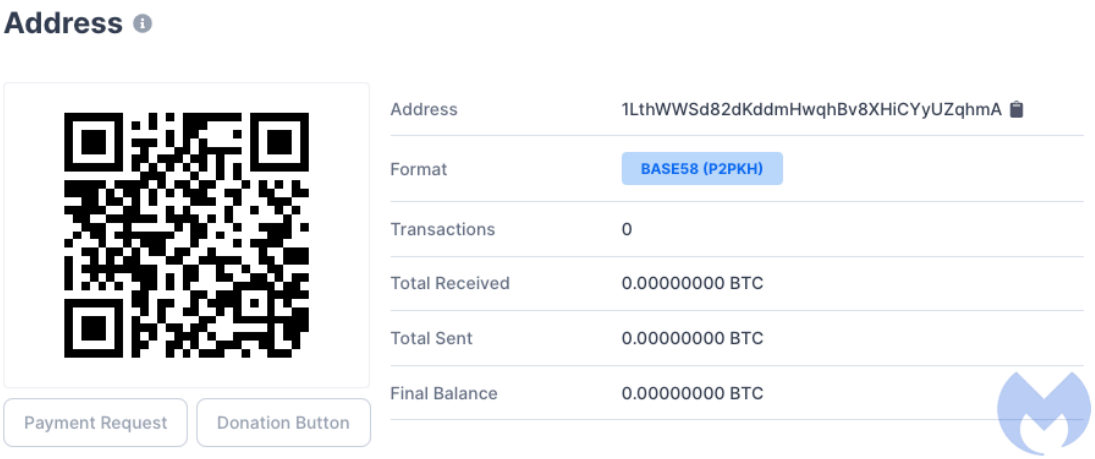
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

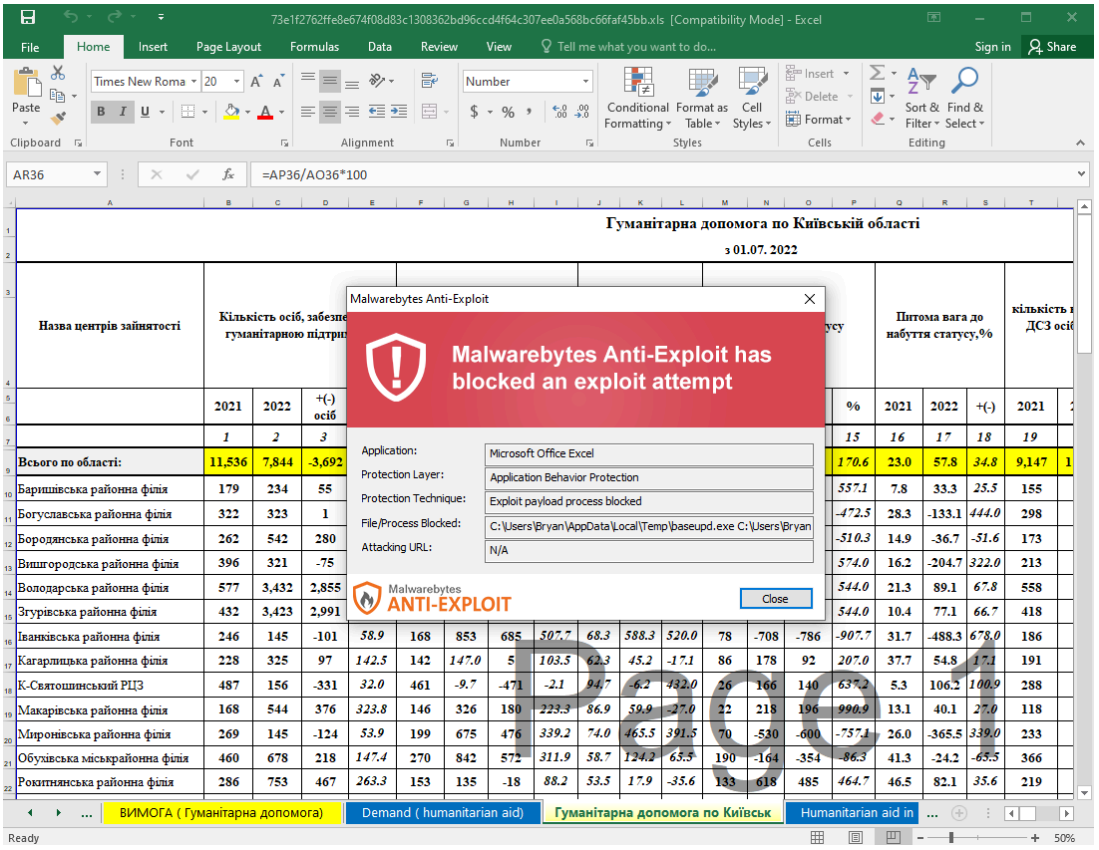
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

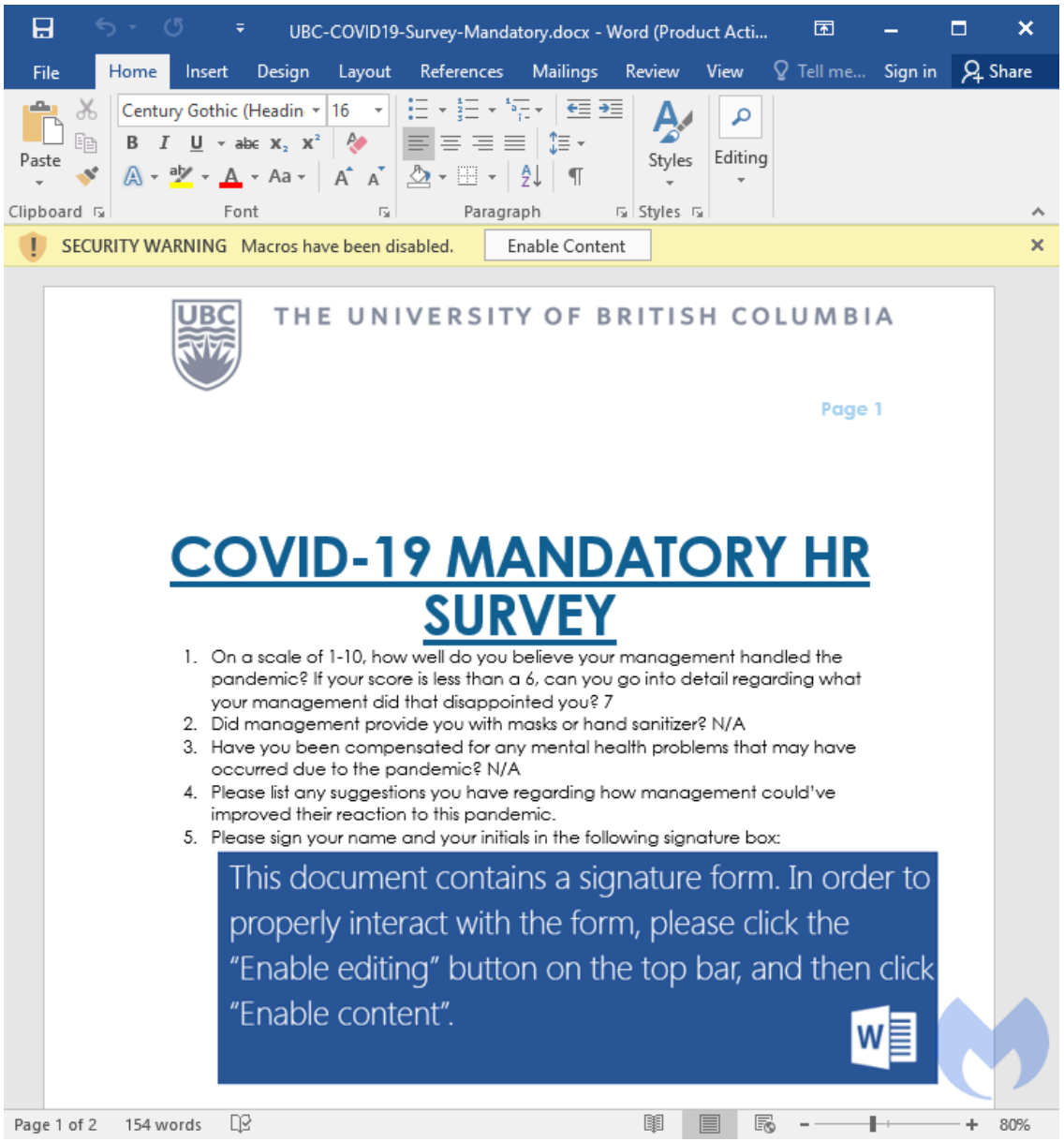
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

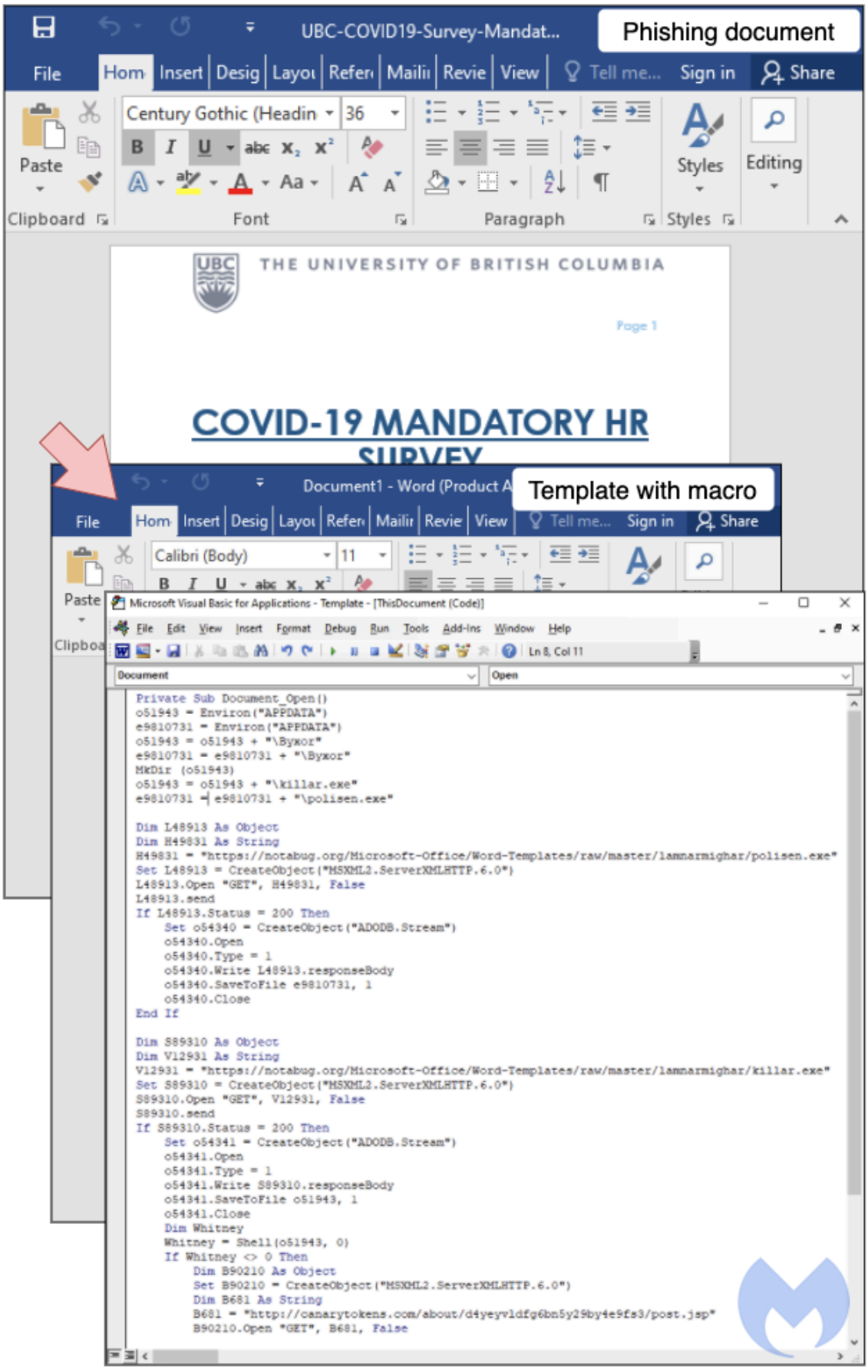
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

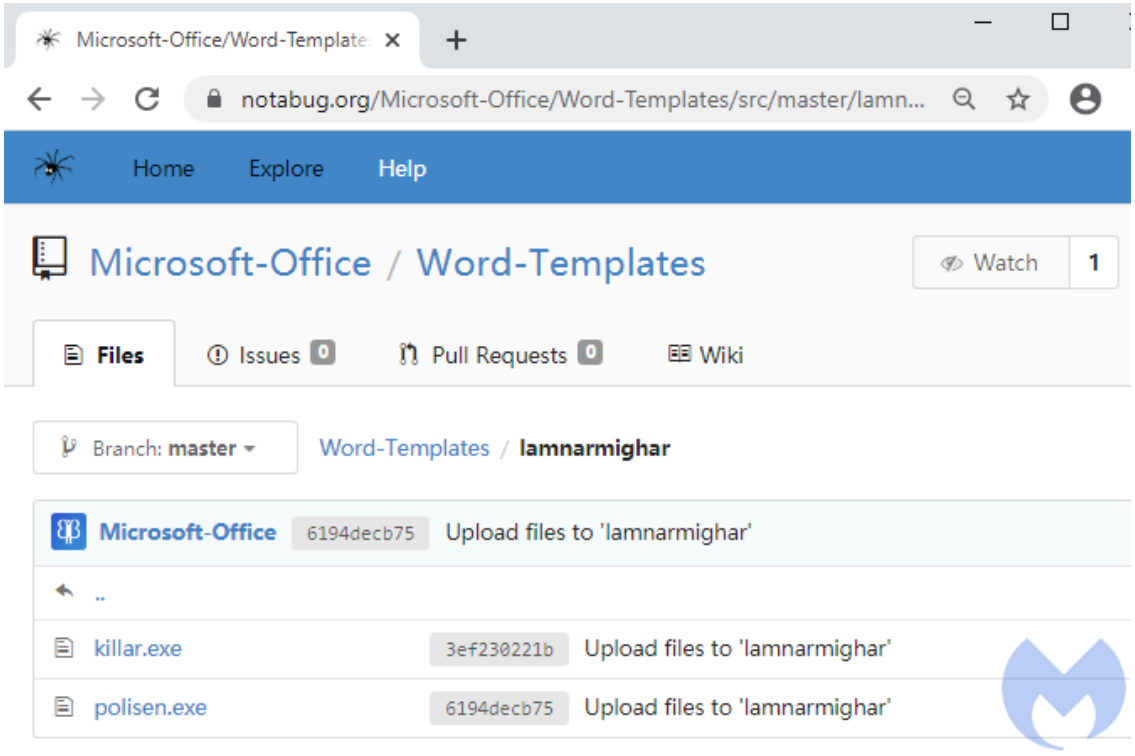
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



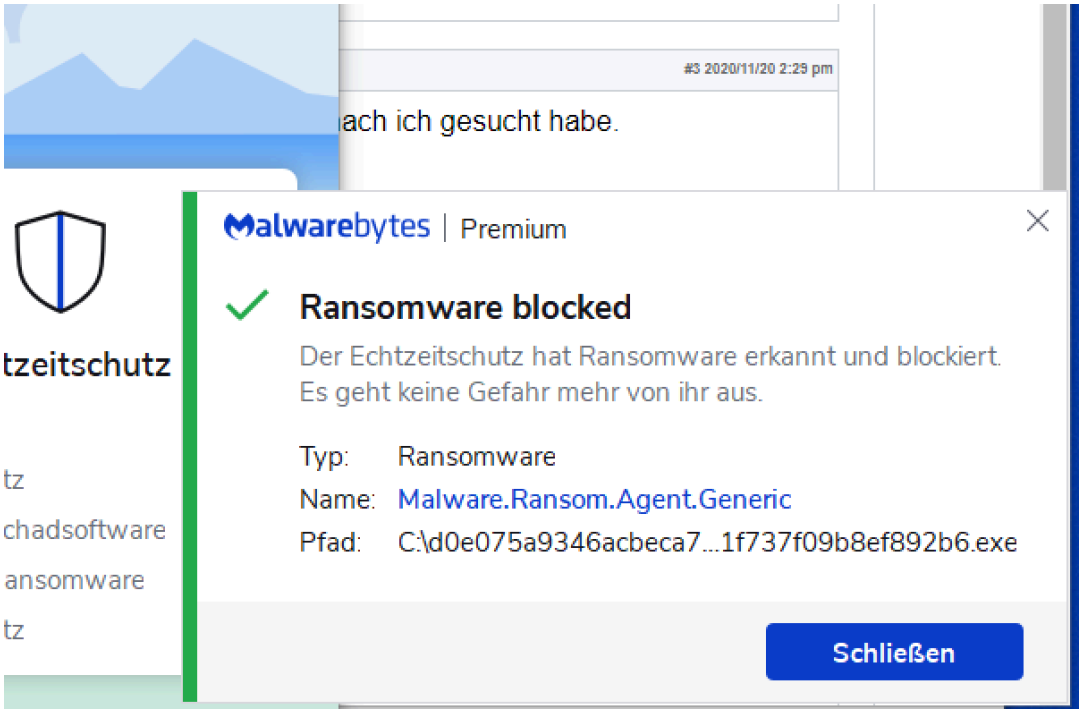
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

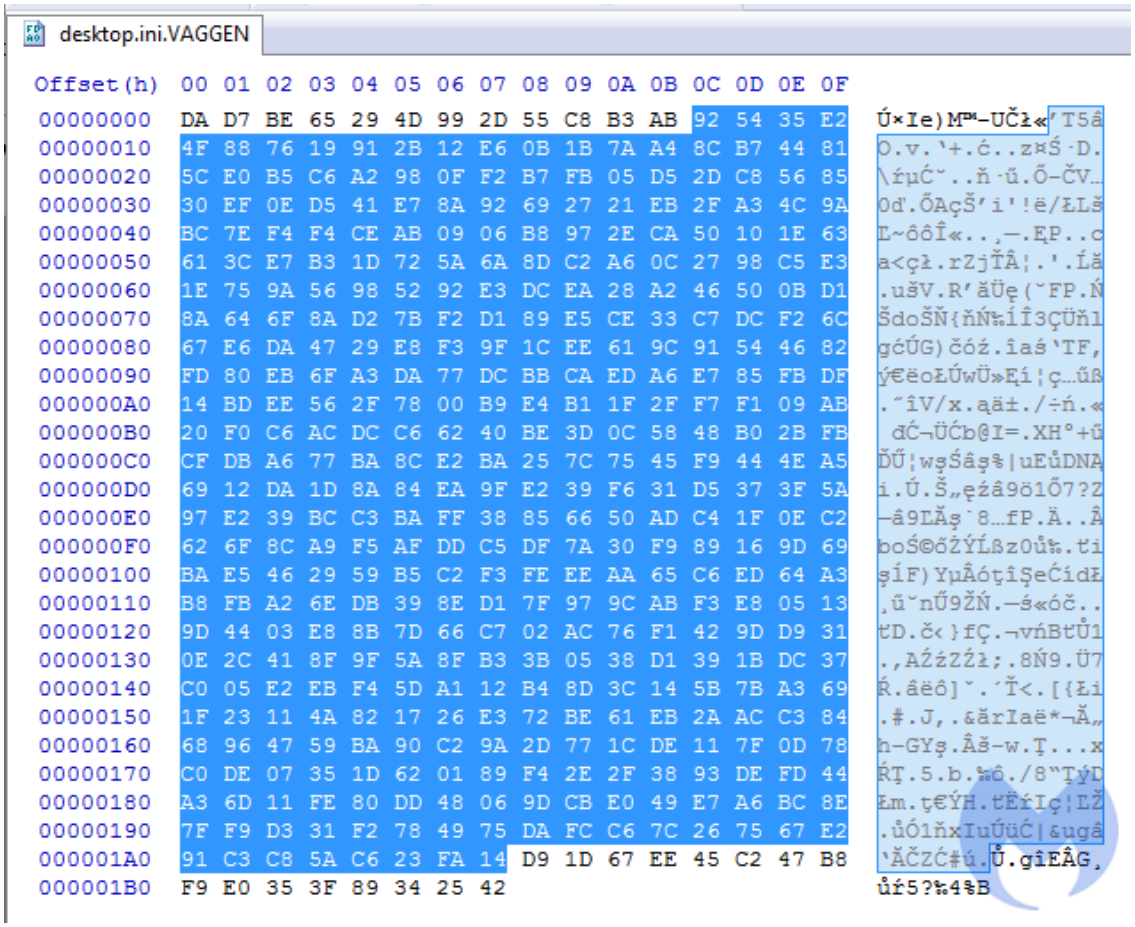
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

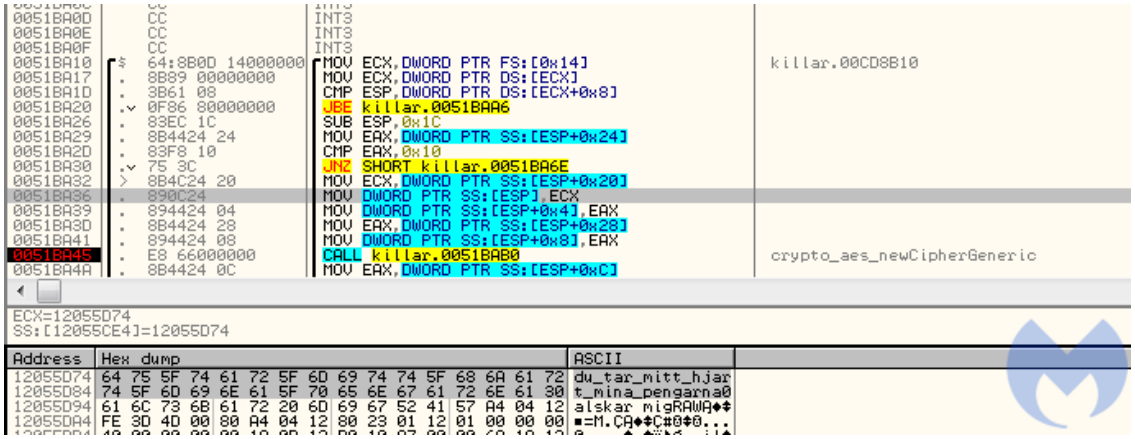


The content of the output file (with .VAGGEN extension) contains:

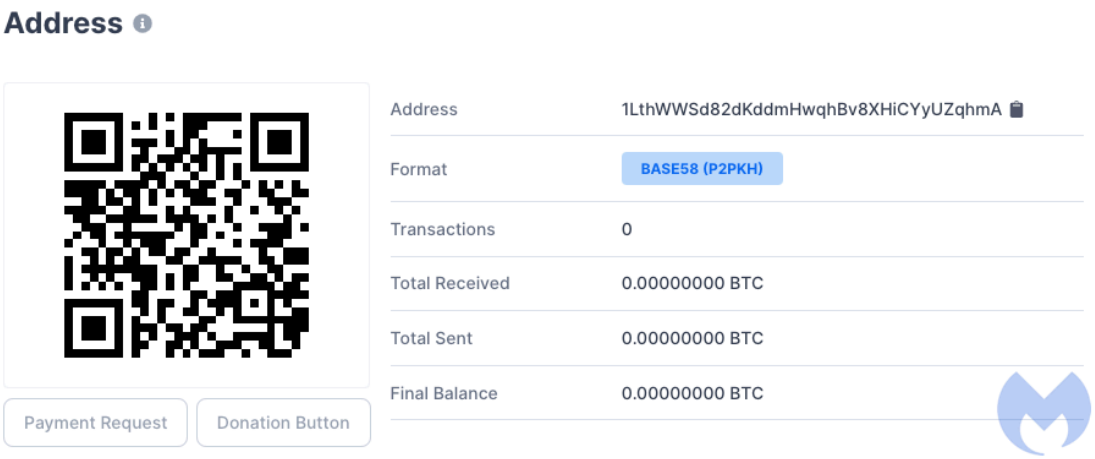
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

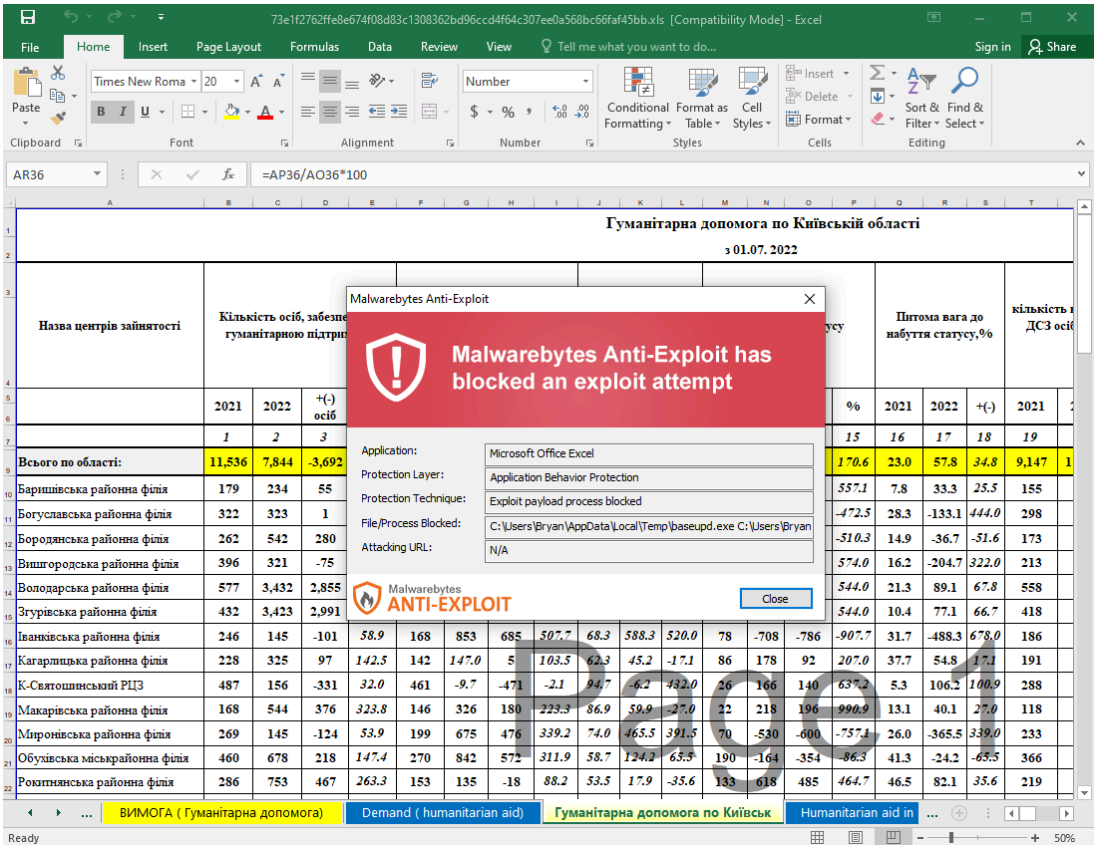
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

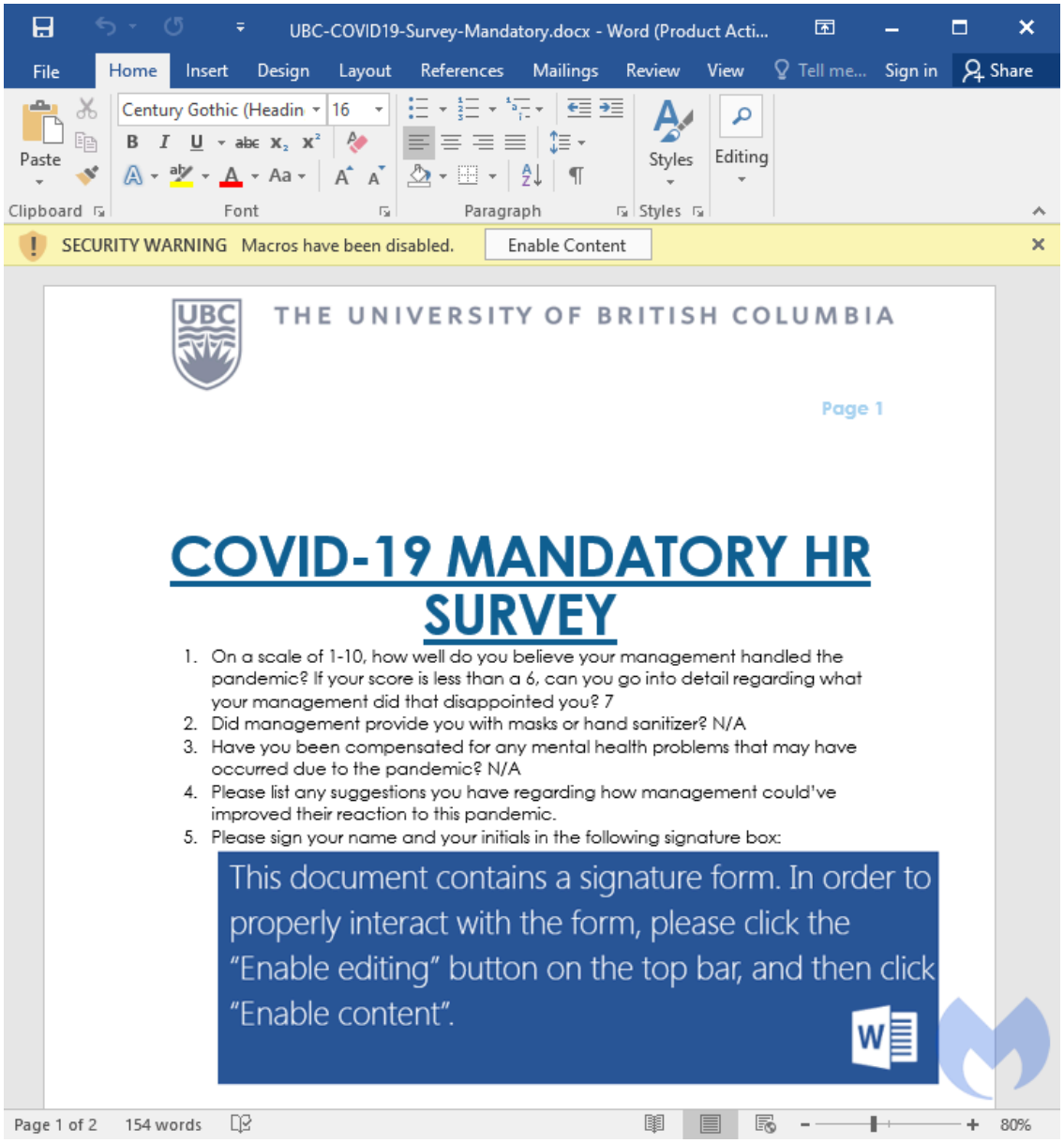
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

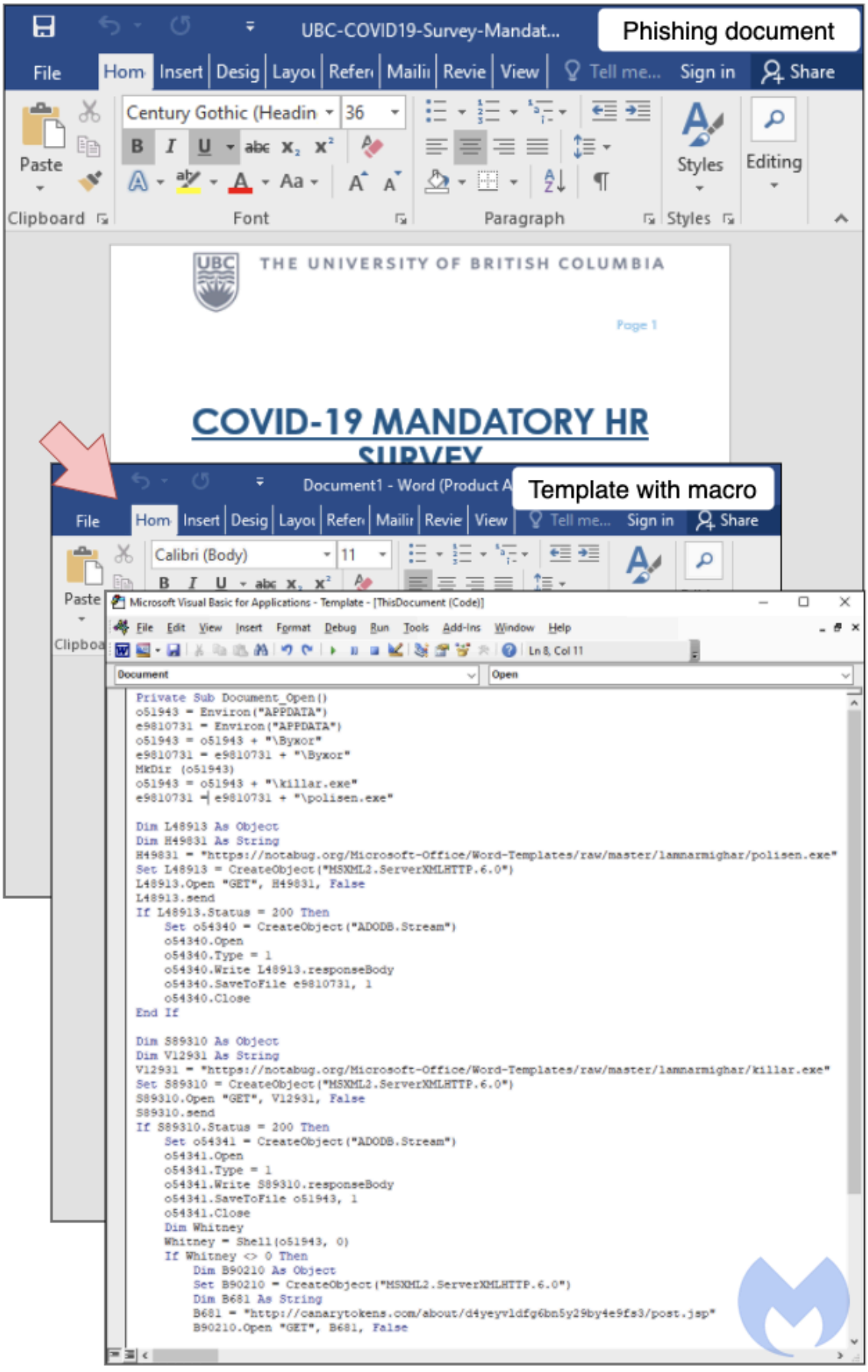
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

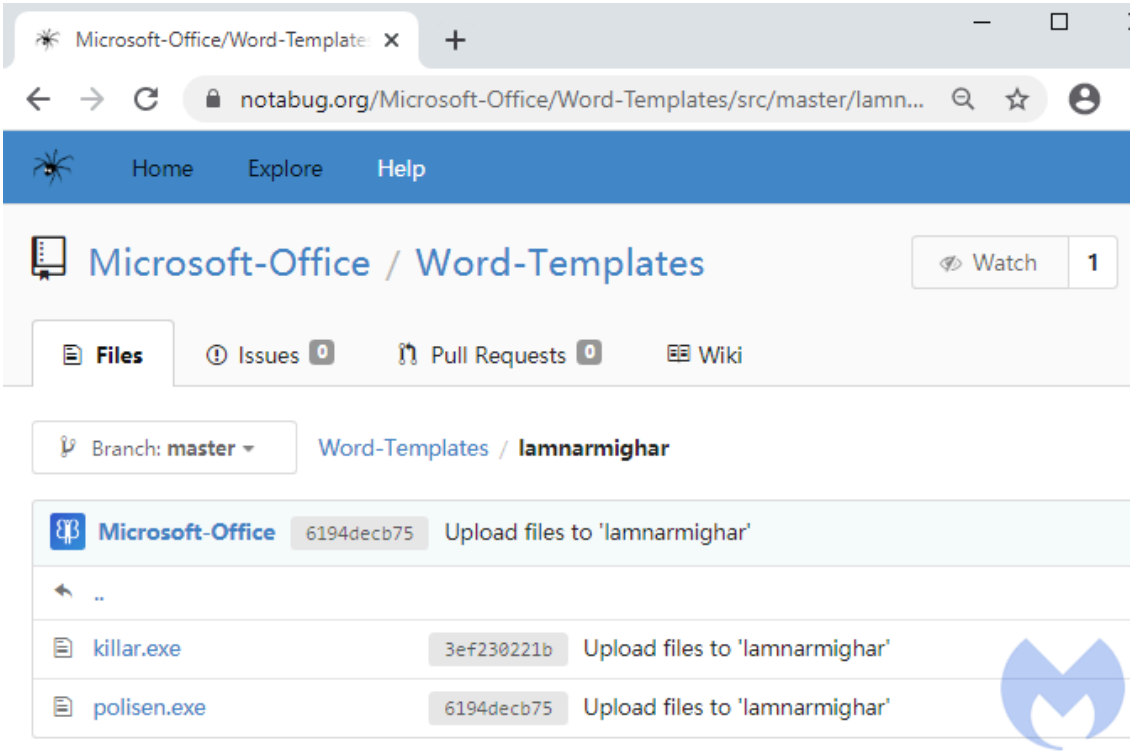
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



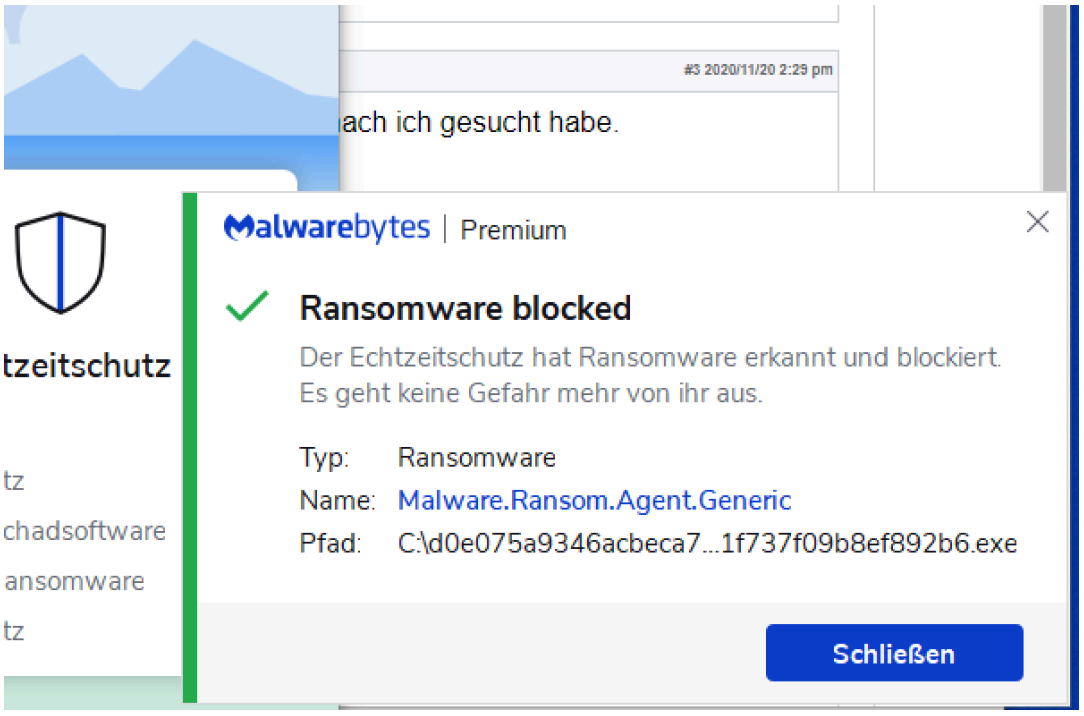
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

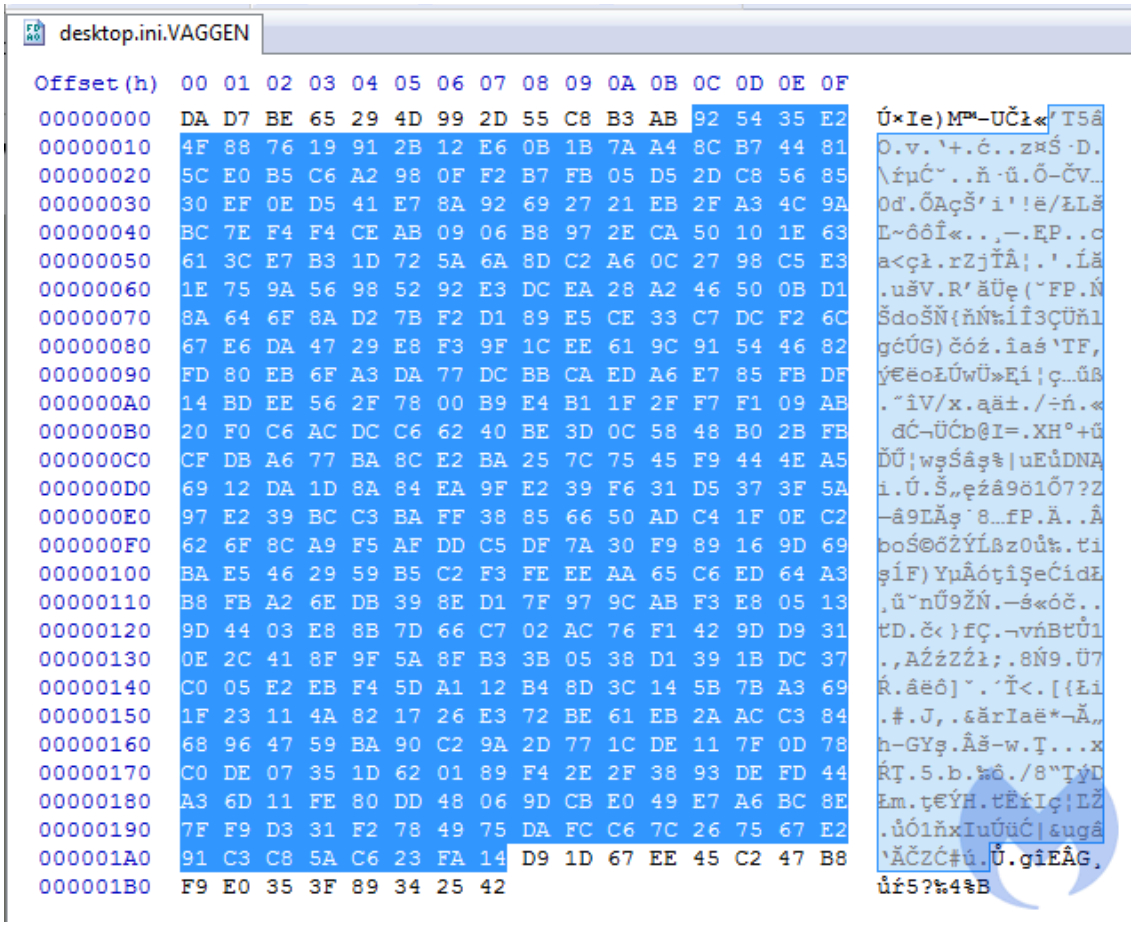
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

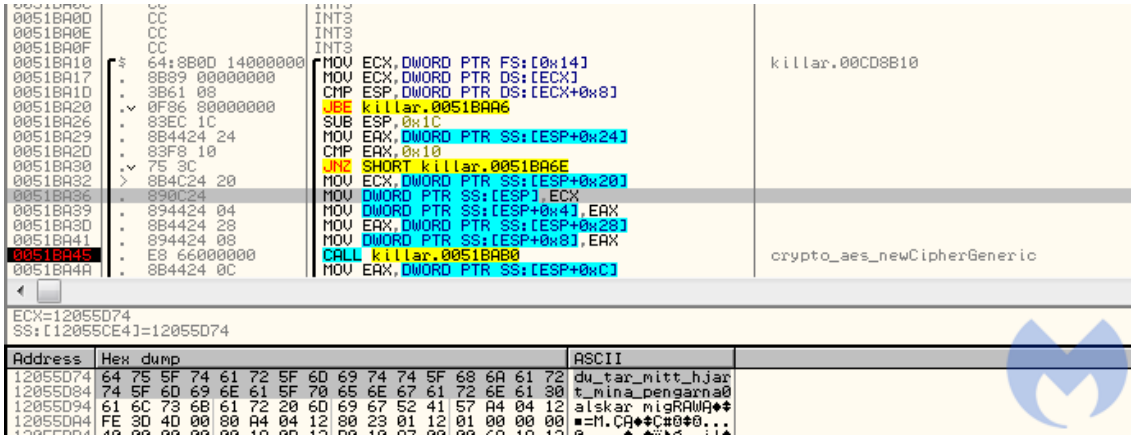


The content of the output file (with .VAGGEN extension) contains:

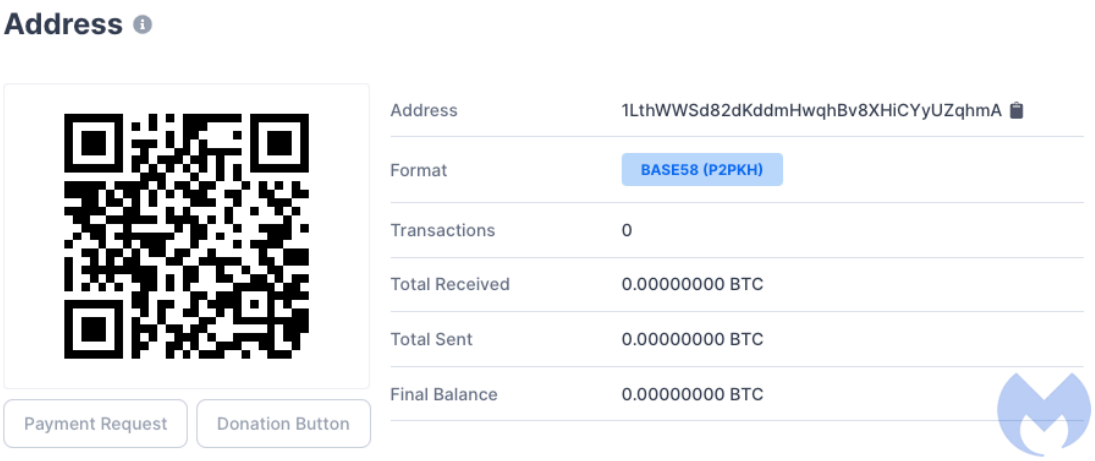
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

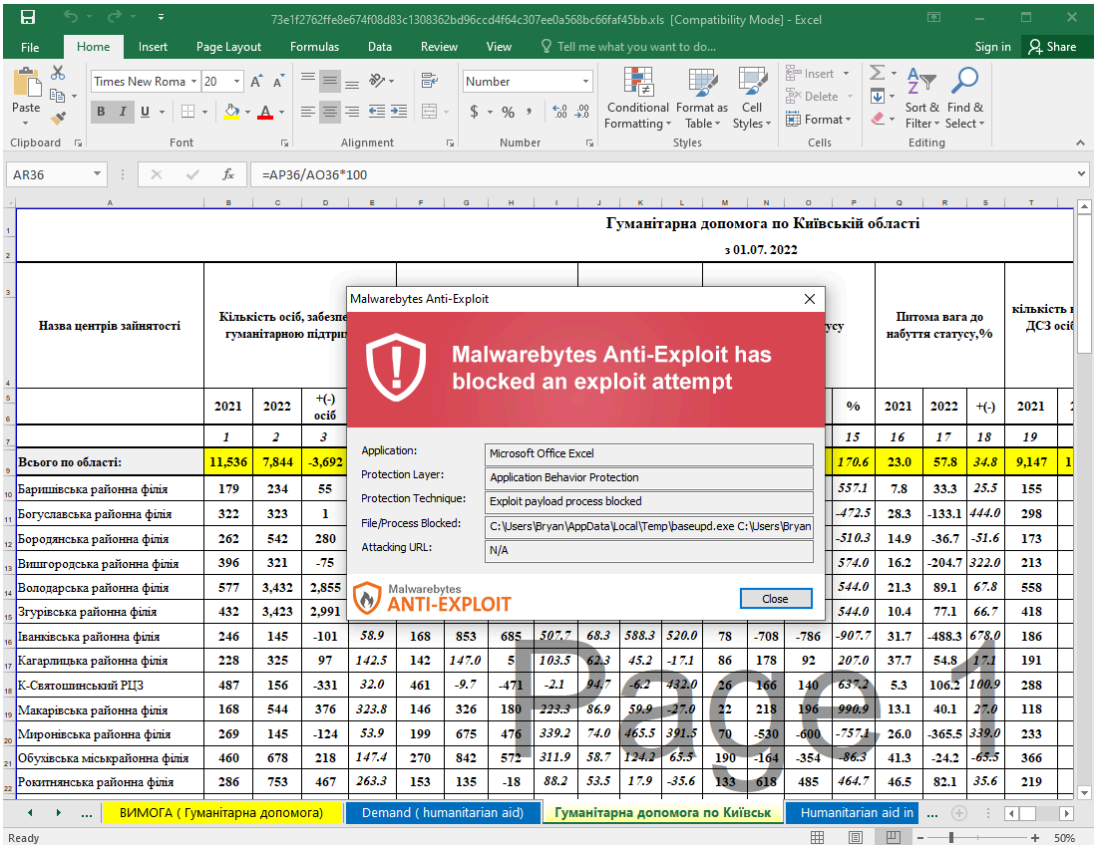
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

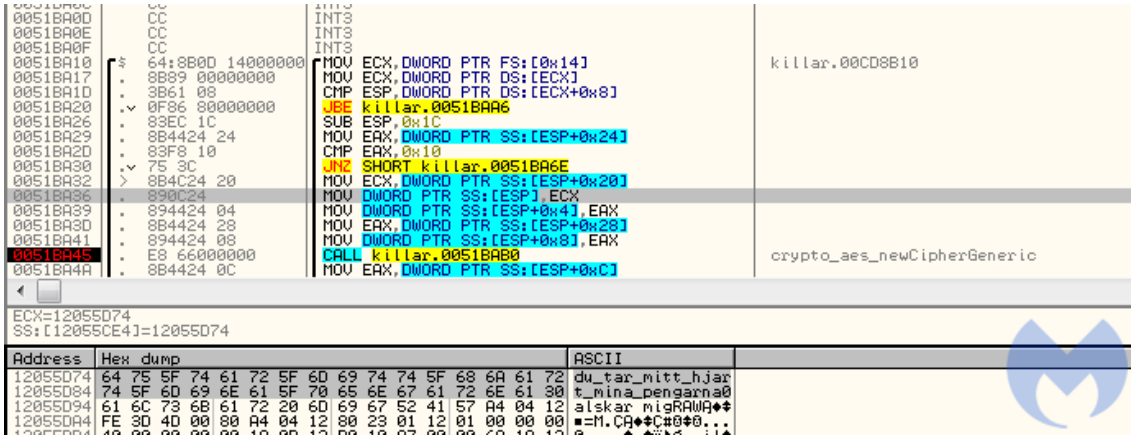
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

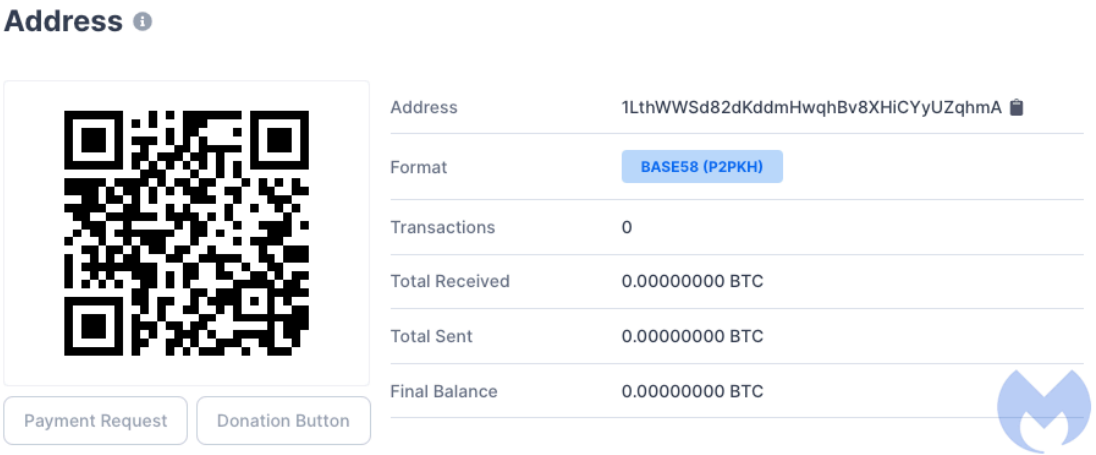
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

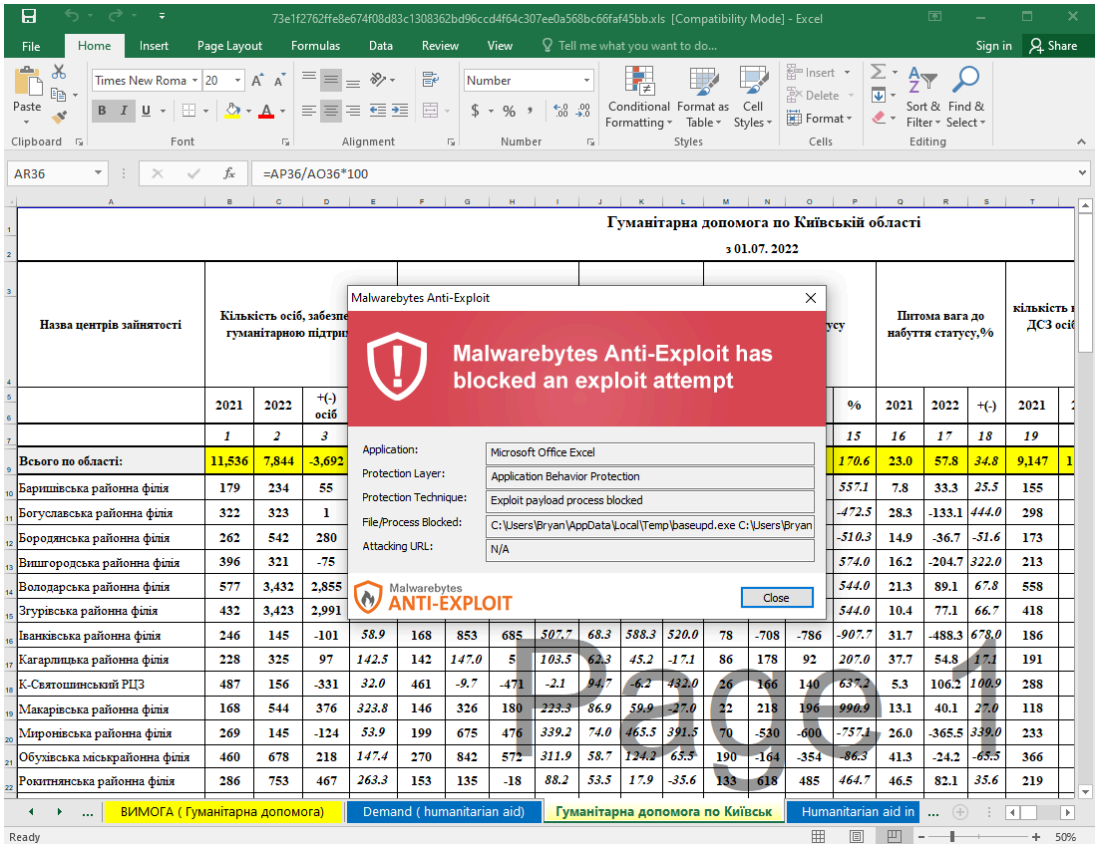
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

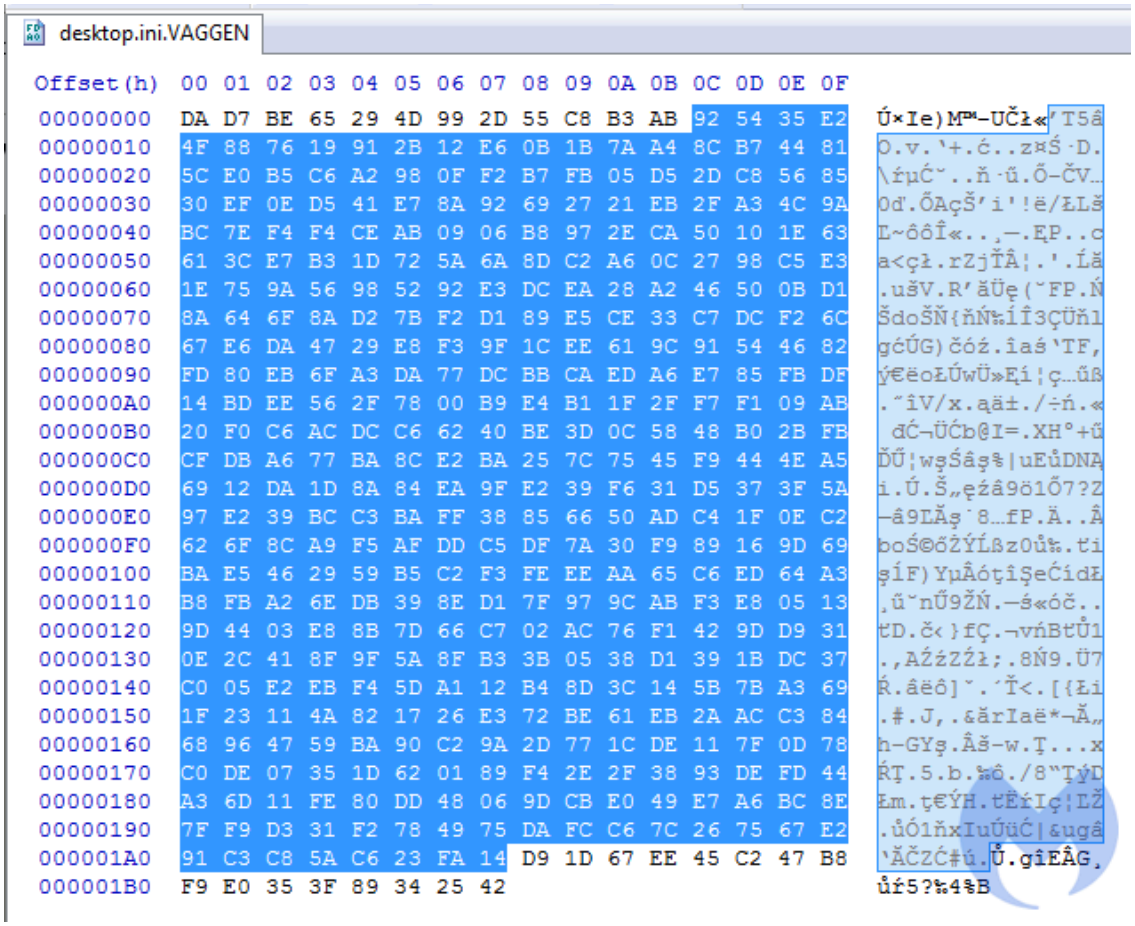
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

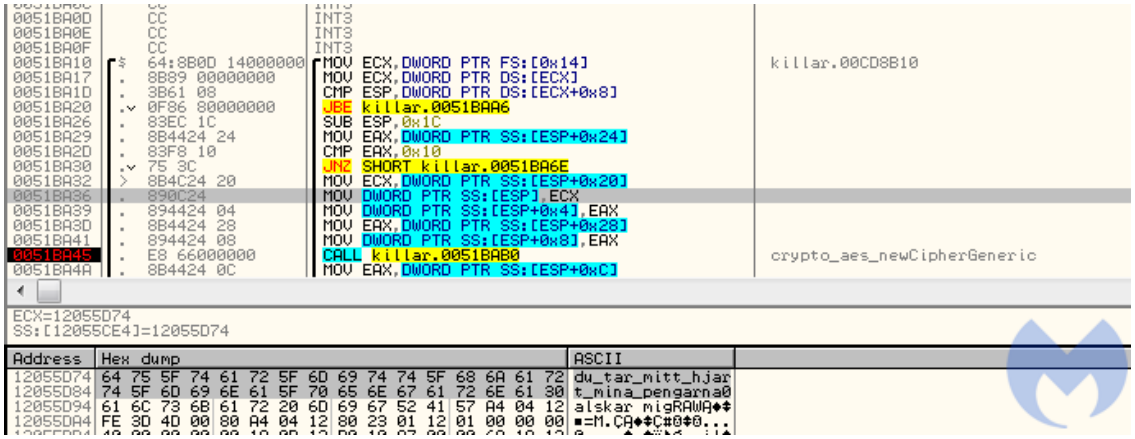


The content of the output file (with .VAGGEN extension) contains:

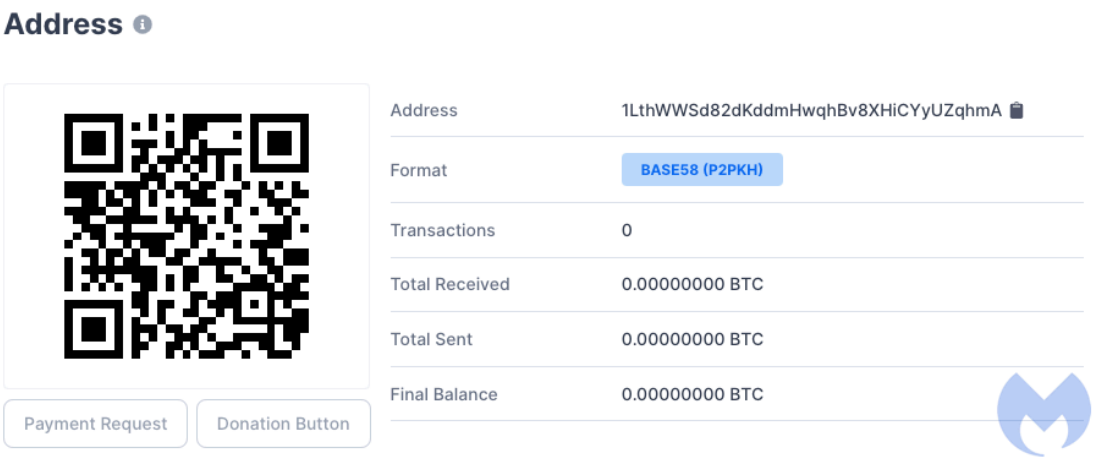
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

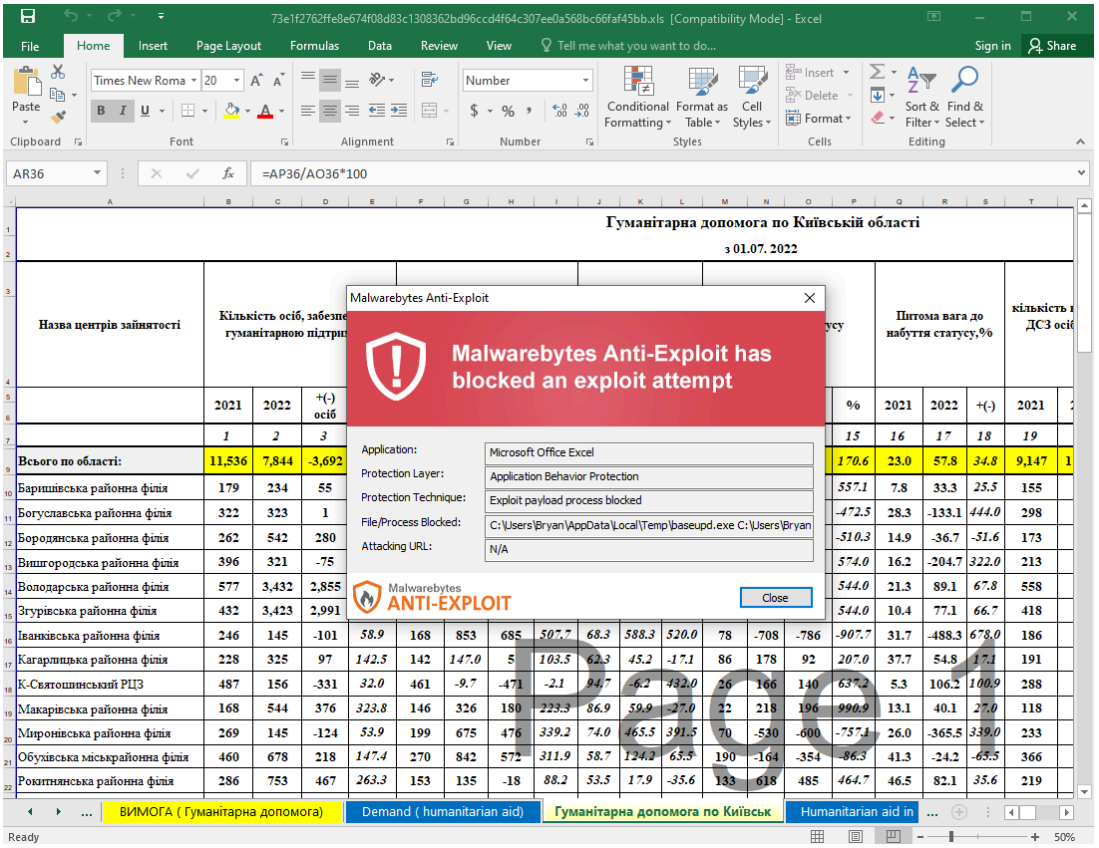
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
46  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
47    runtime_morestack_noctxt();
48  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
49  ((void (*)(void))loc_45CDA0)();
50  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
51  v38 = v4;
52  v30 = v5;
53  v31 = v6;
54  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
55  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
56  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
57  v32 = v10;
58  v11 = *(void (__cdecl **)(int))(v10 + 16);
59  v39 = v12;
60  v11(v12);
61  v29 = v13;
62  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
63  v40 = v15;
64  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
65  v17 = v27;
66  if ( v27 )
67  {
68    if ( v27 )
69      v17 = *(_DWORD *)(v27 + 4);
70    v43 = v17;
71    v44 = v28;
72    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
73  }
```

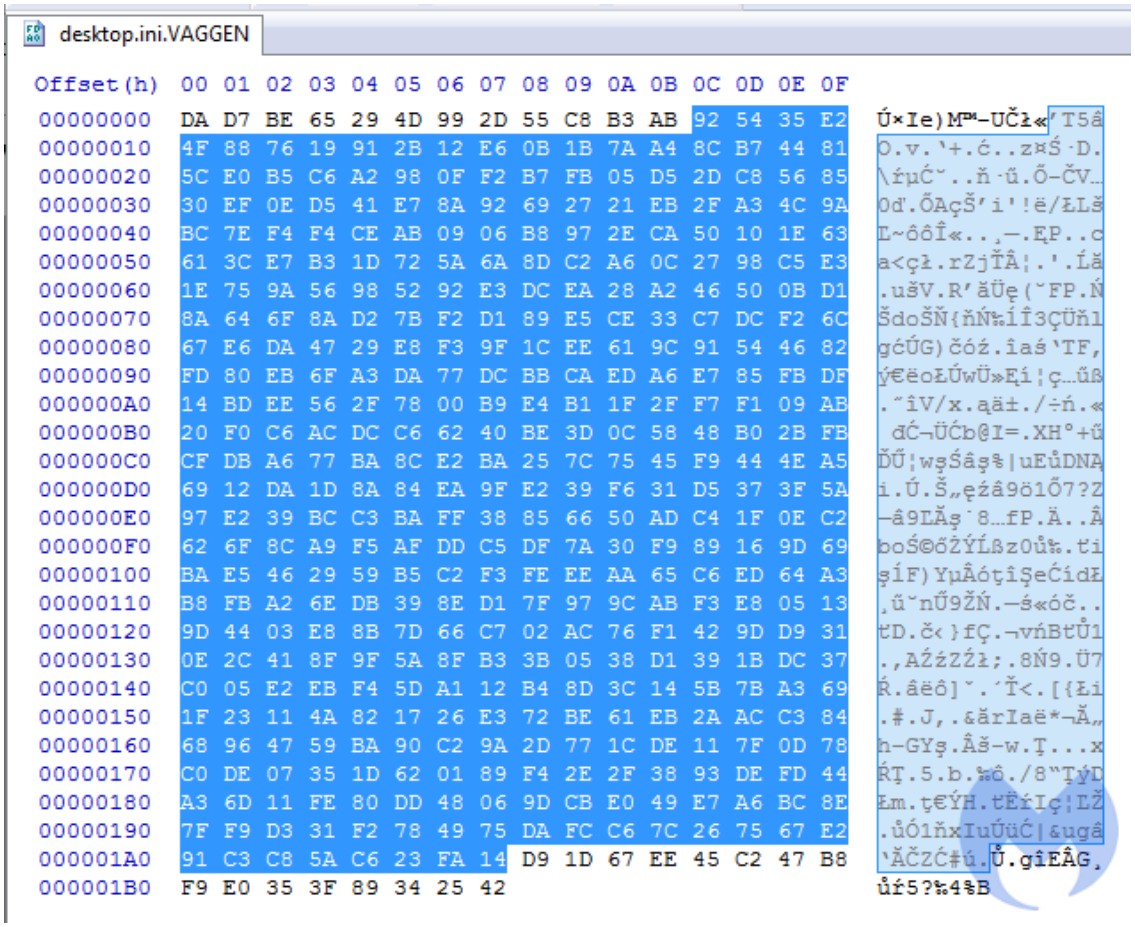


The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

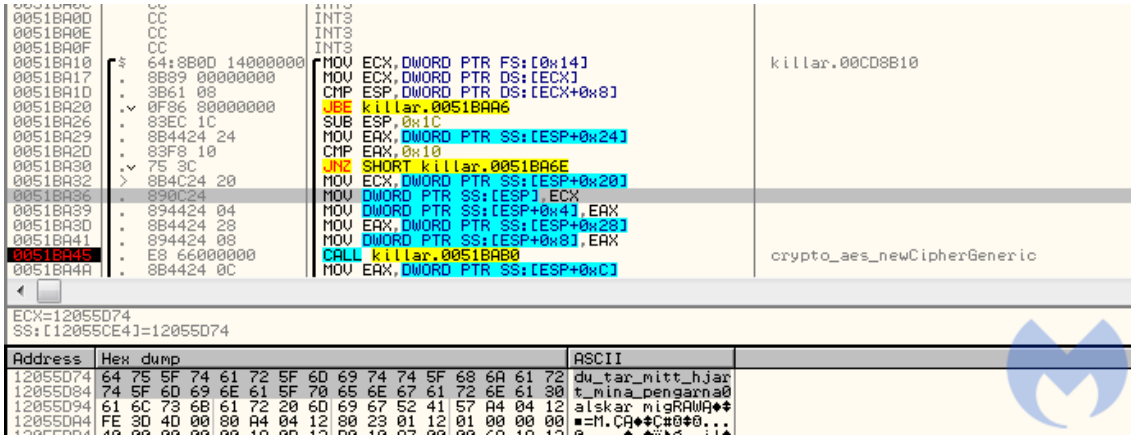


The content of the output file (with .VAGGEN extension) contains:

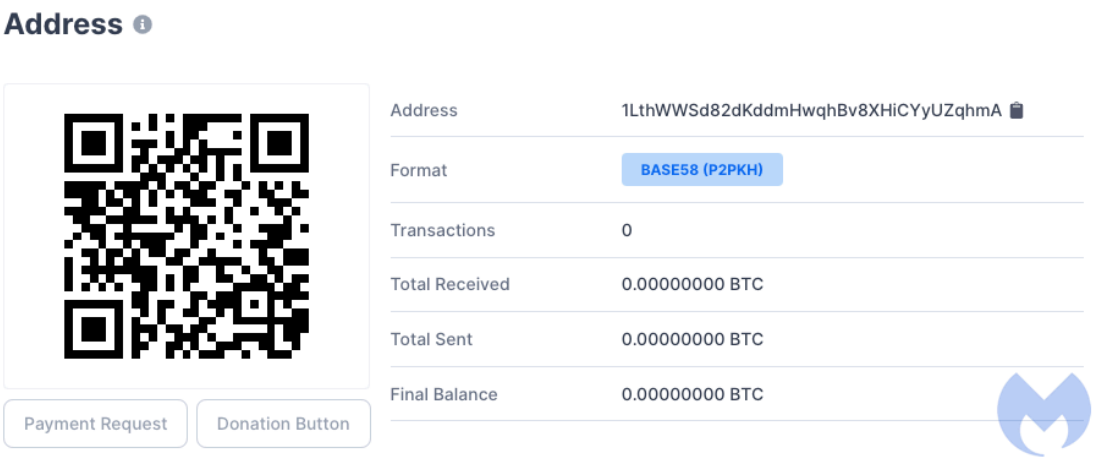
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

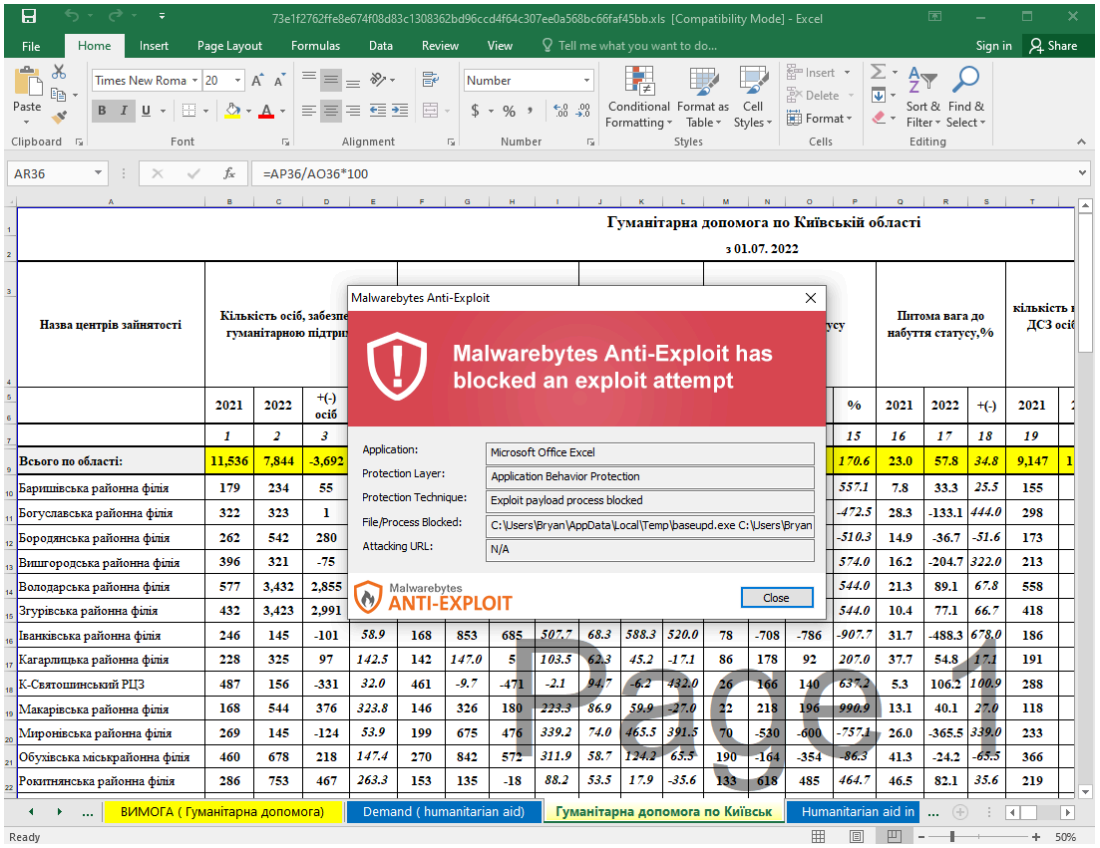
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

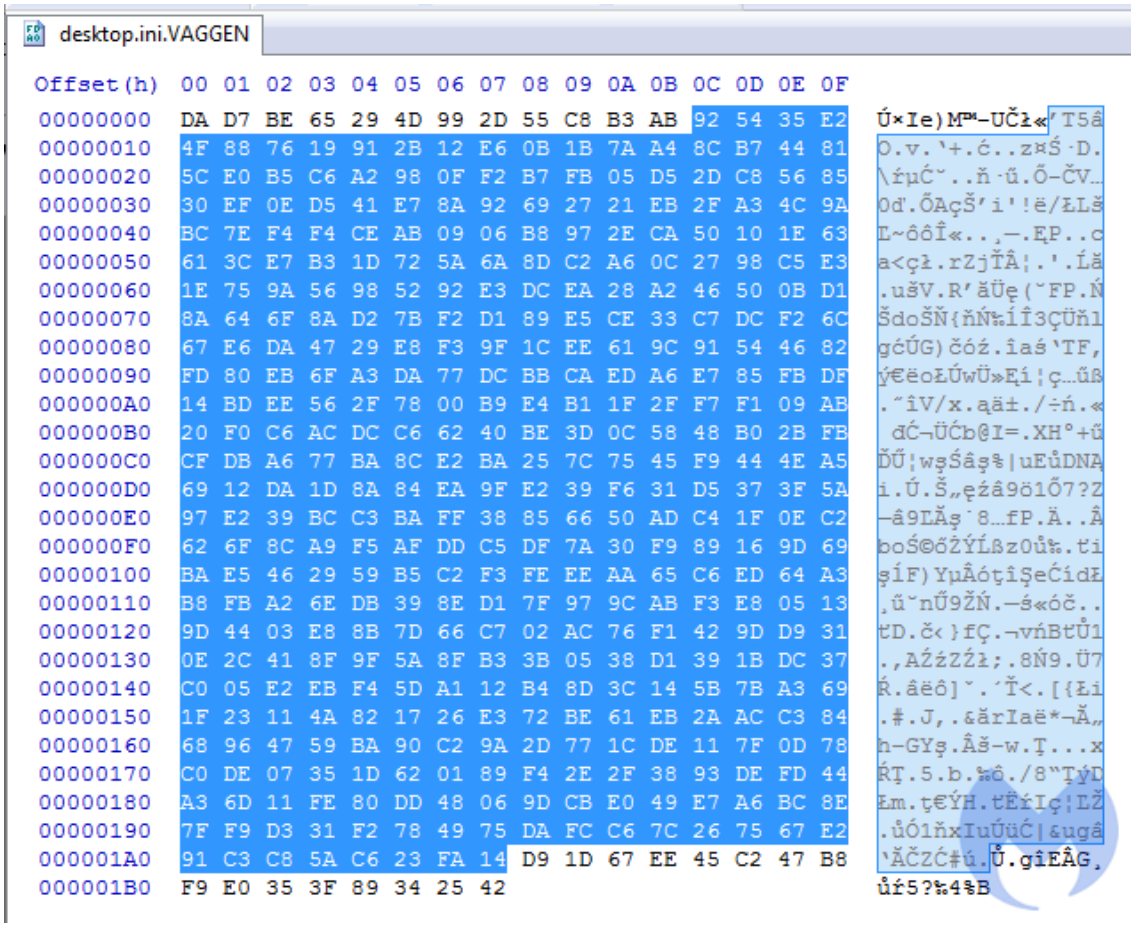
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

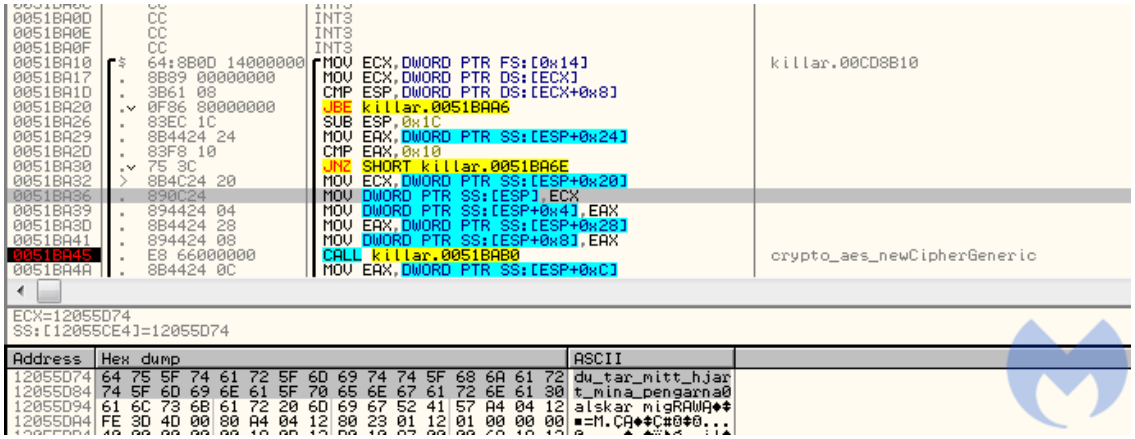


The content of the output file (with .VAGGEN extension) contains:

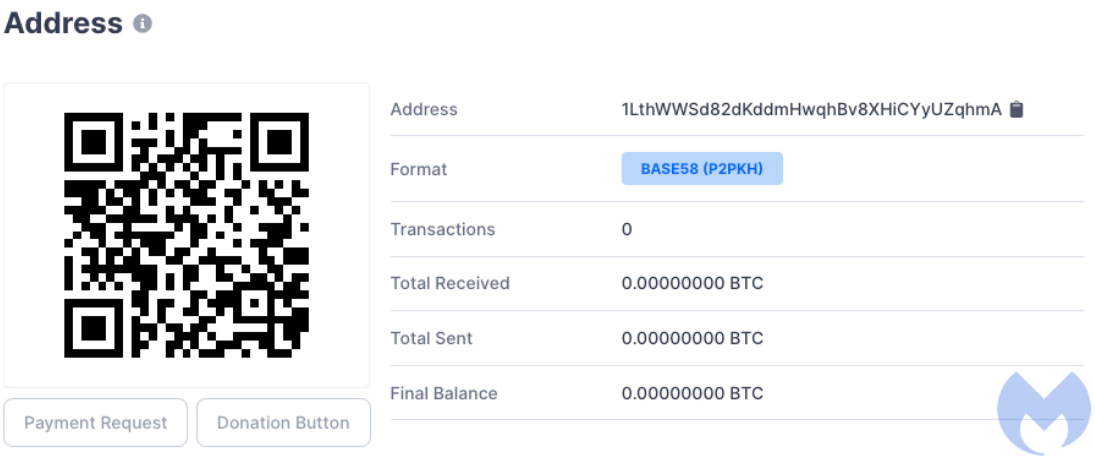
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

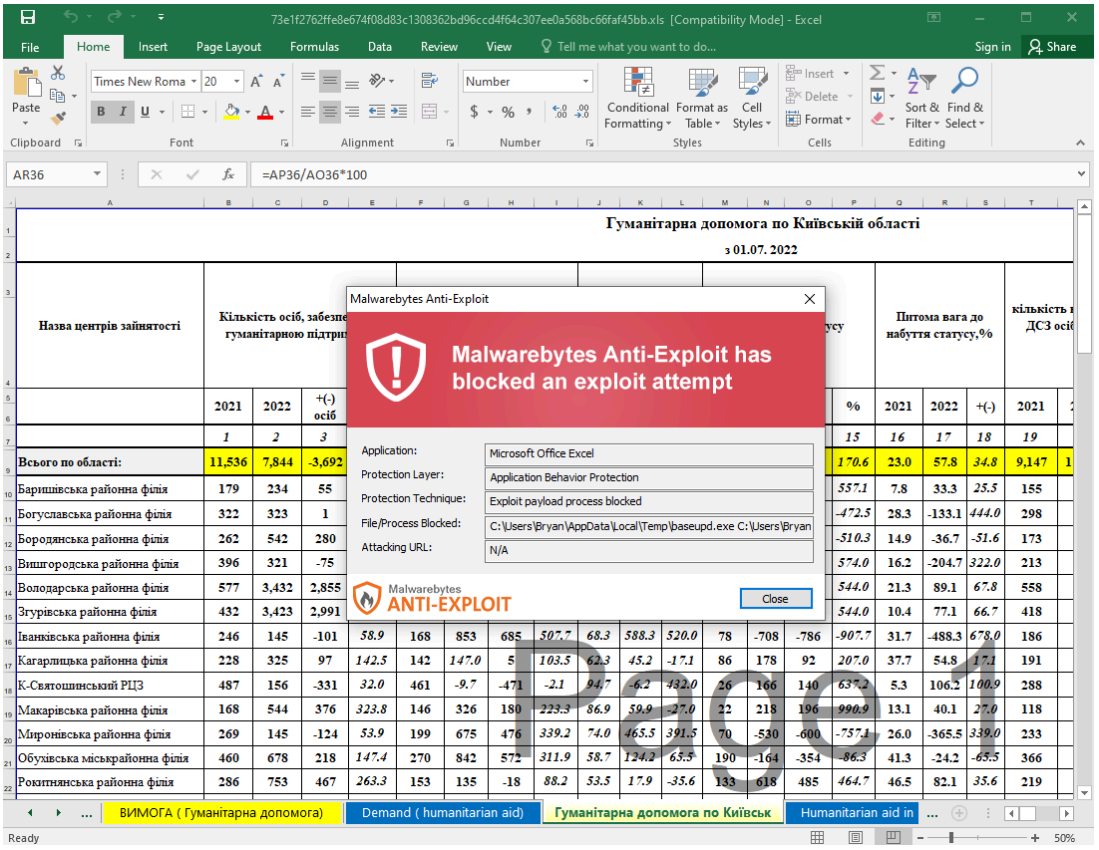
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

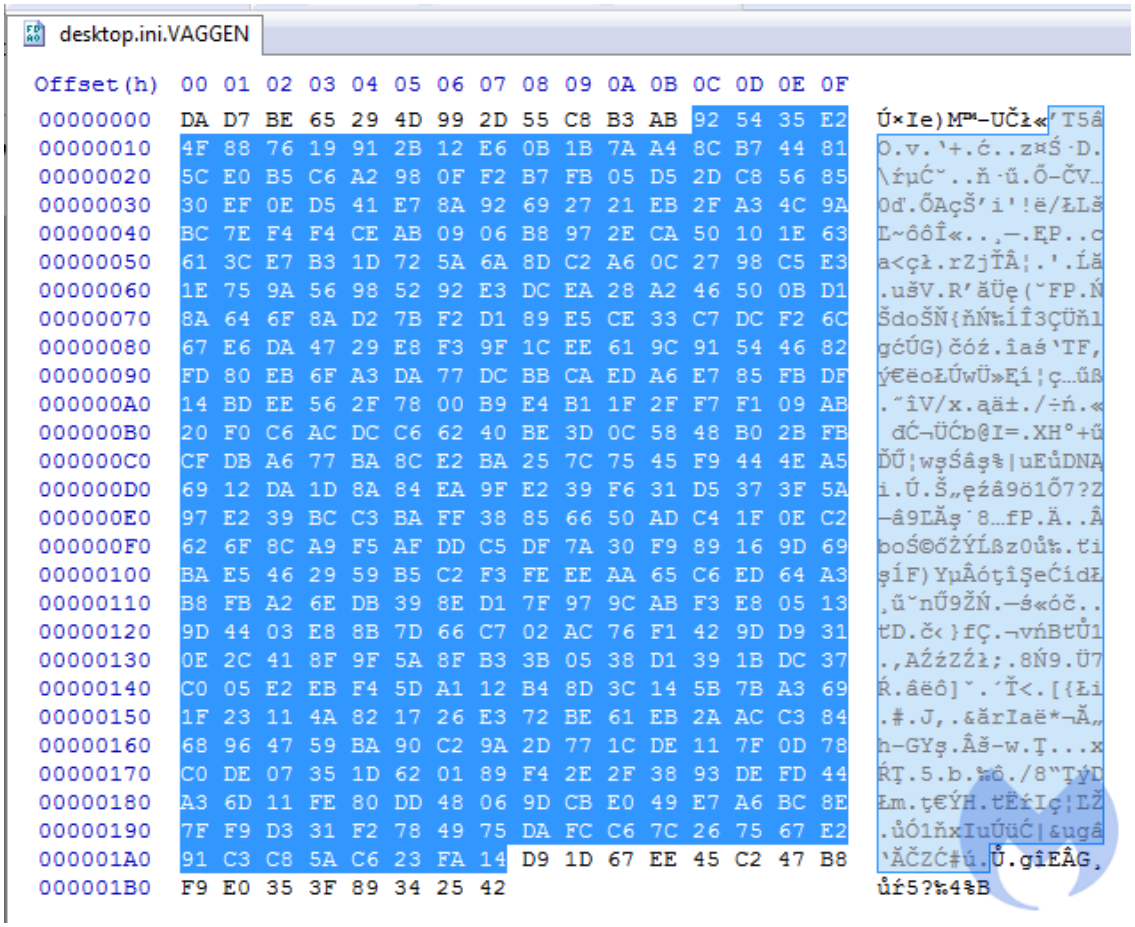
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

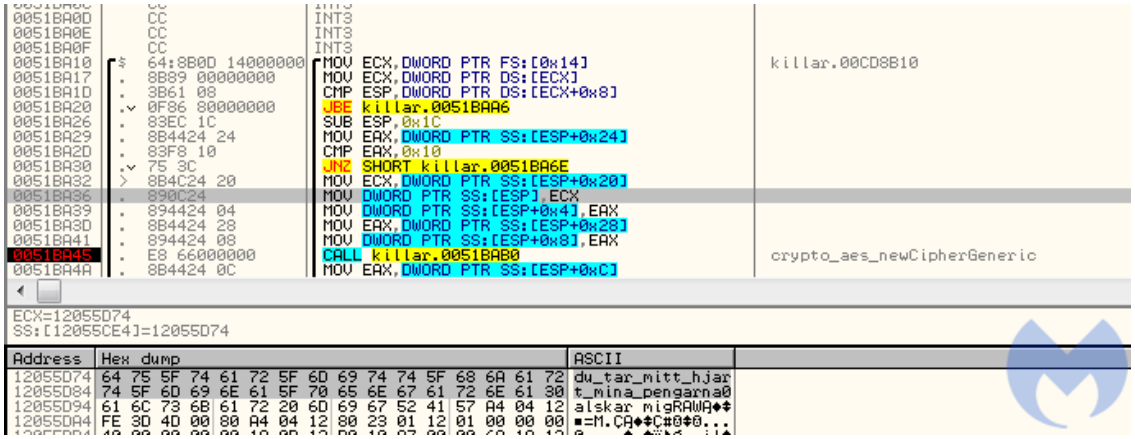


The content of the output file (with .VAGGEN extension) contains:

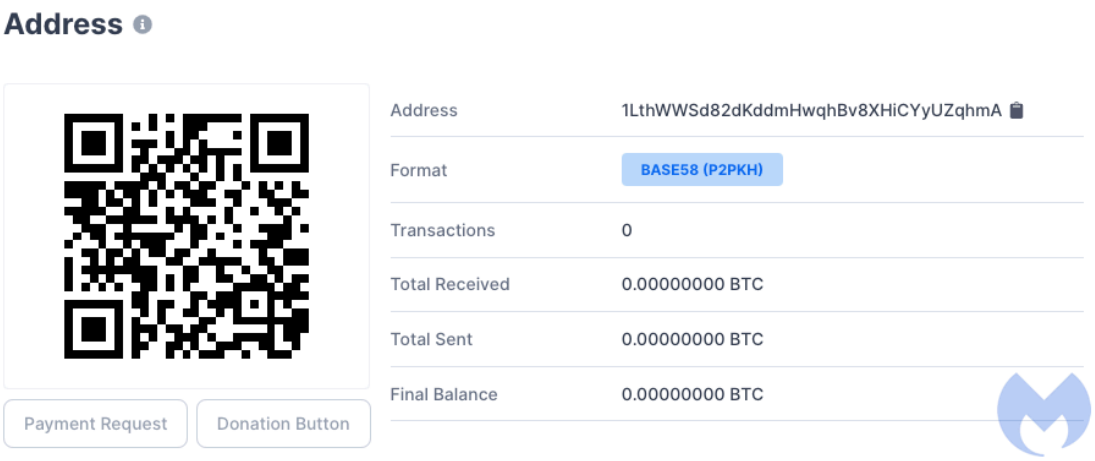
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

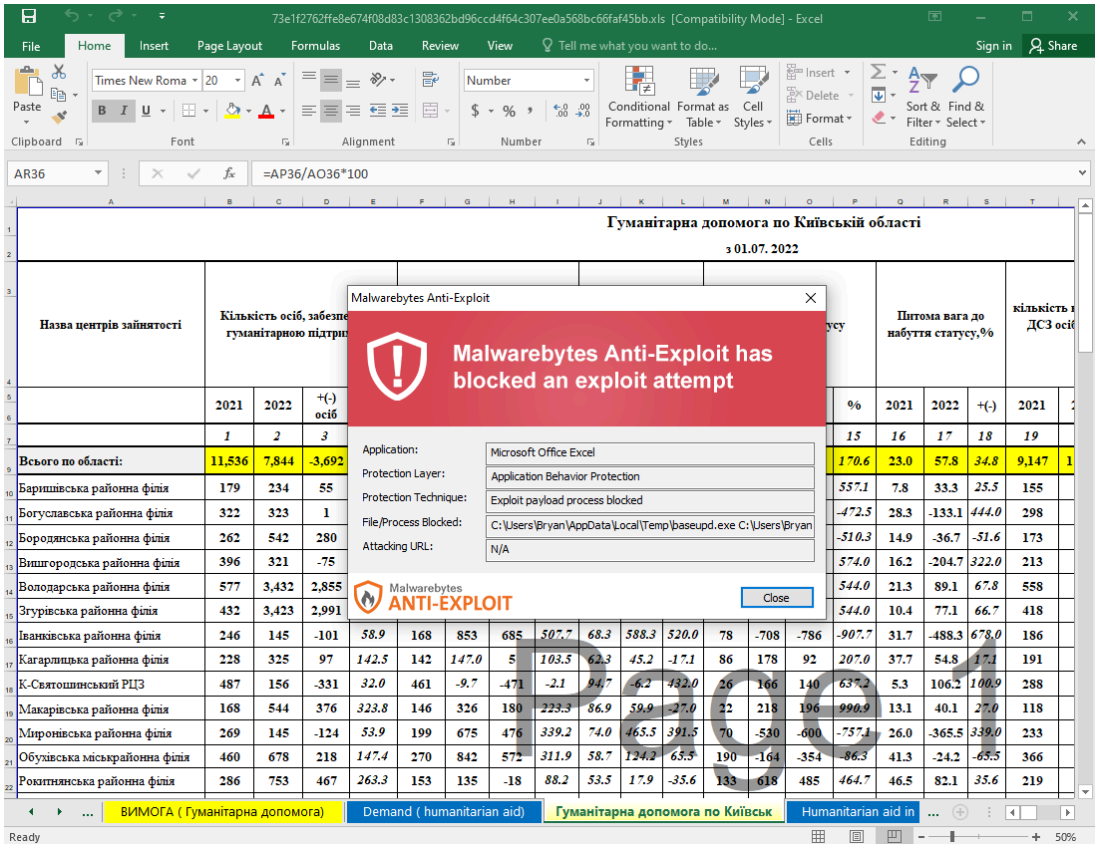
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

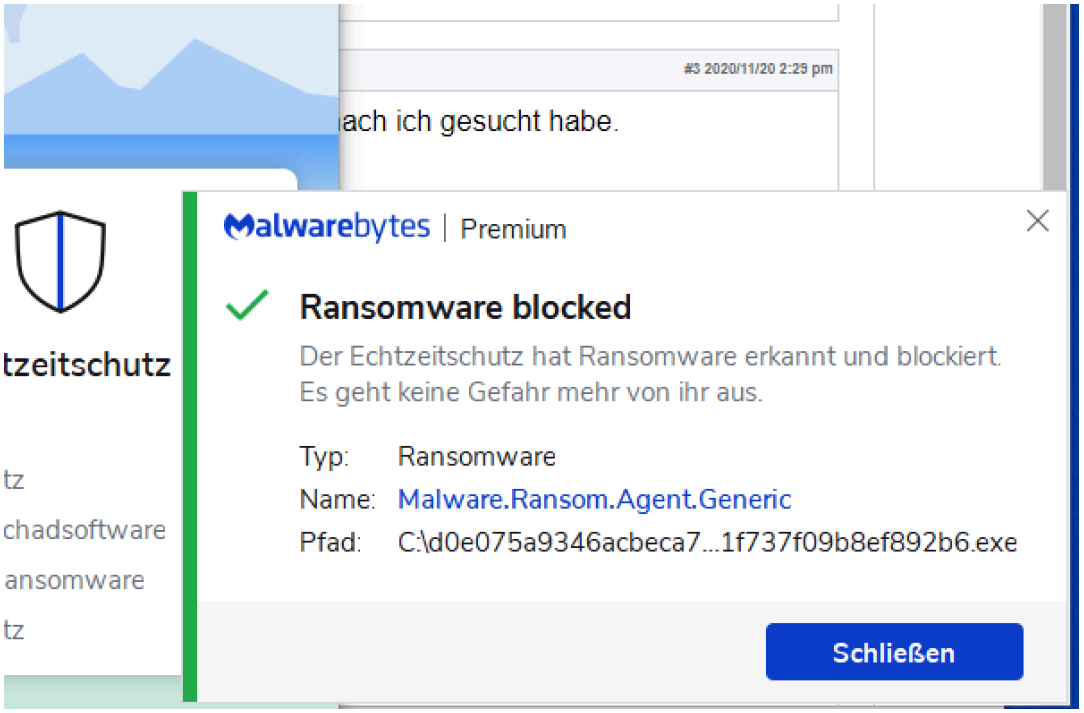
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

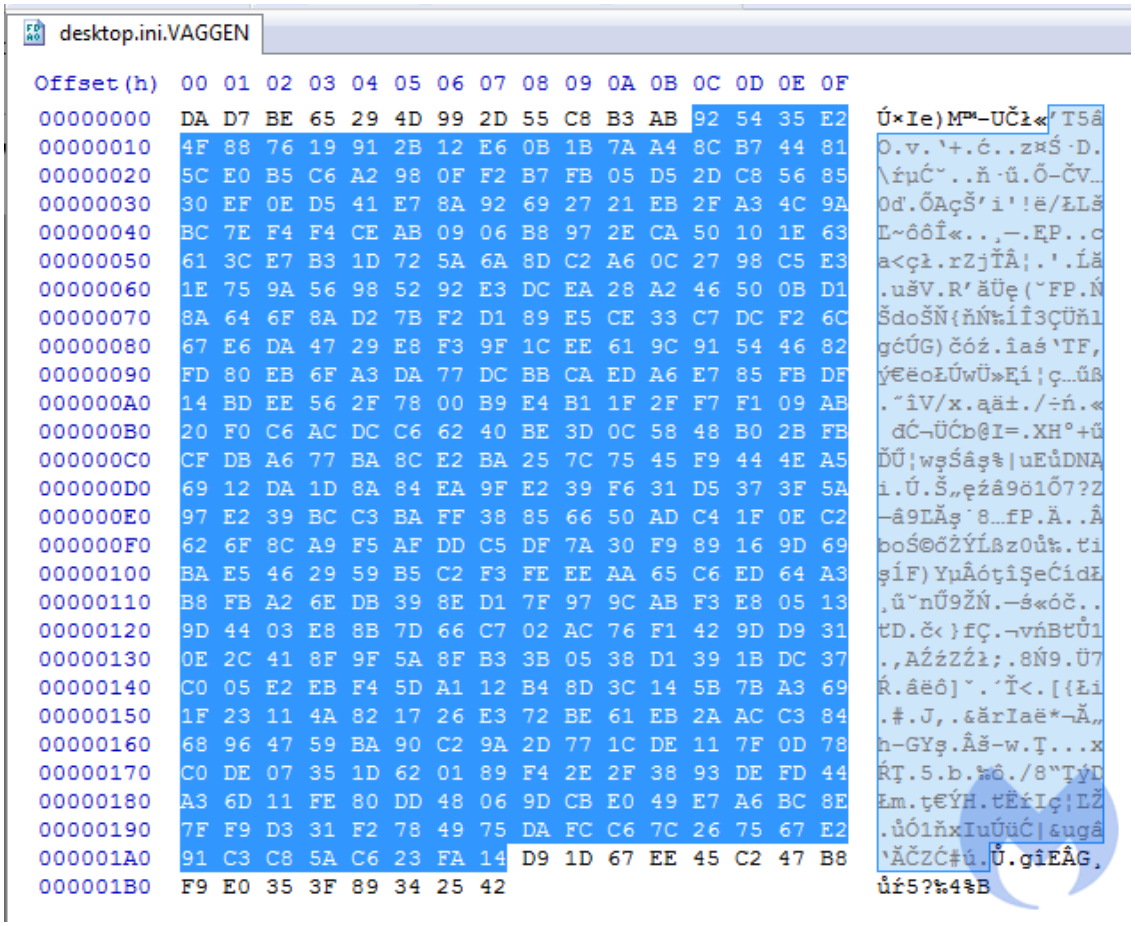
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

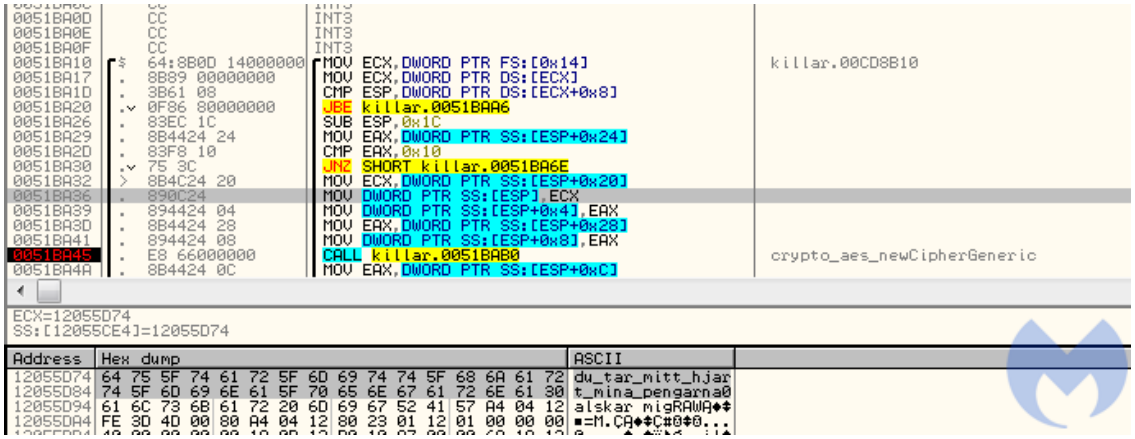


The content of the output file (with .VAGGEN extension) contains:

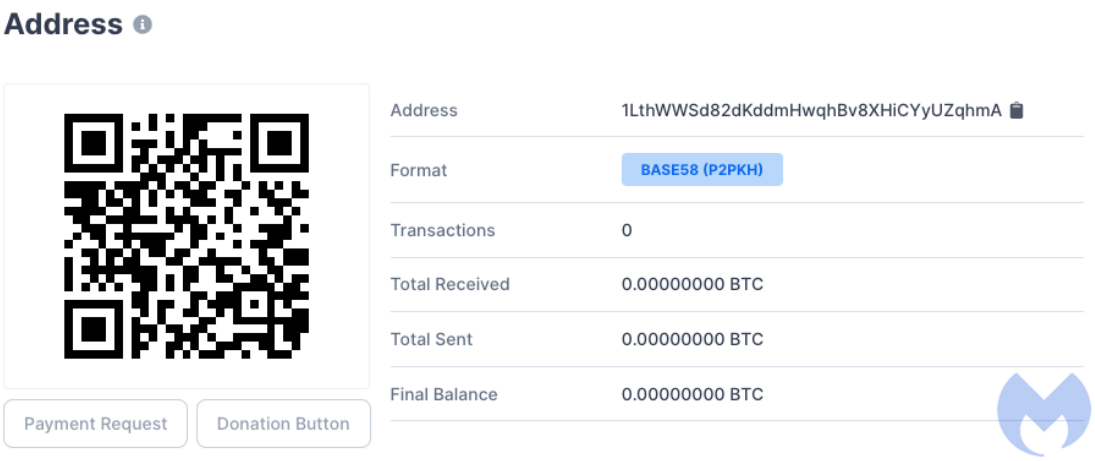
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

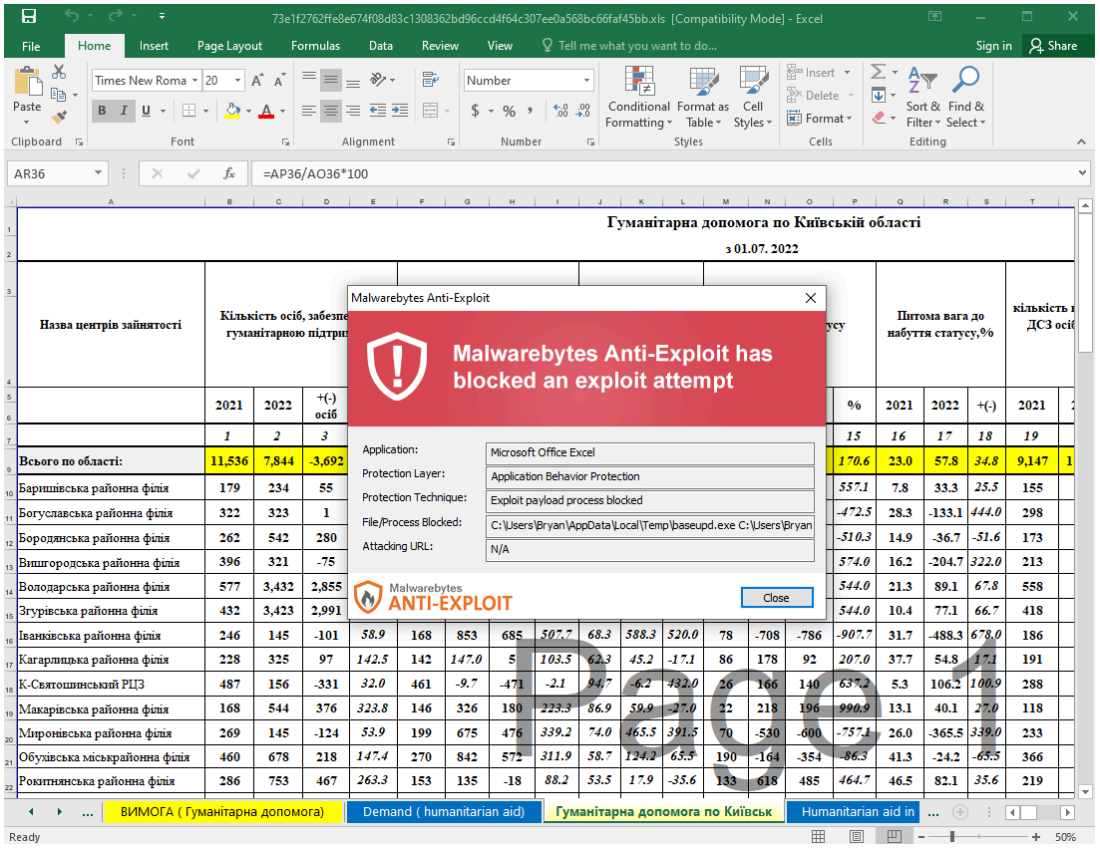
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

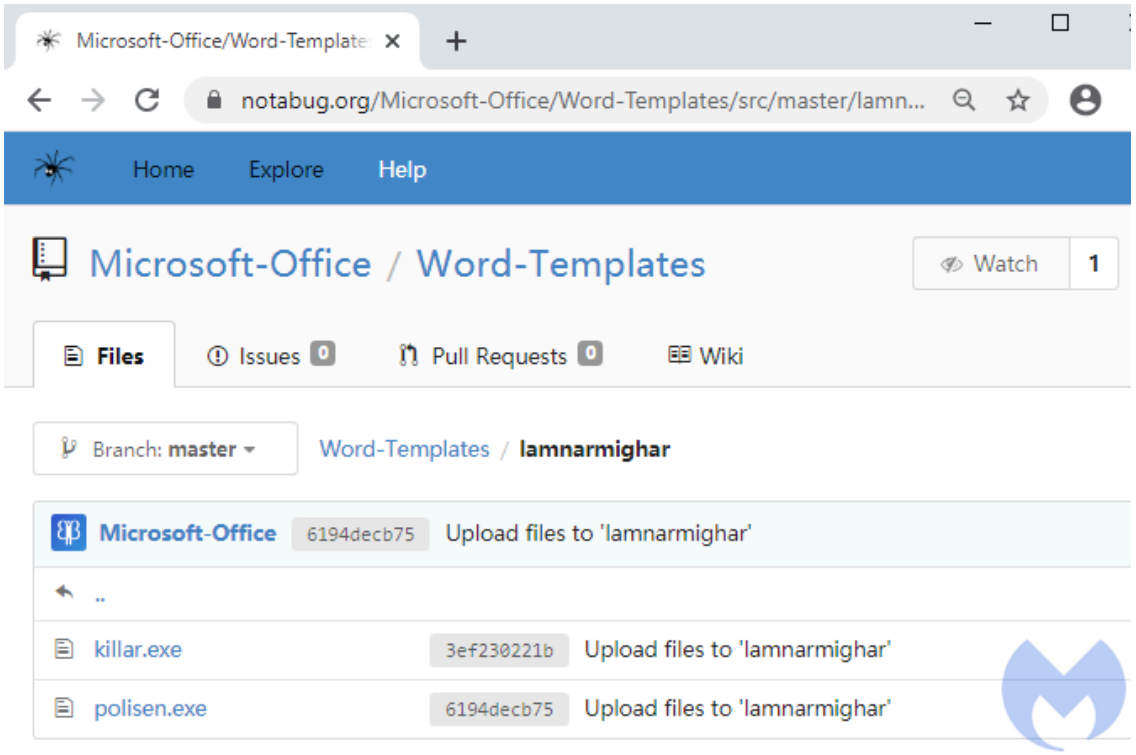
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



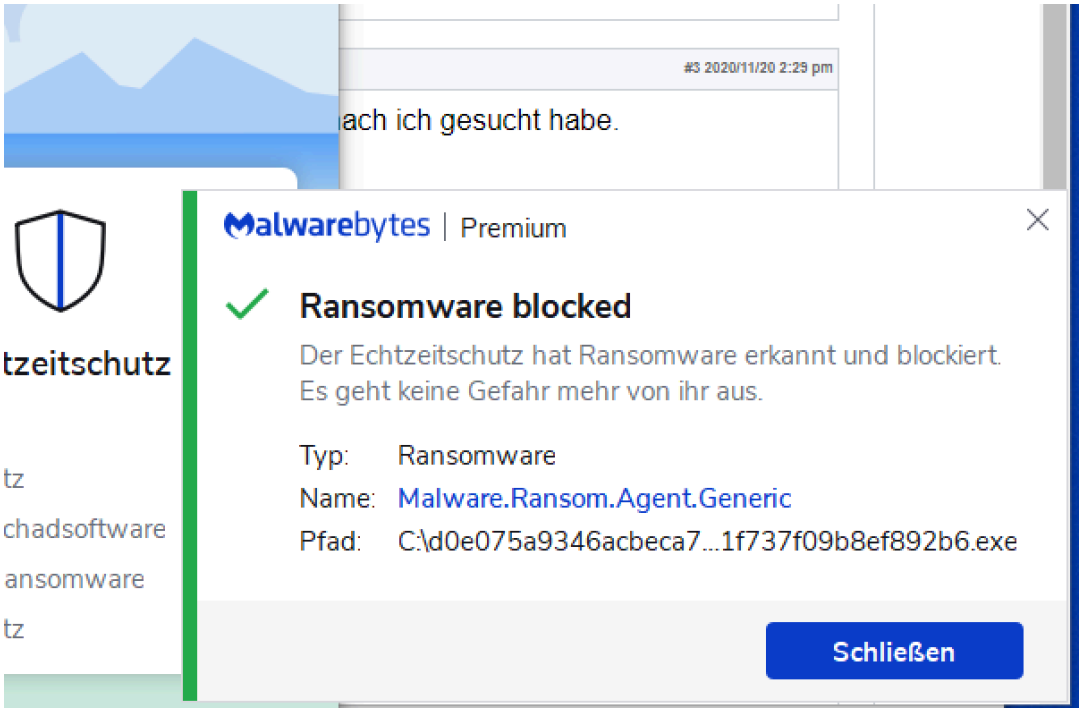
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

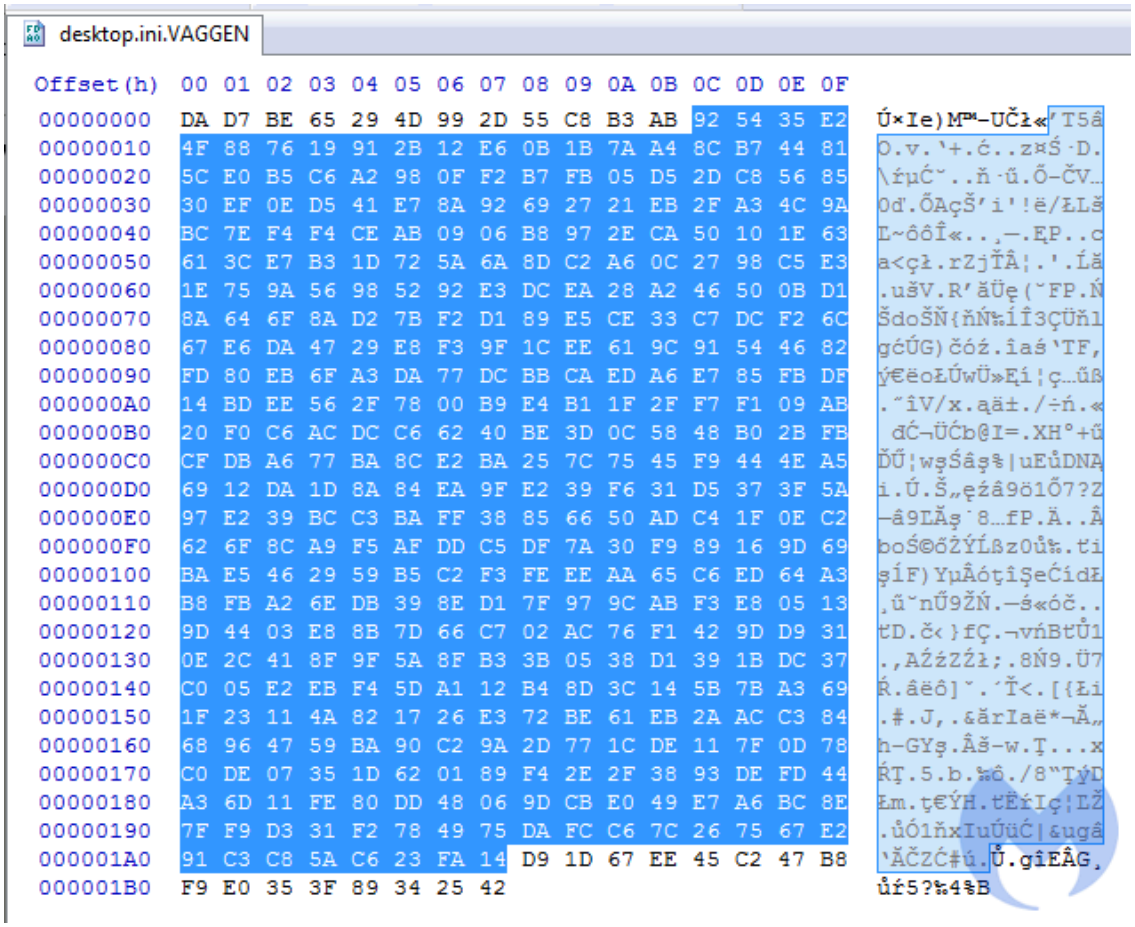
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

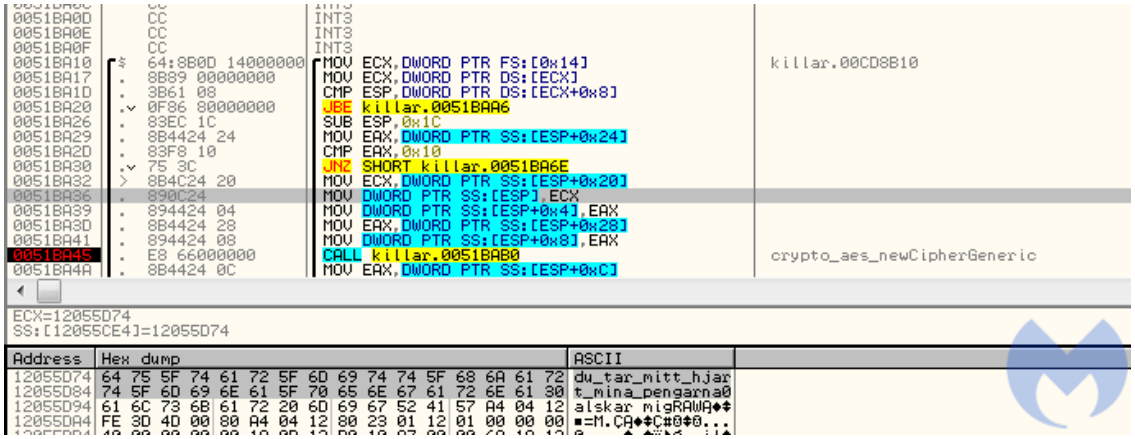


The content of the output file (with .VAGGEN extension) contains:

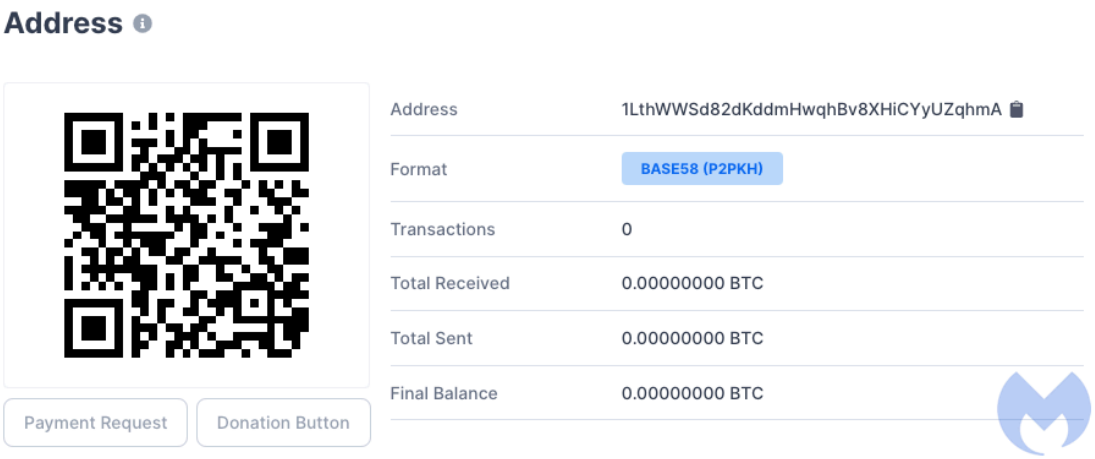
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

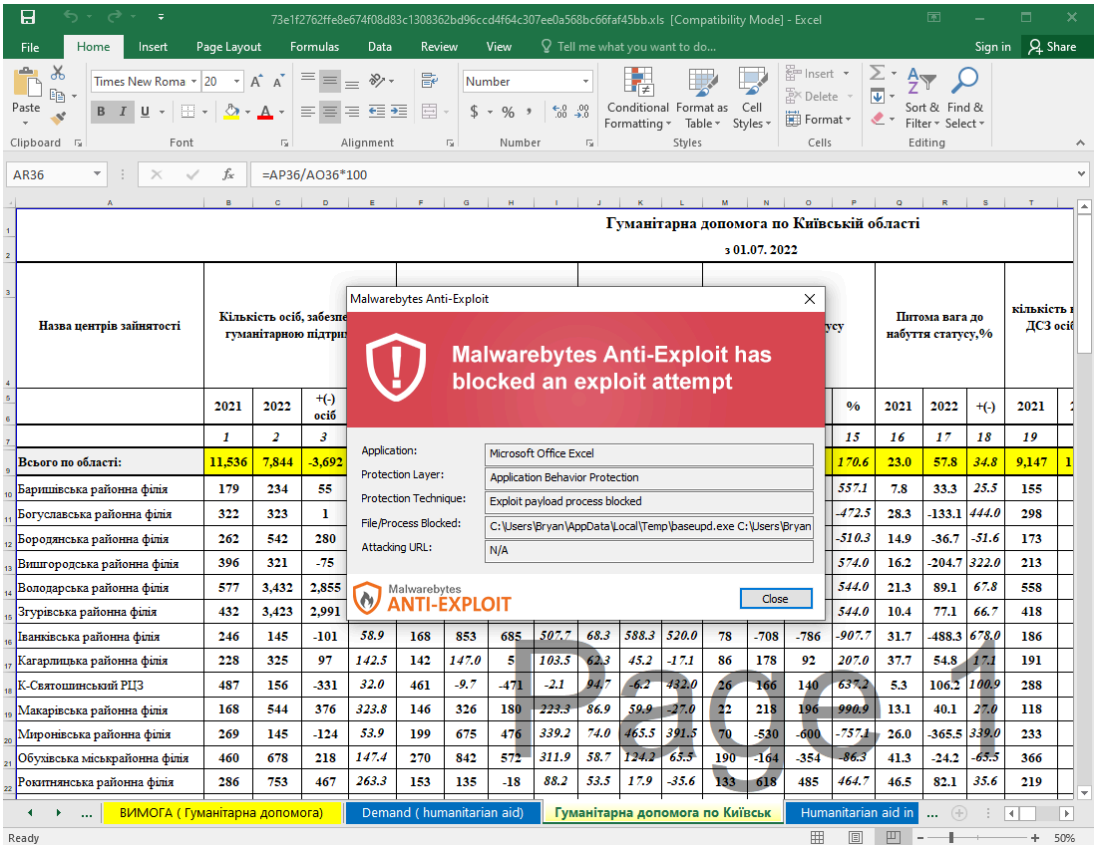
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

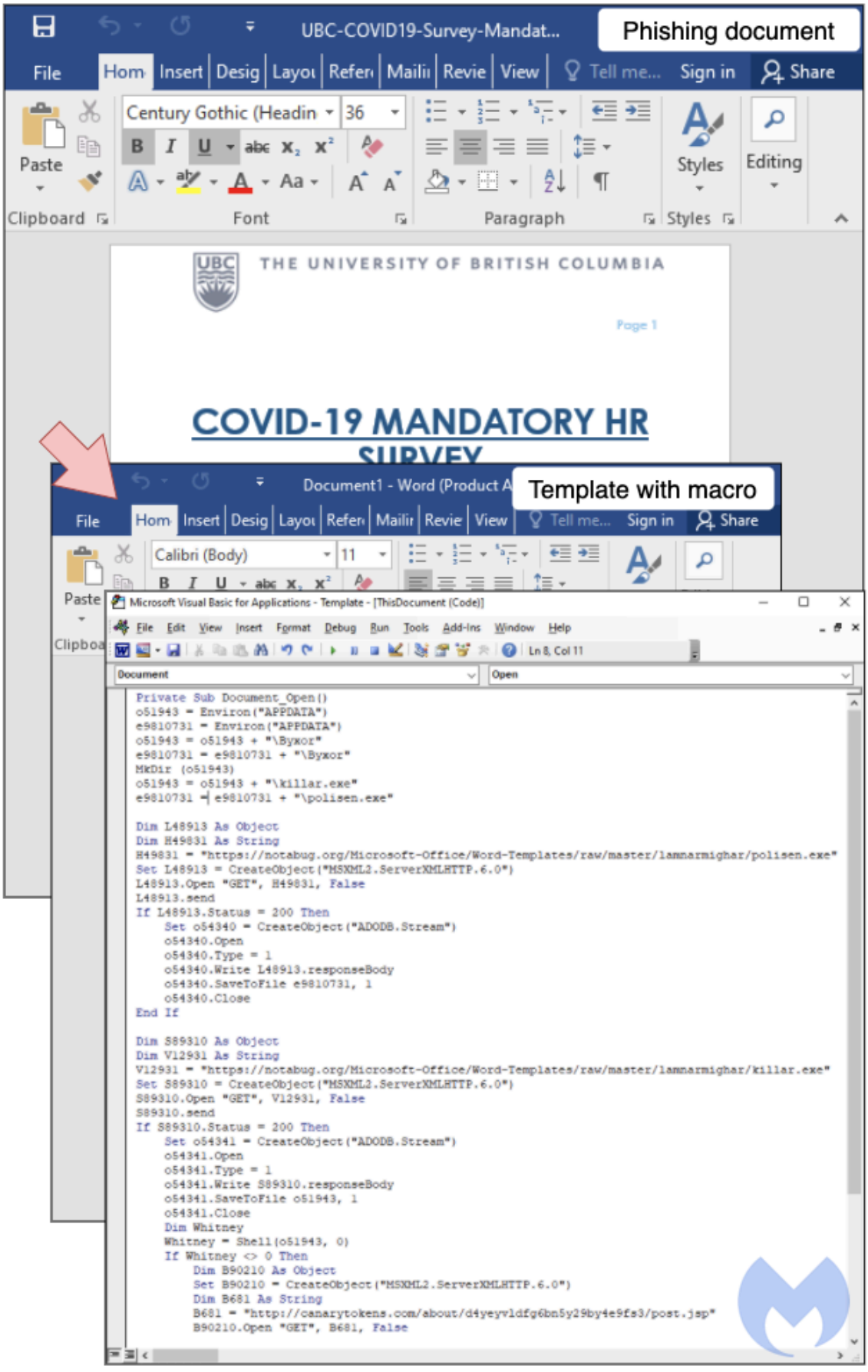
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

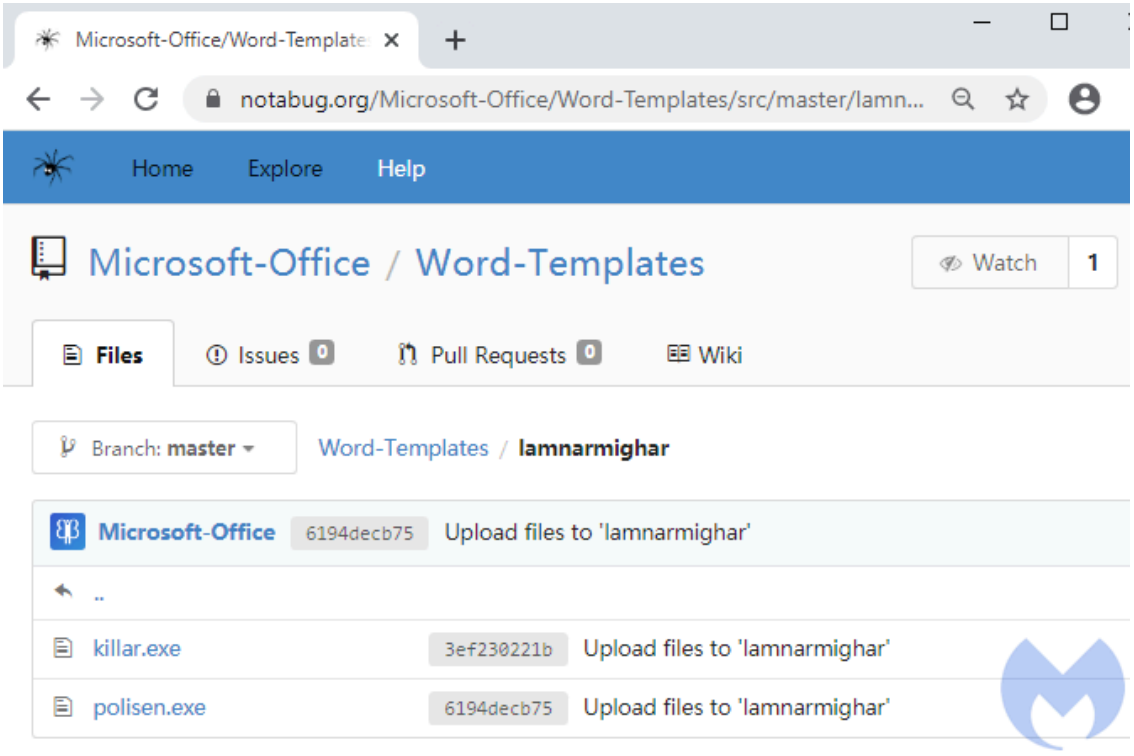
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



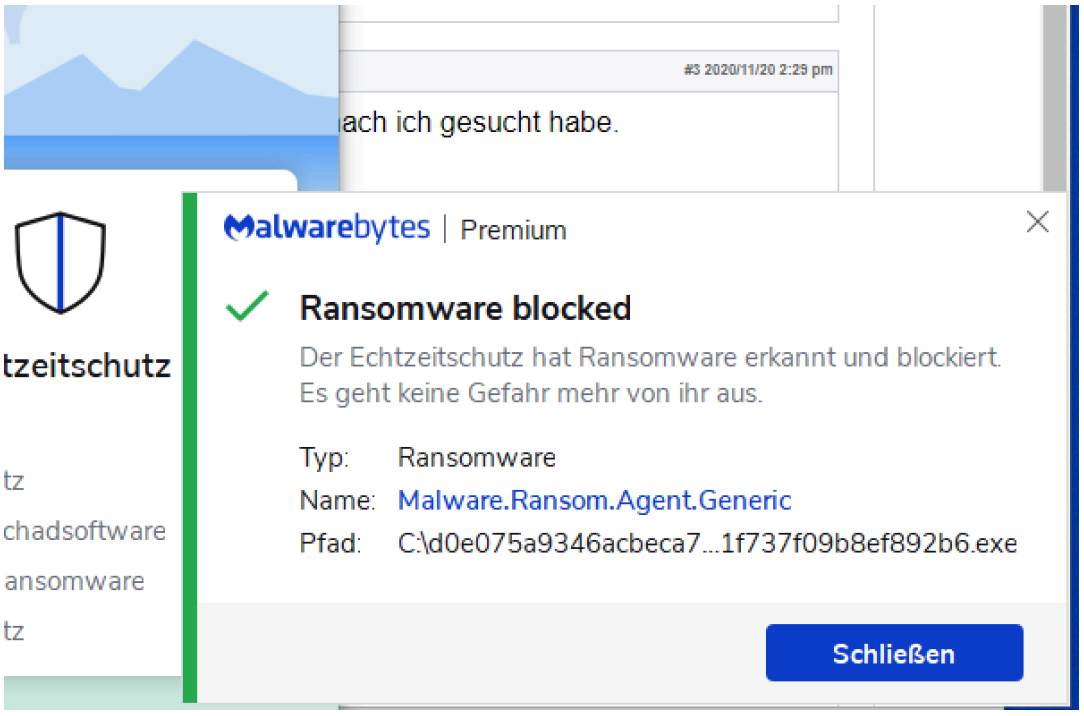
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

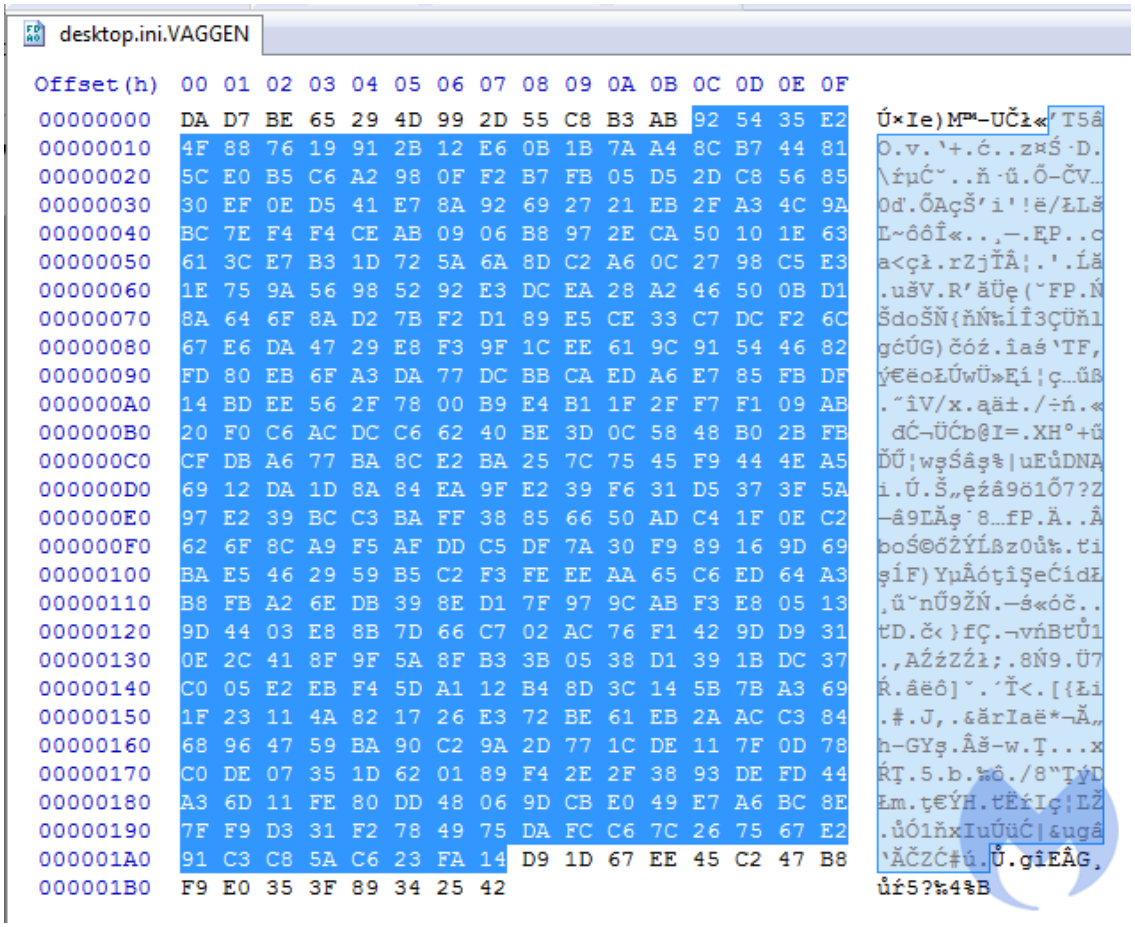
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

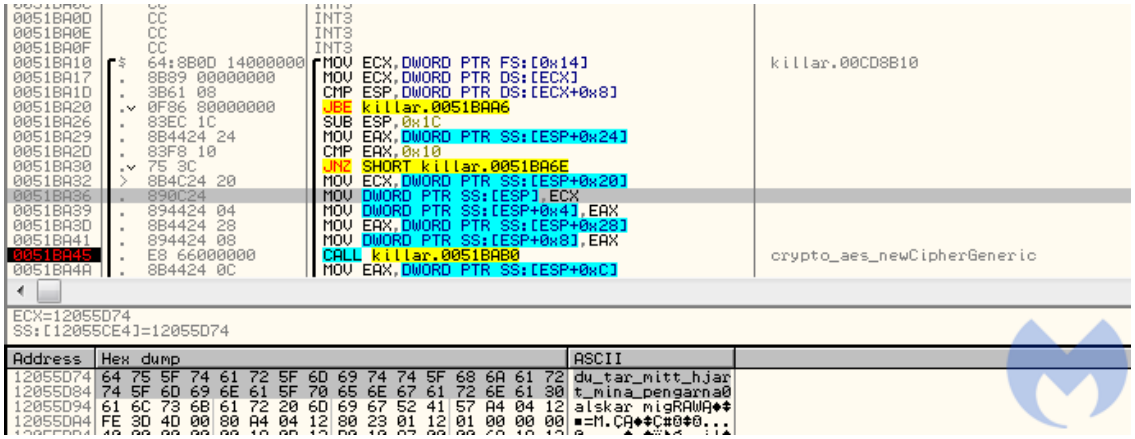


The content of the output file (with .VAGGEN extension) contains:

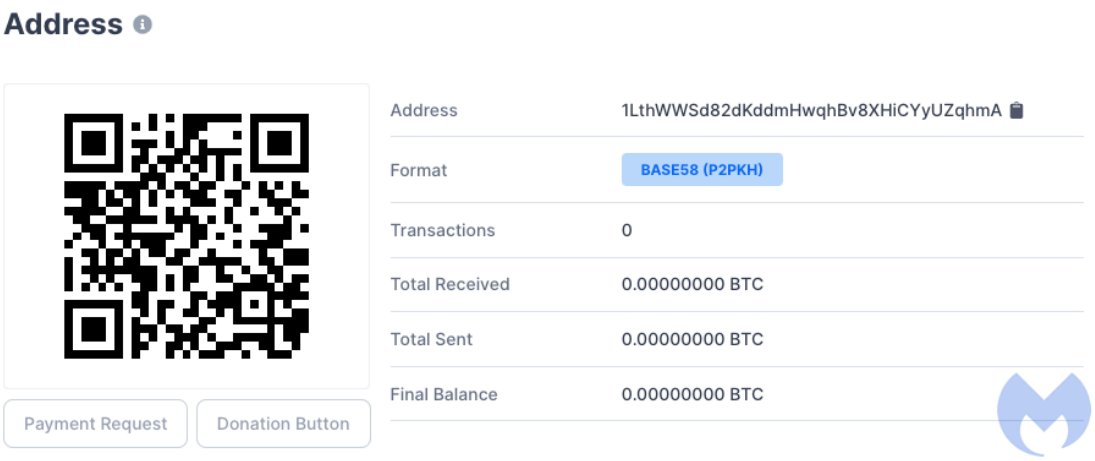
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

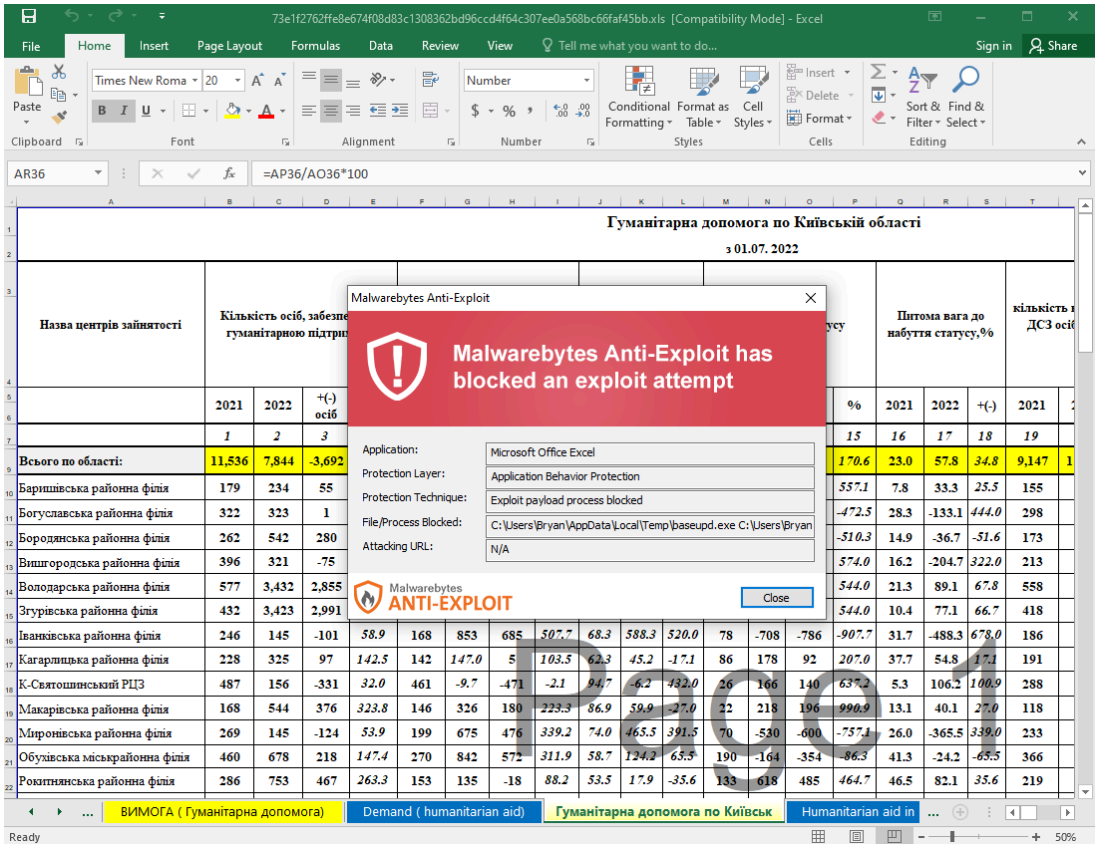
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

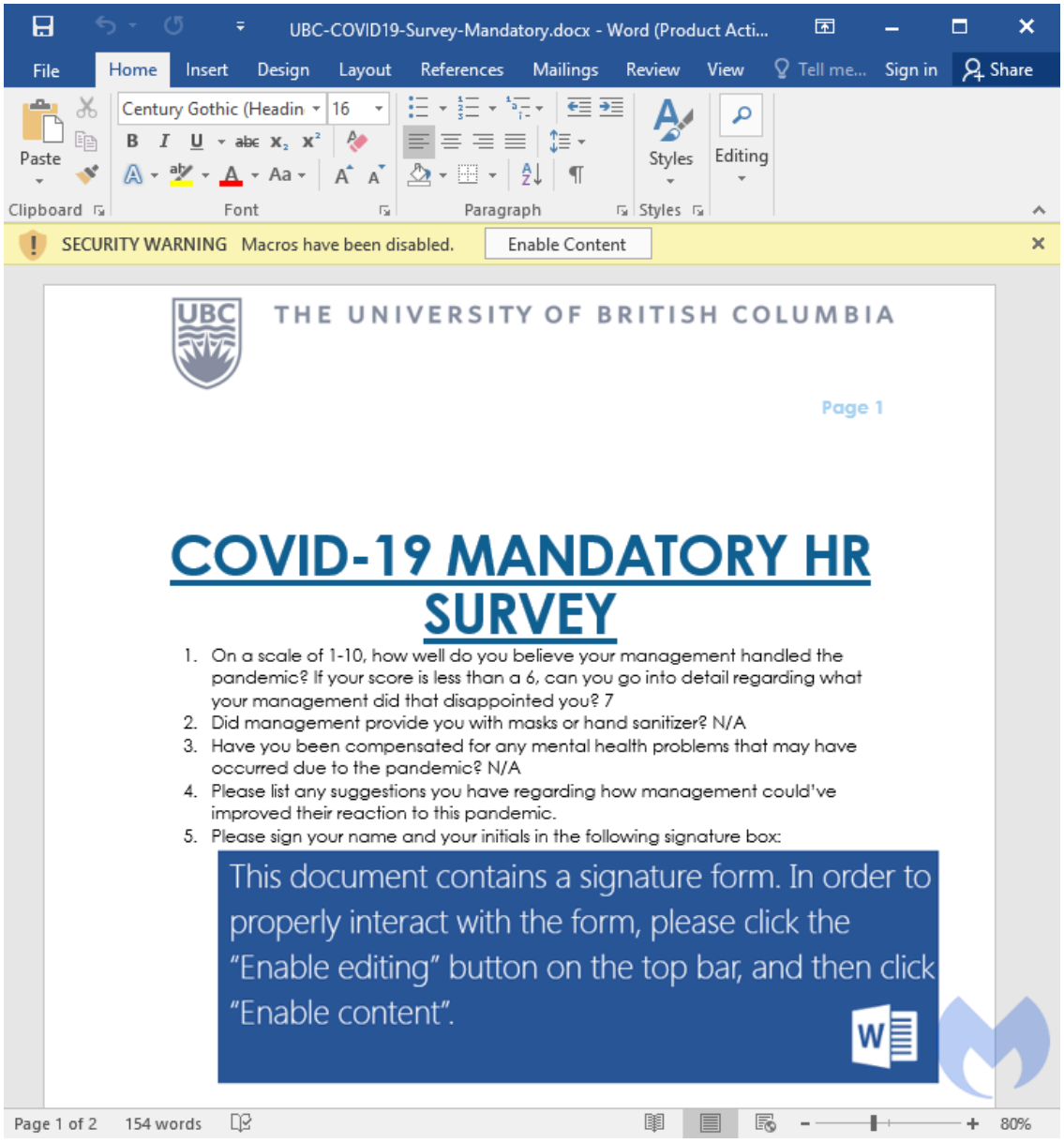
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

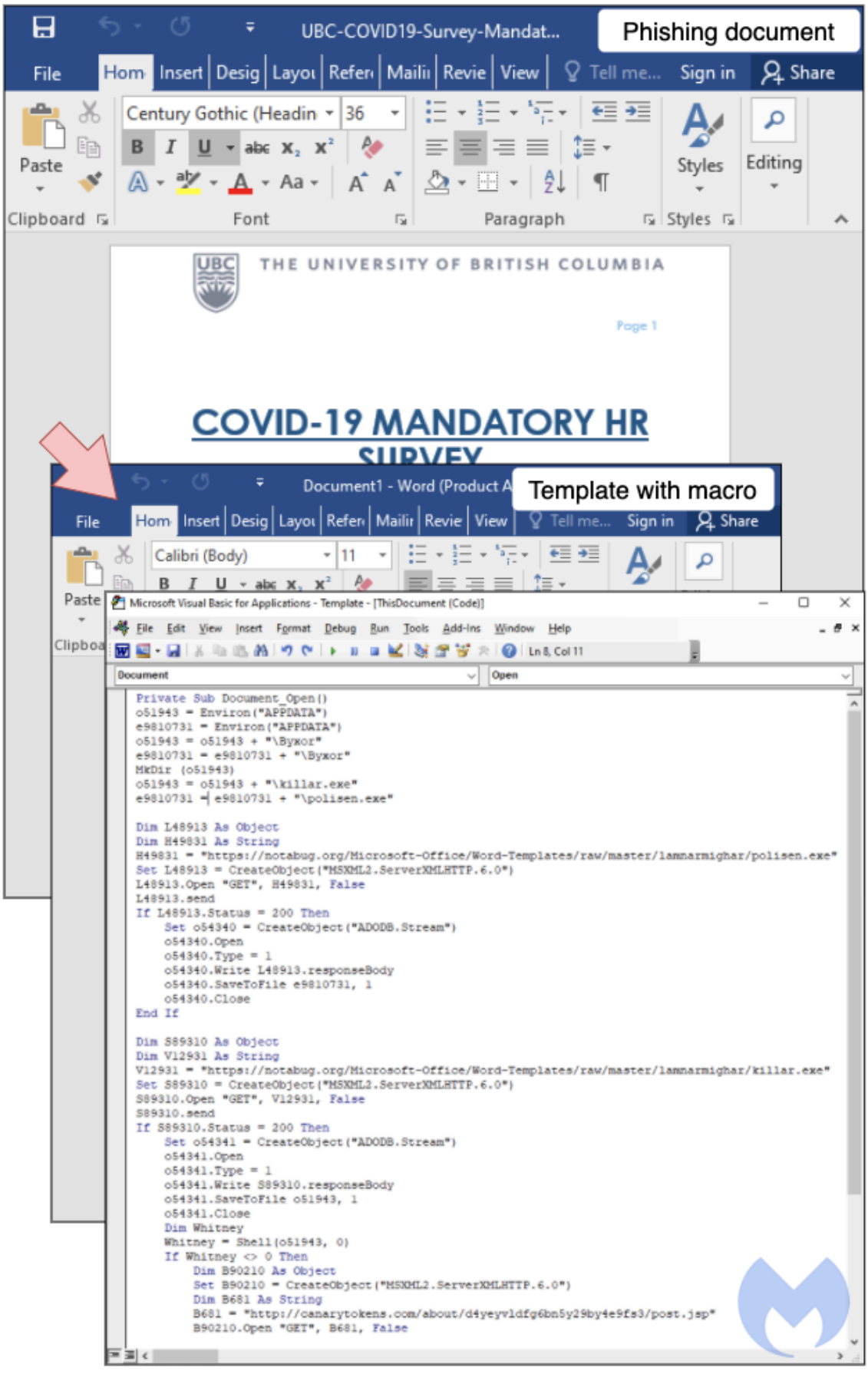
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

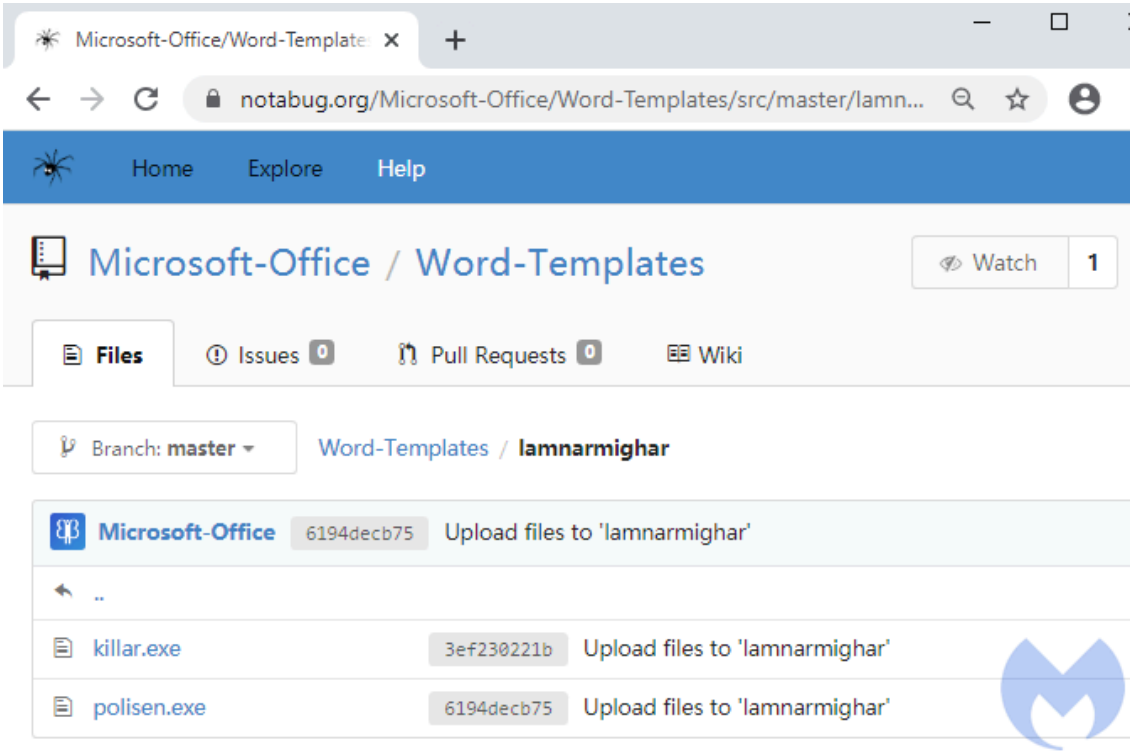
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



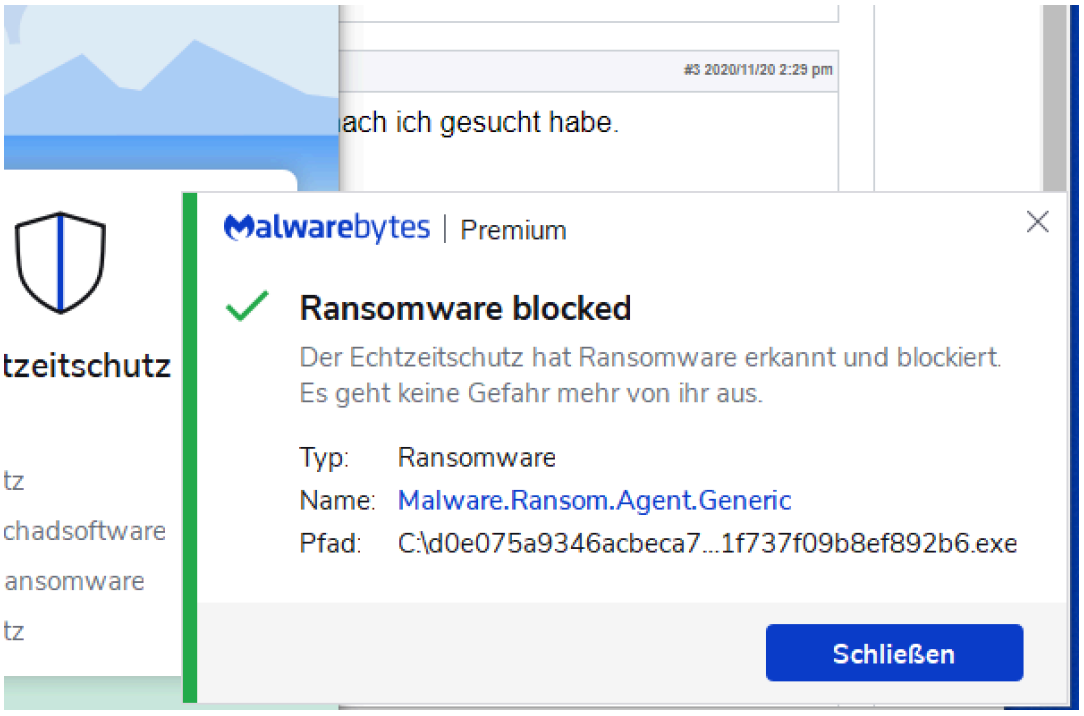
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

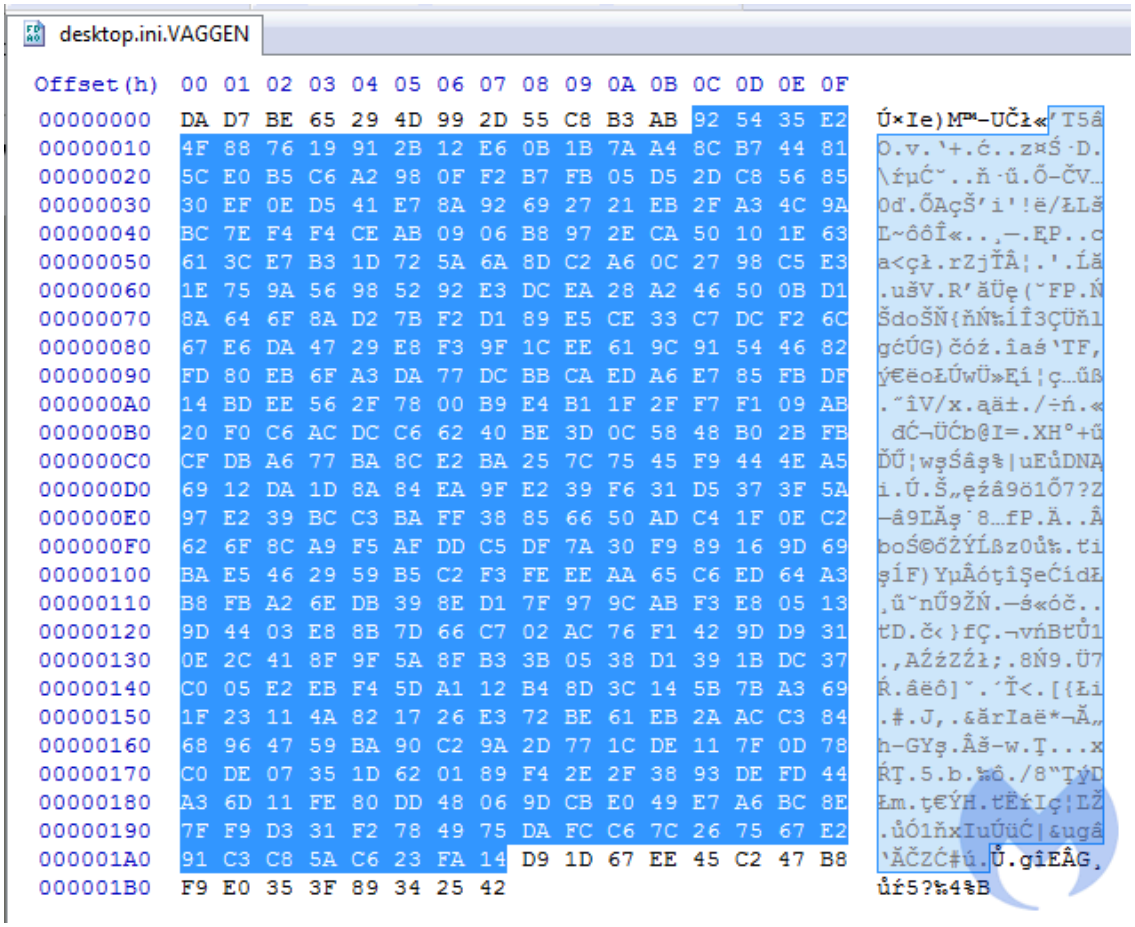
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

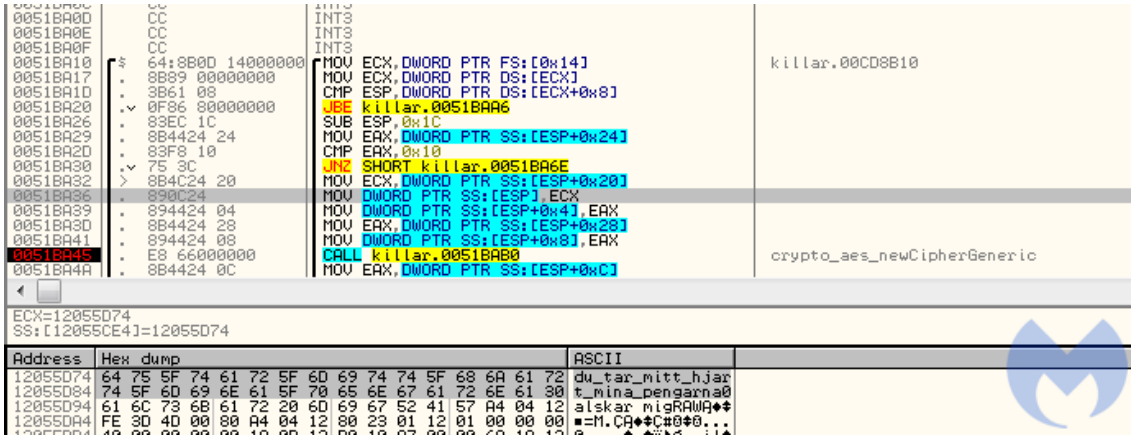


The content of the output file (with .VAGGEN extension) contains:

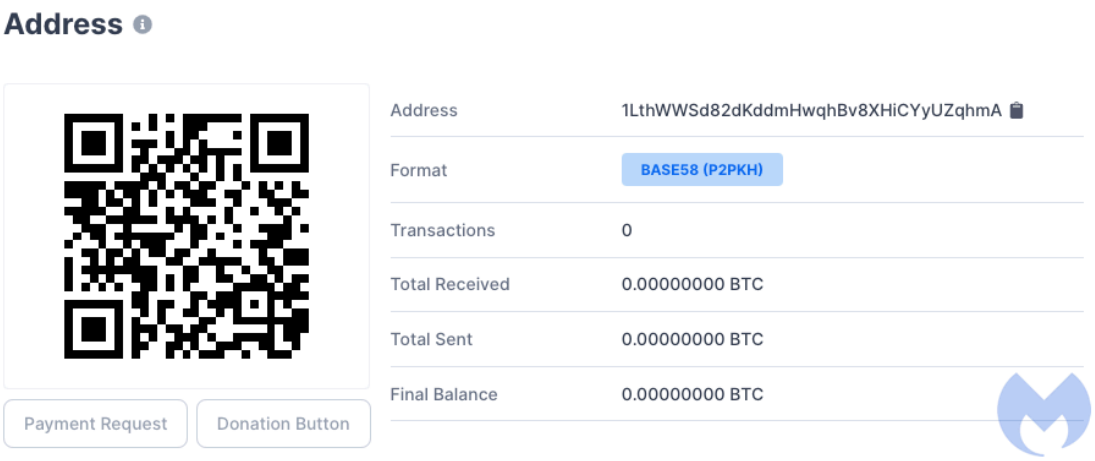
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

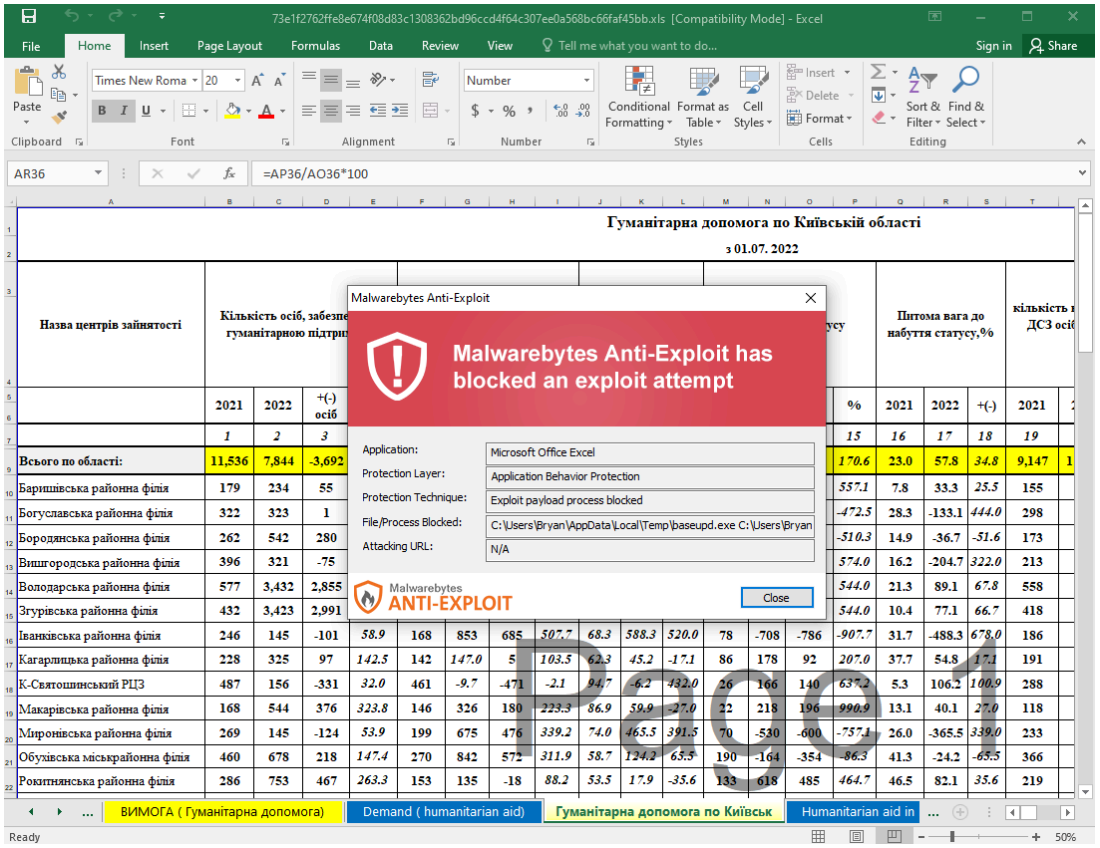
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

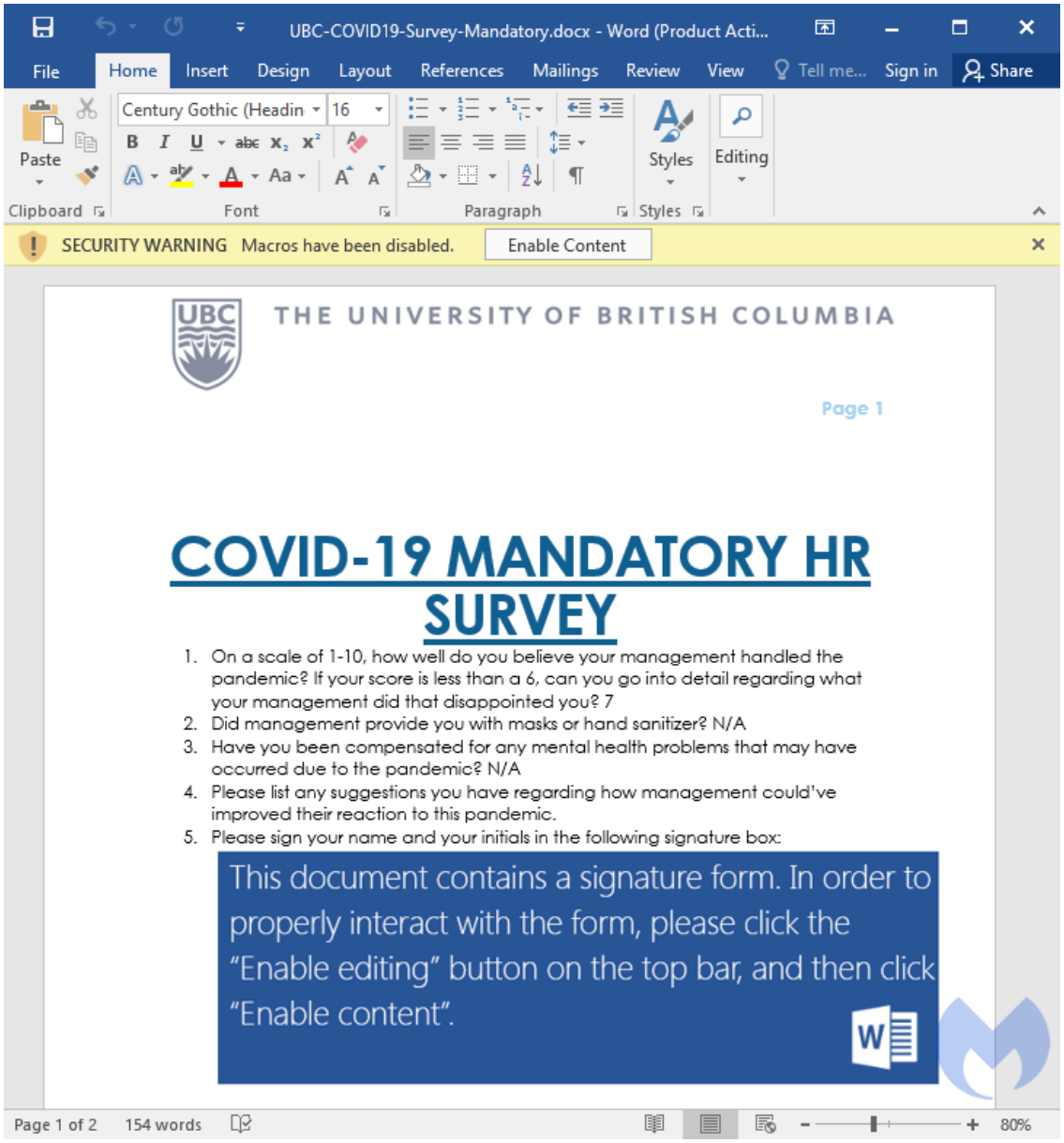
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

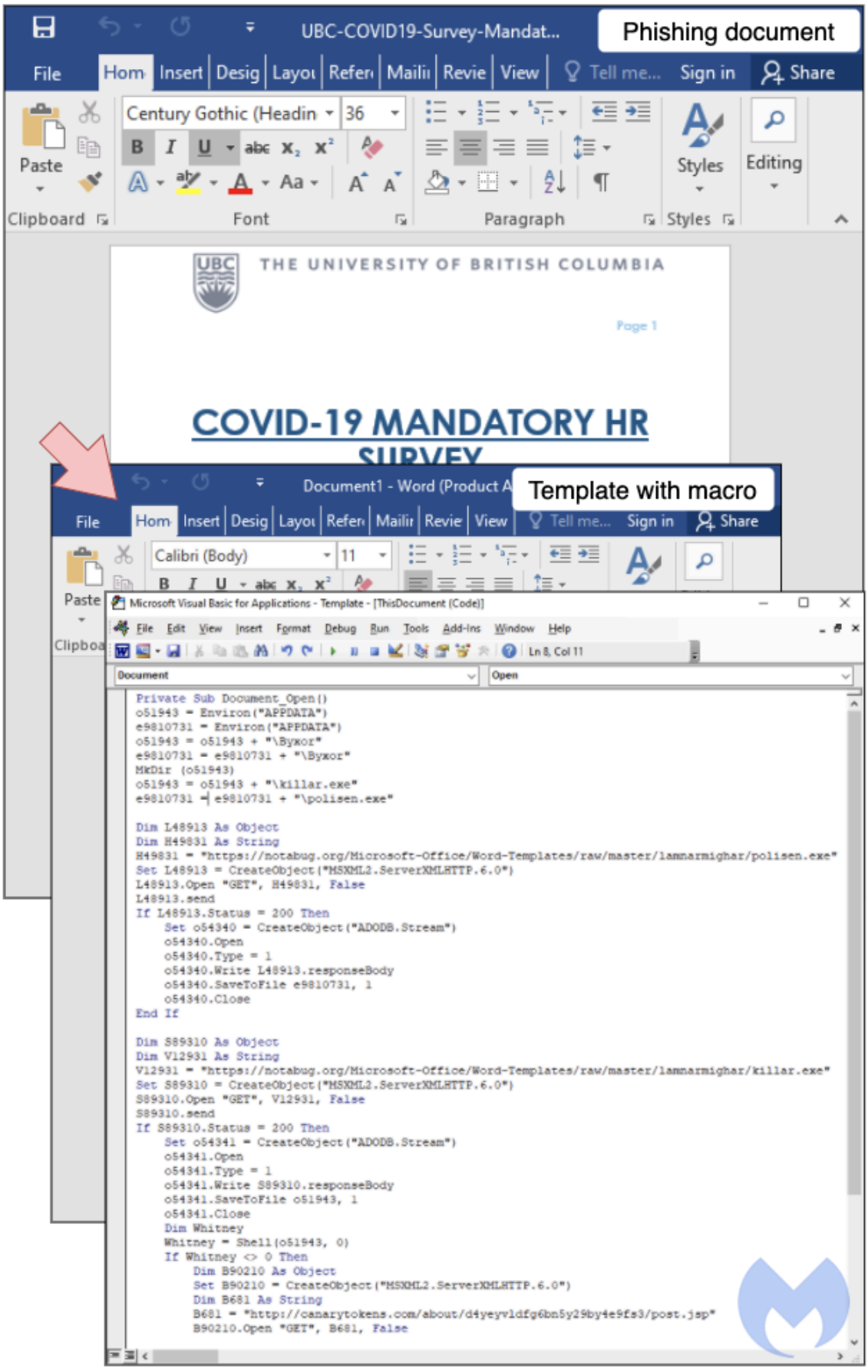
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

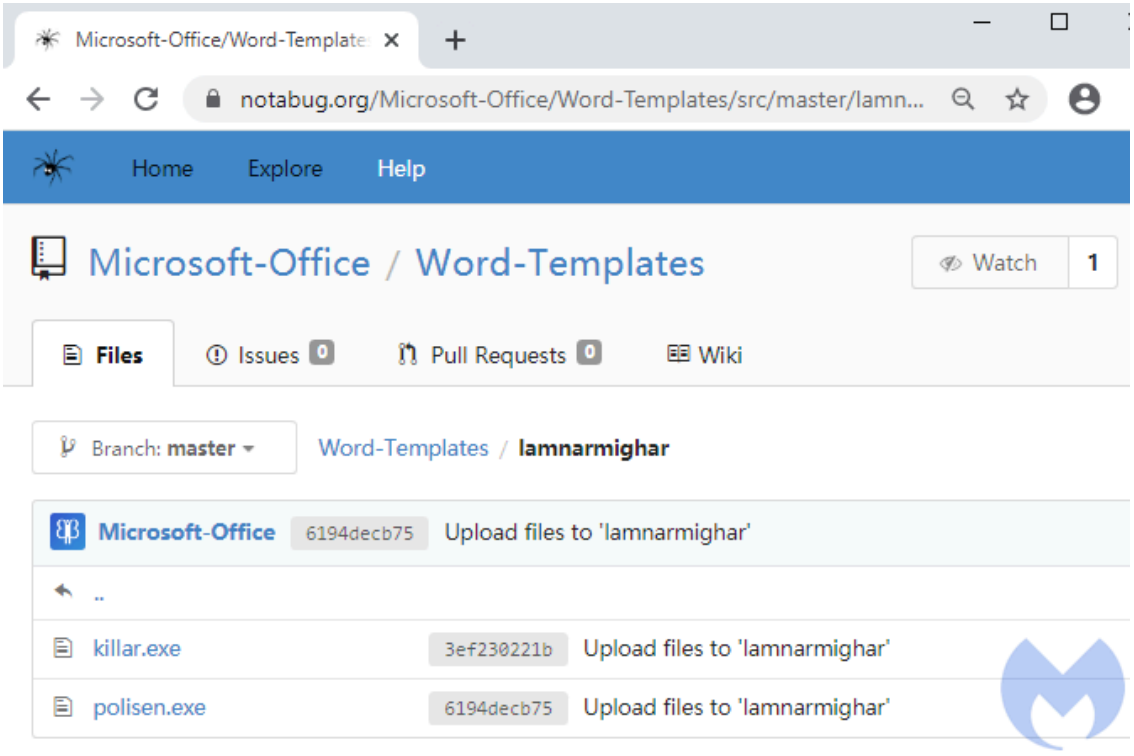
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



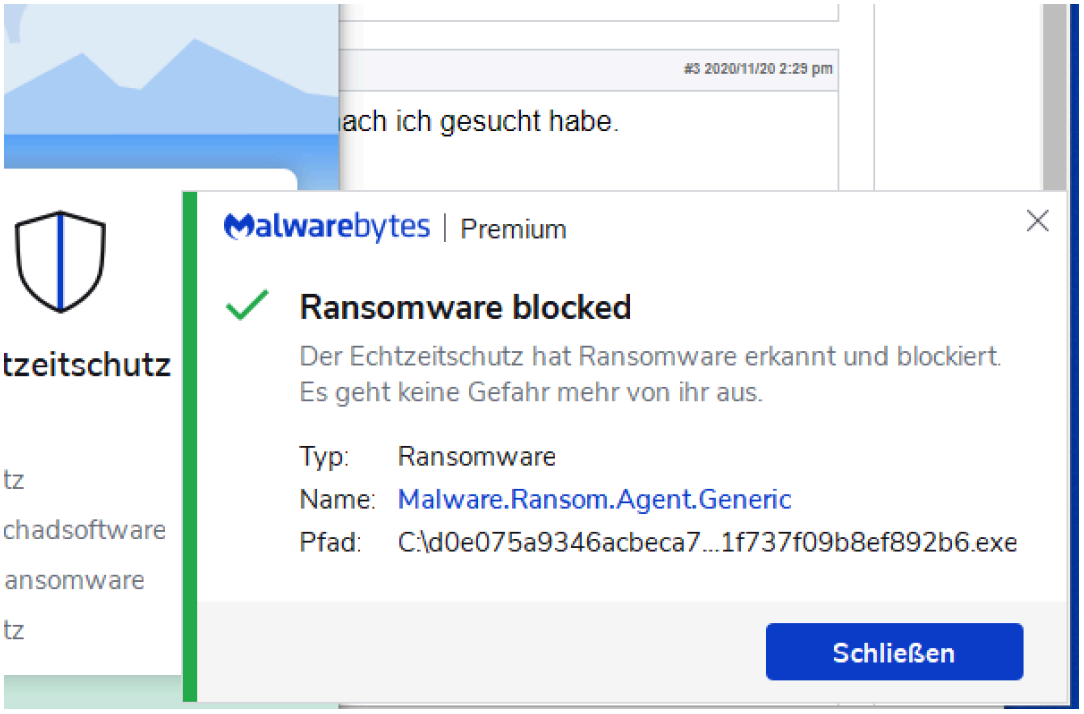
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

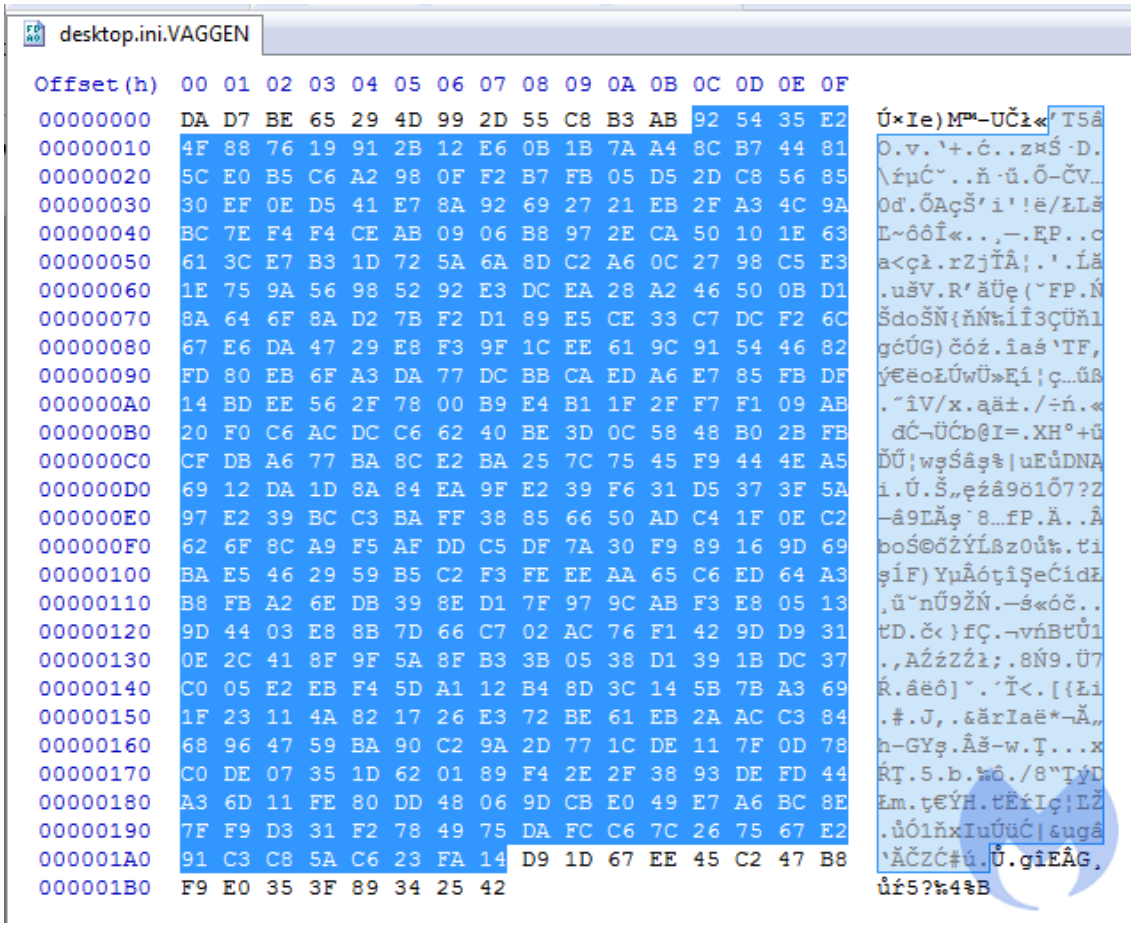
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

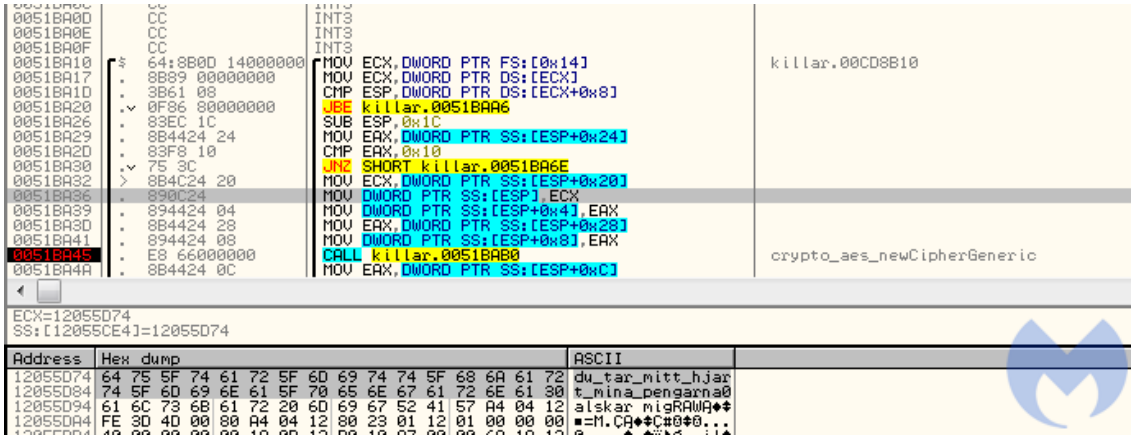


The content of the output file (with .VAGGEN extension) contains:

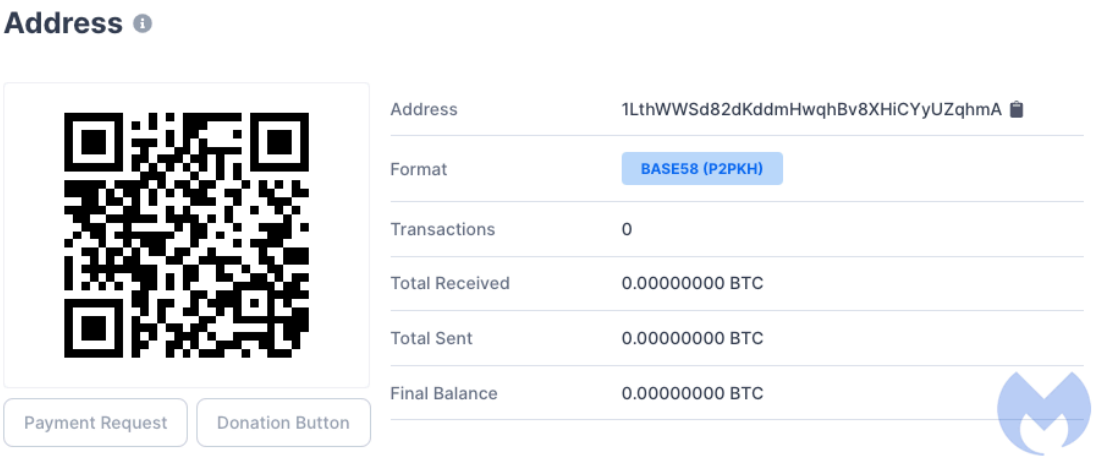
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

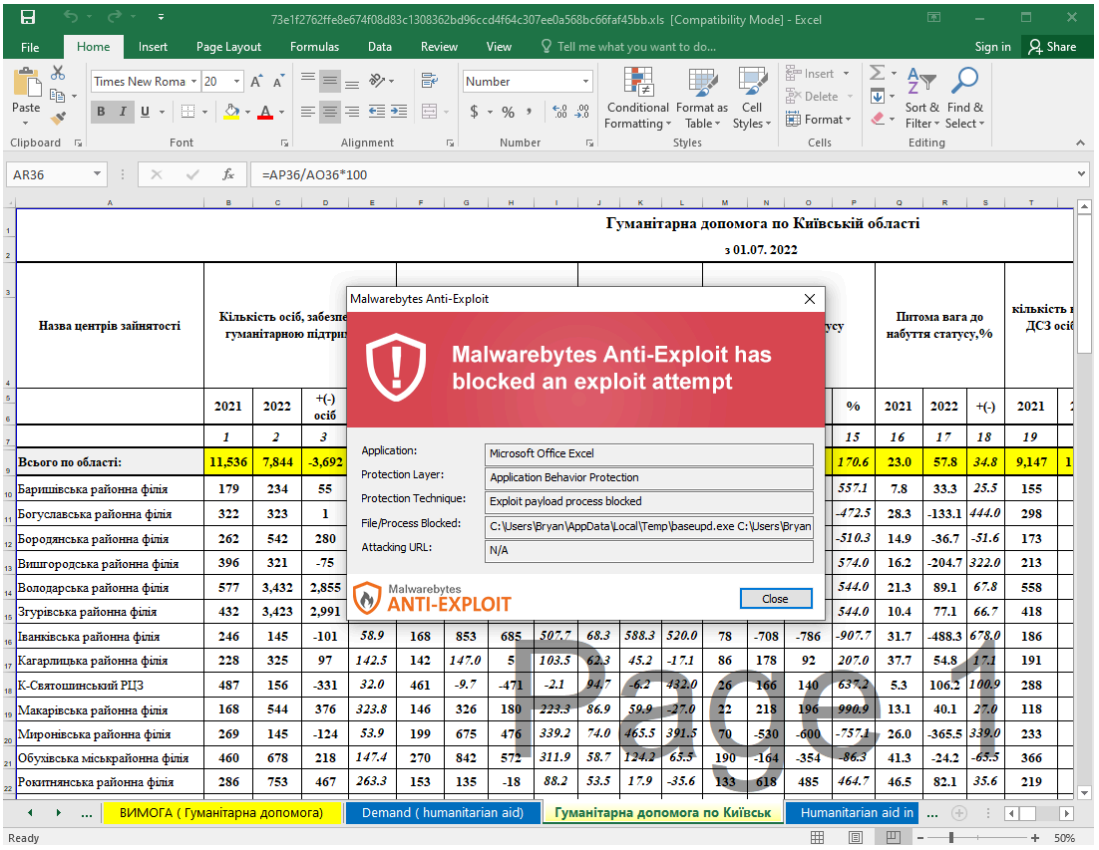
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

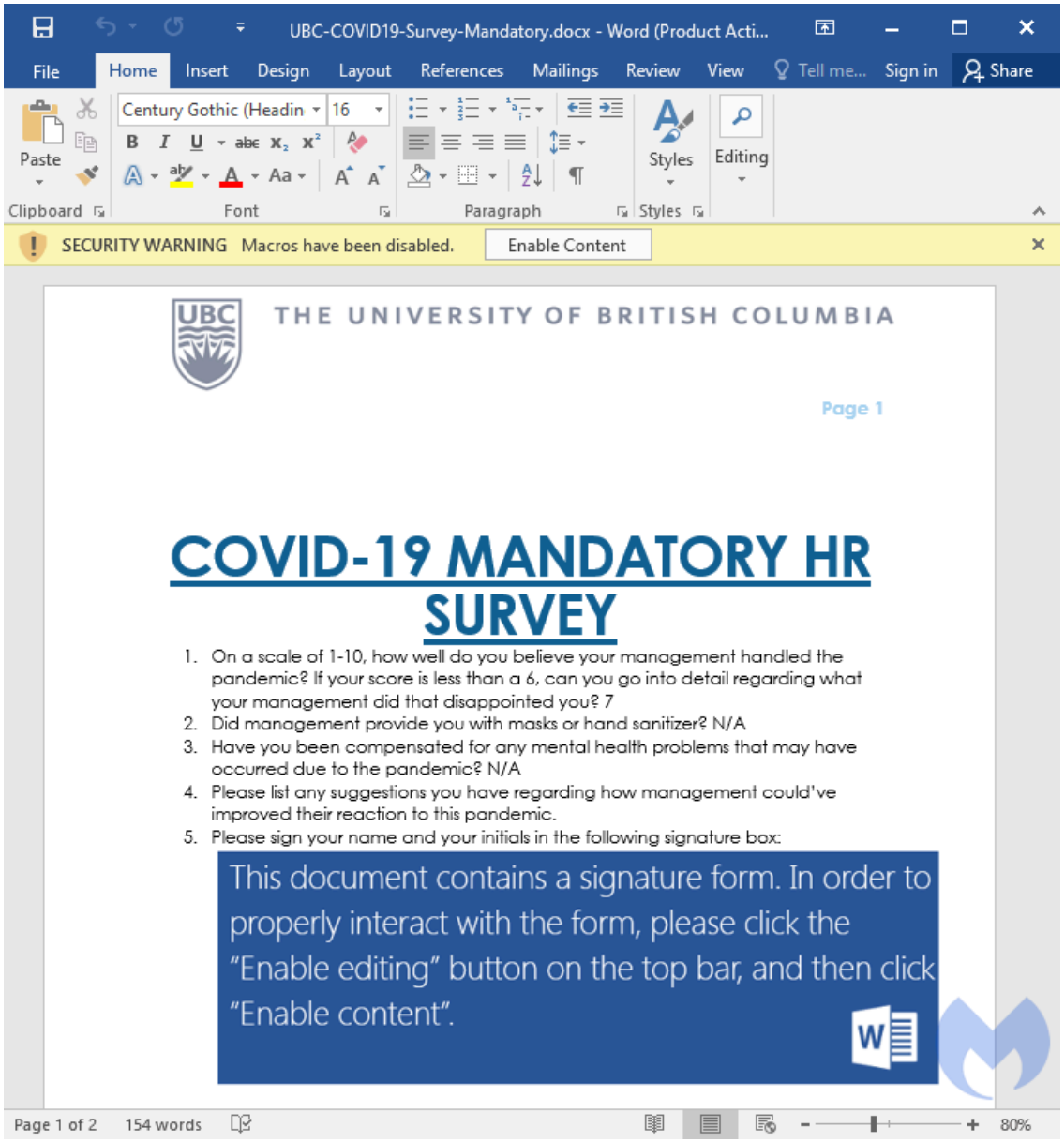
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

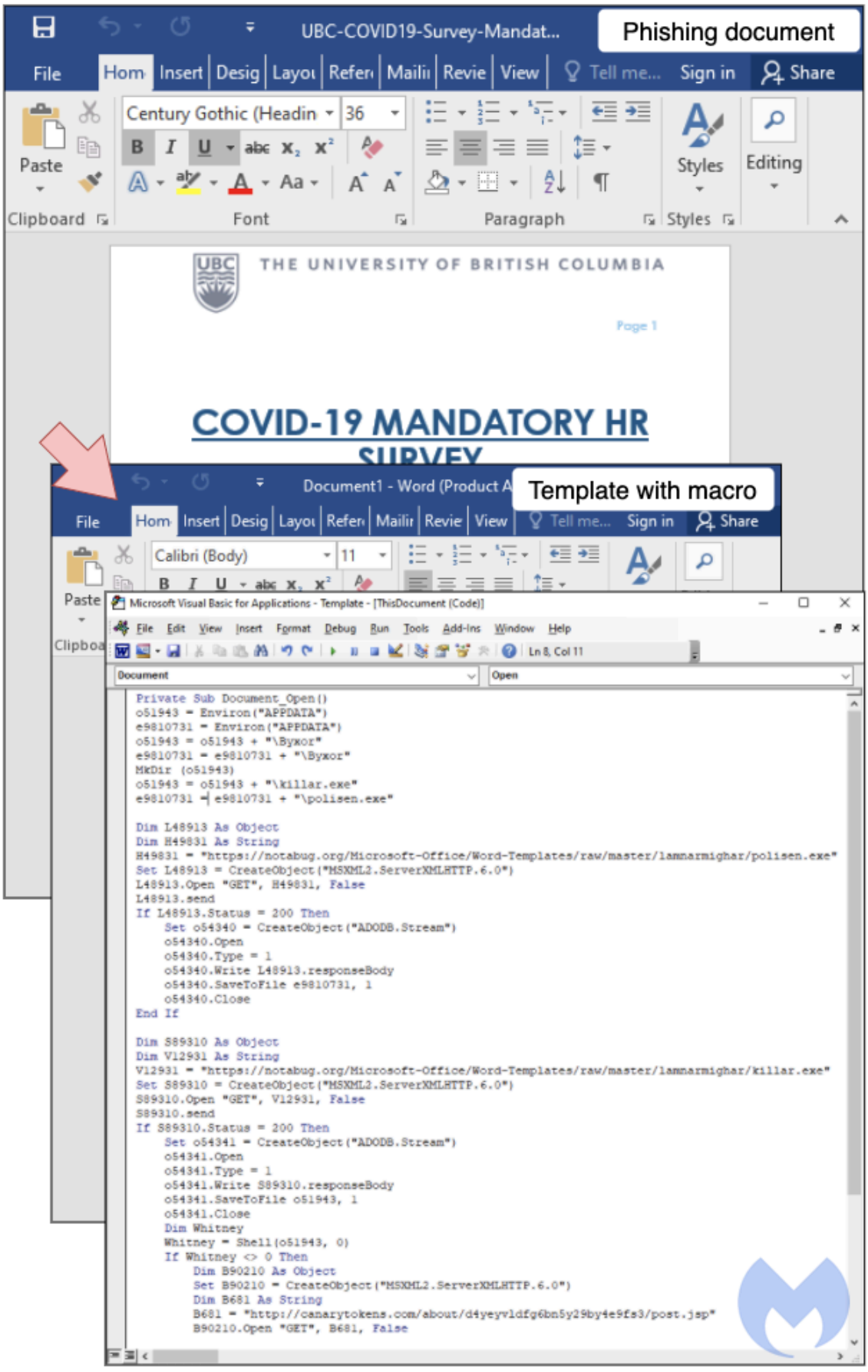
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

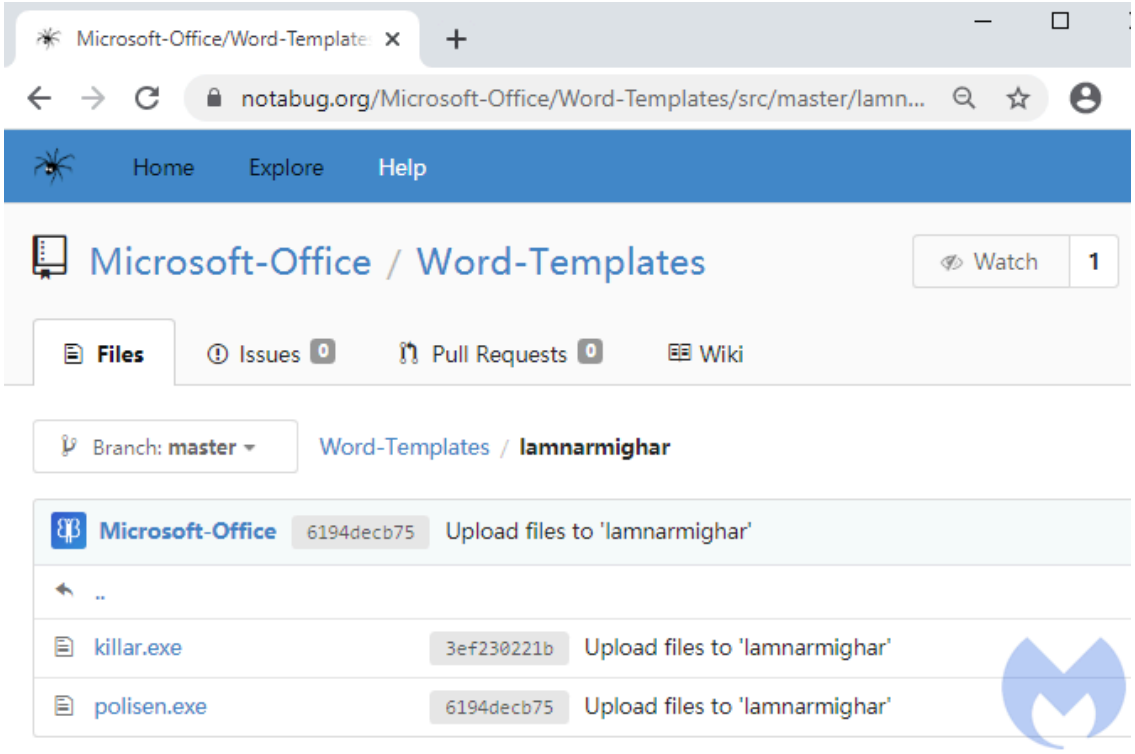
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



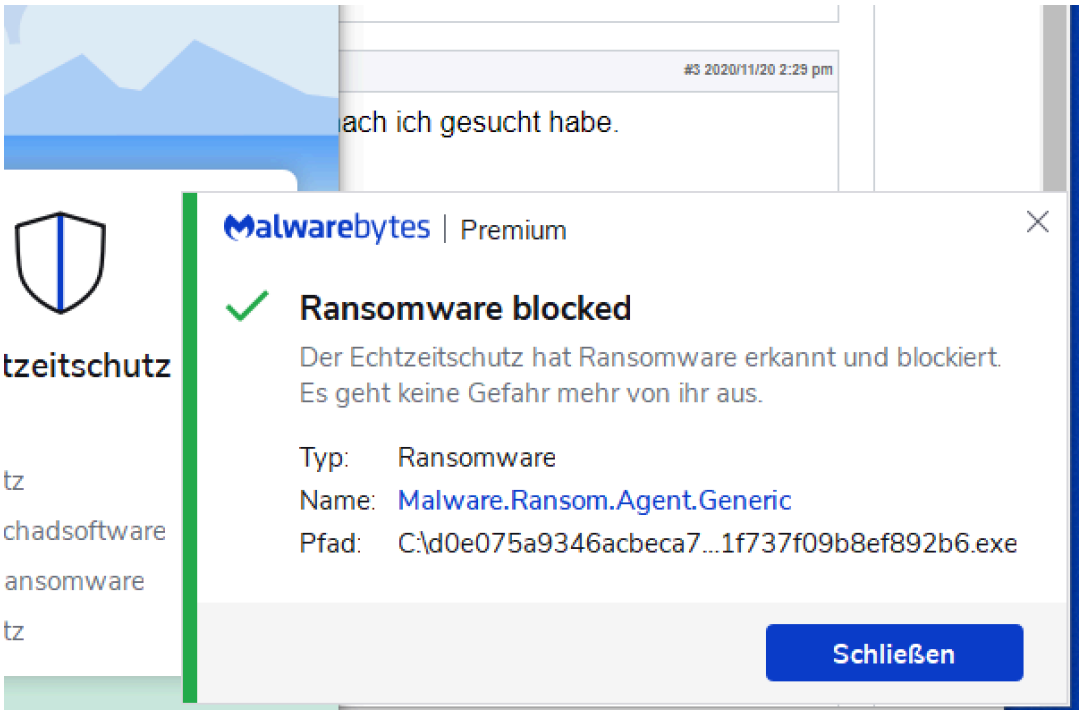
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

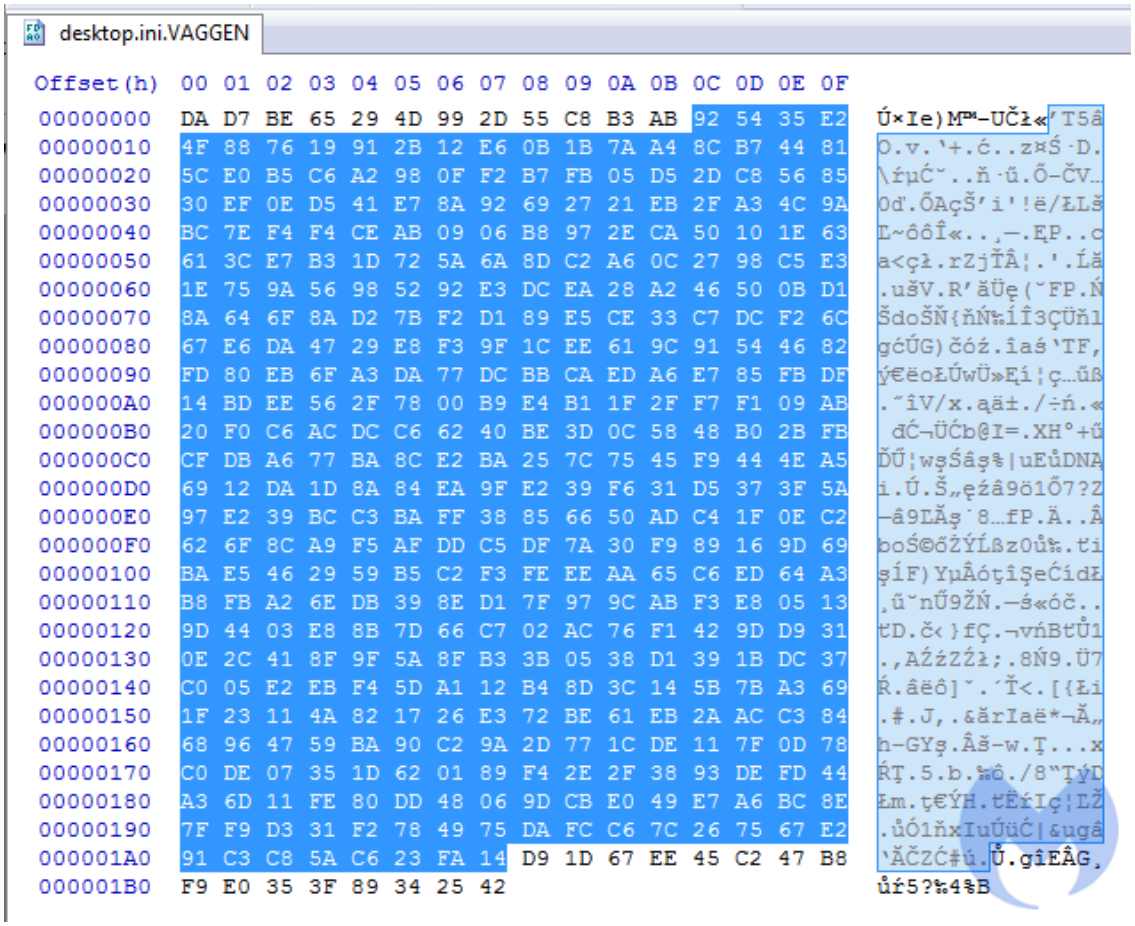
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

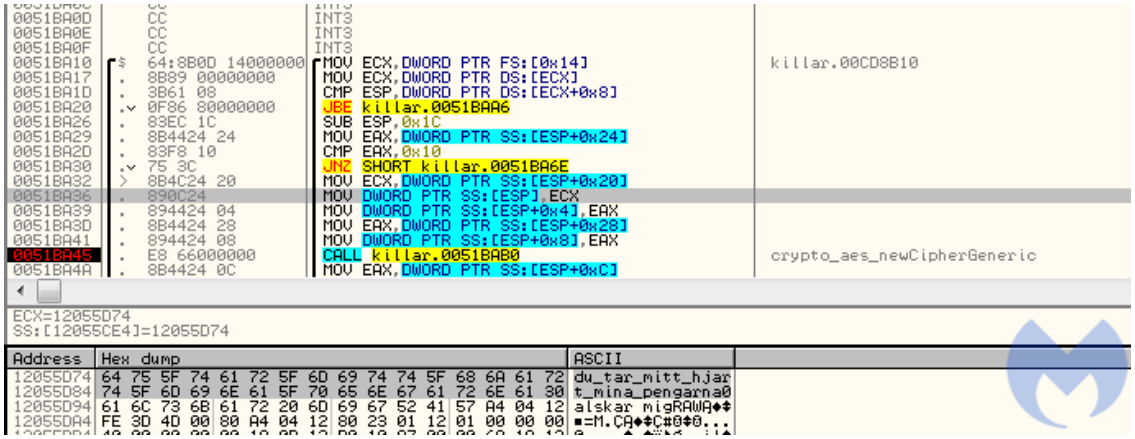


The content of the output file (with .VAGGEN extension) contains:

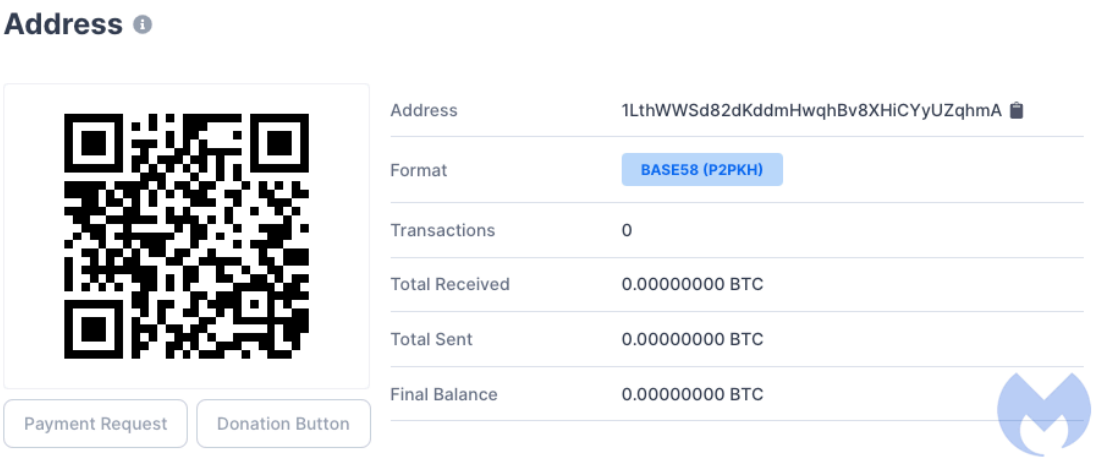
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

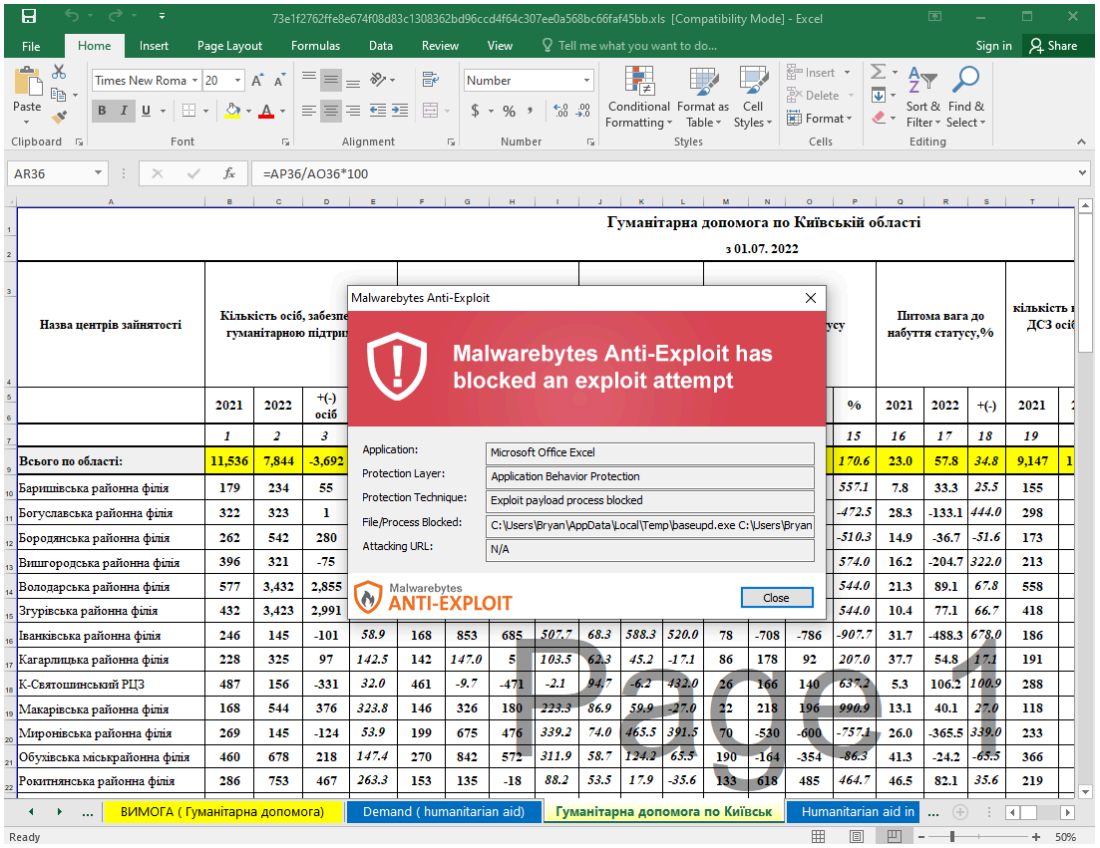
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

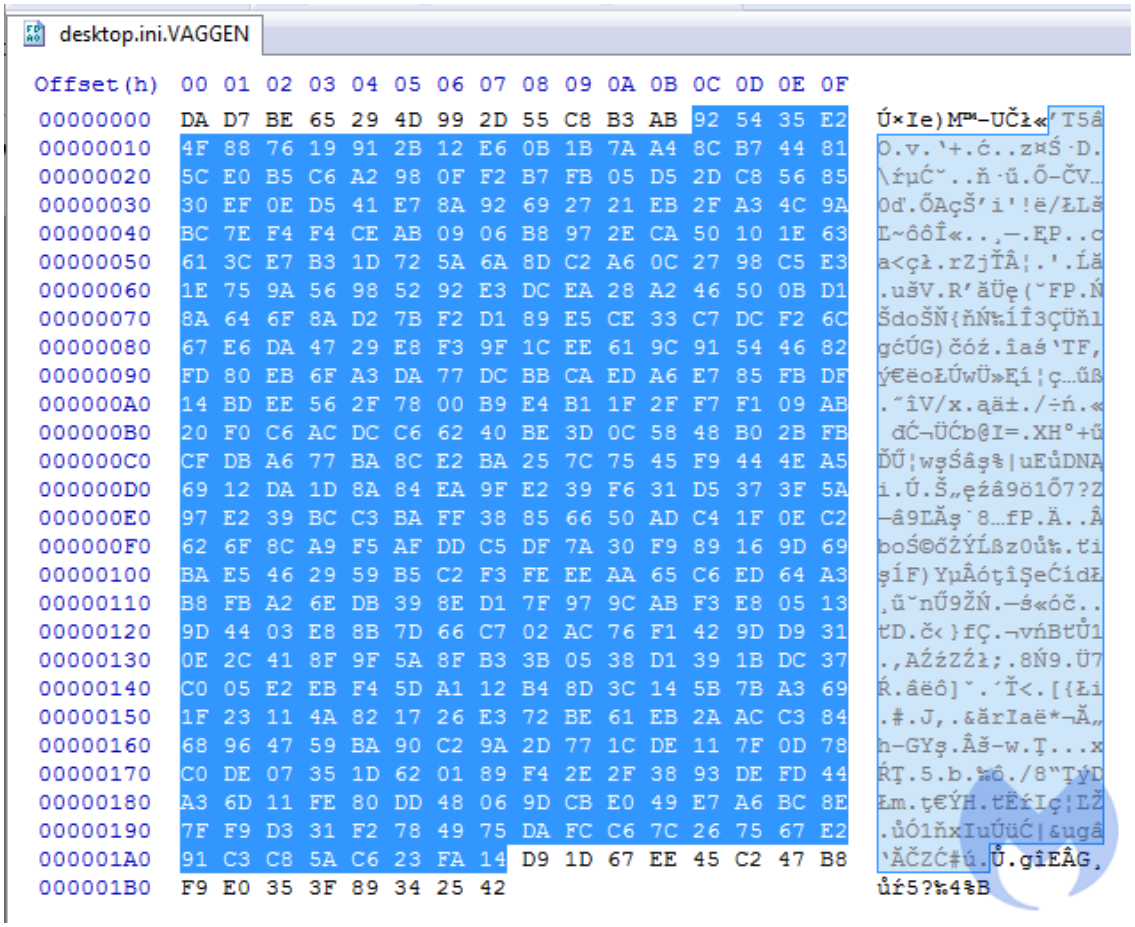
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

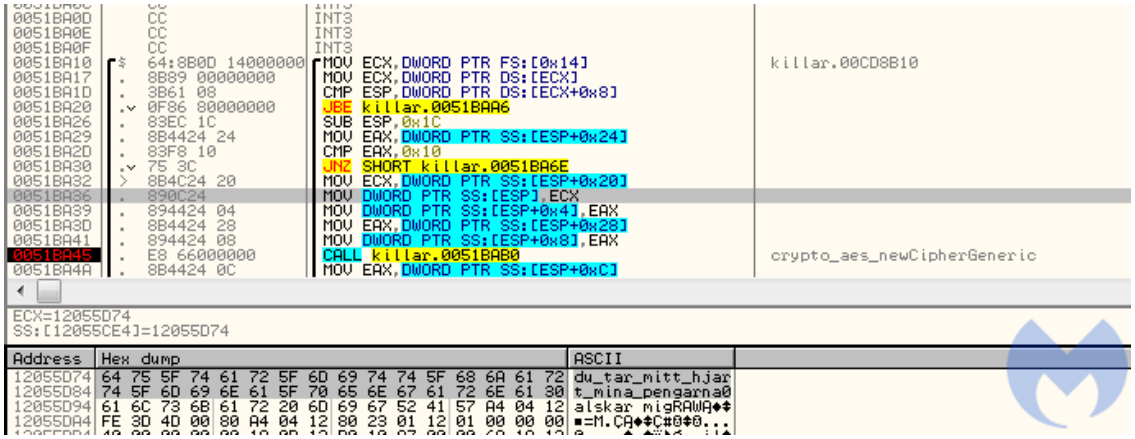
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

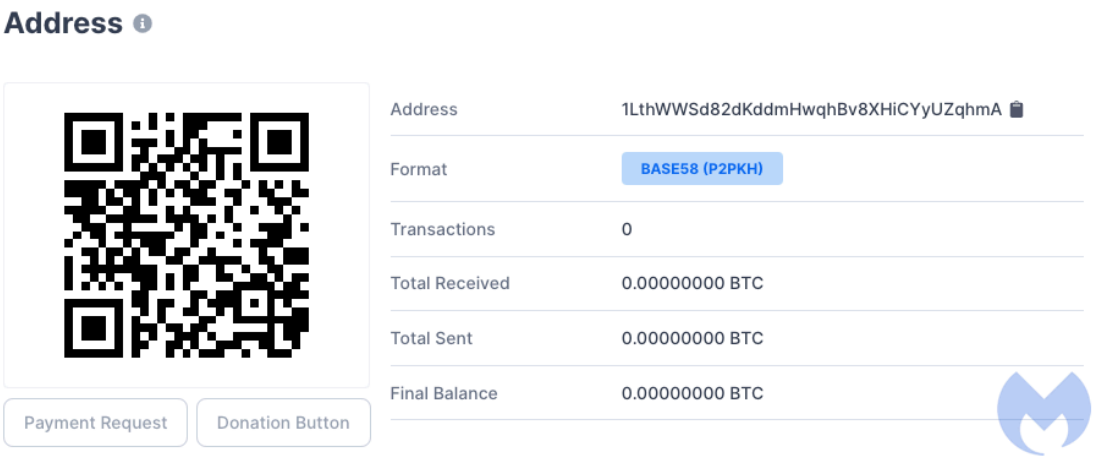
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

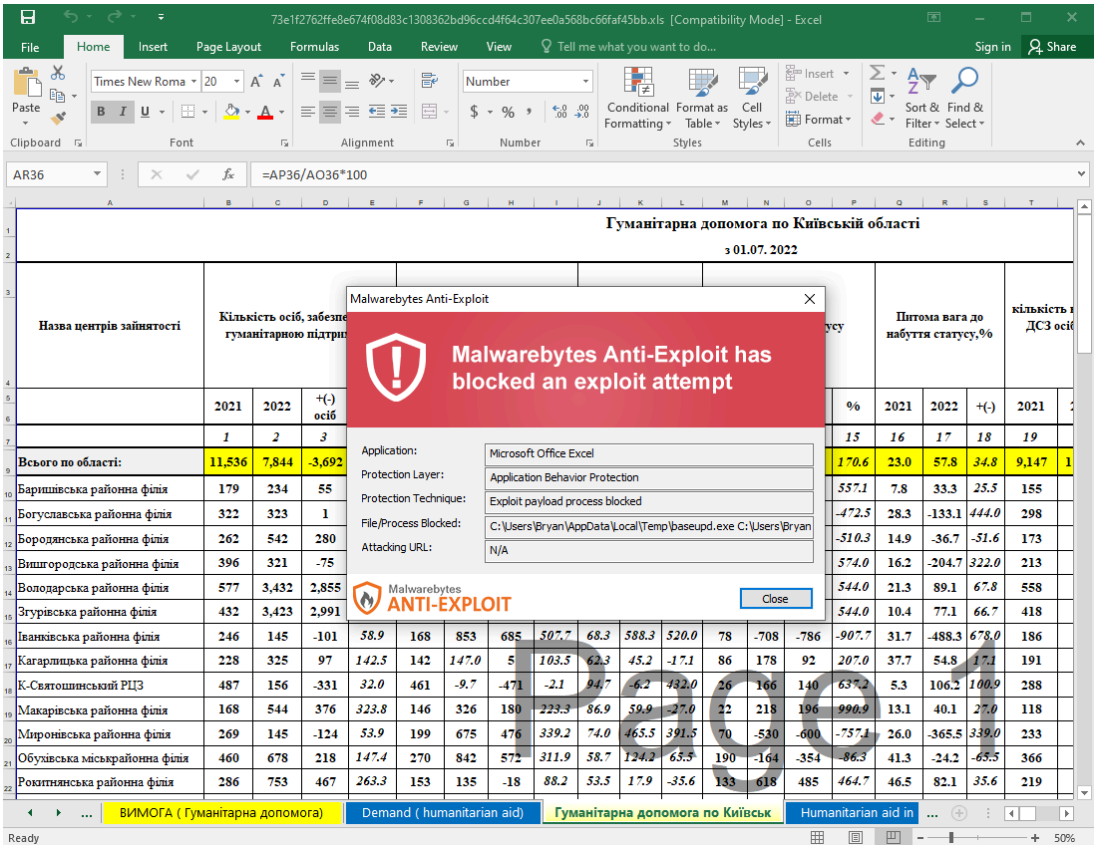
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

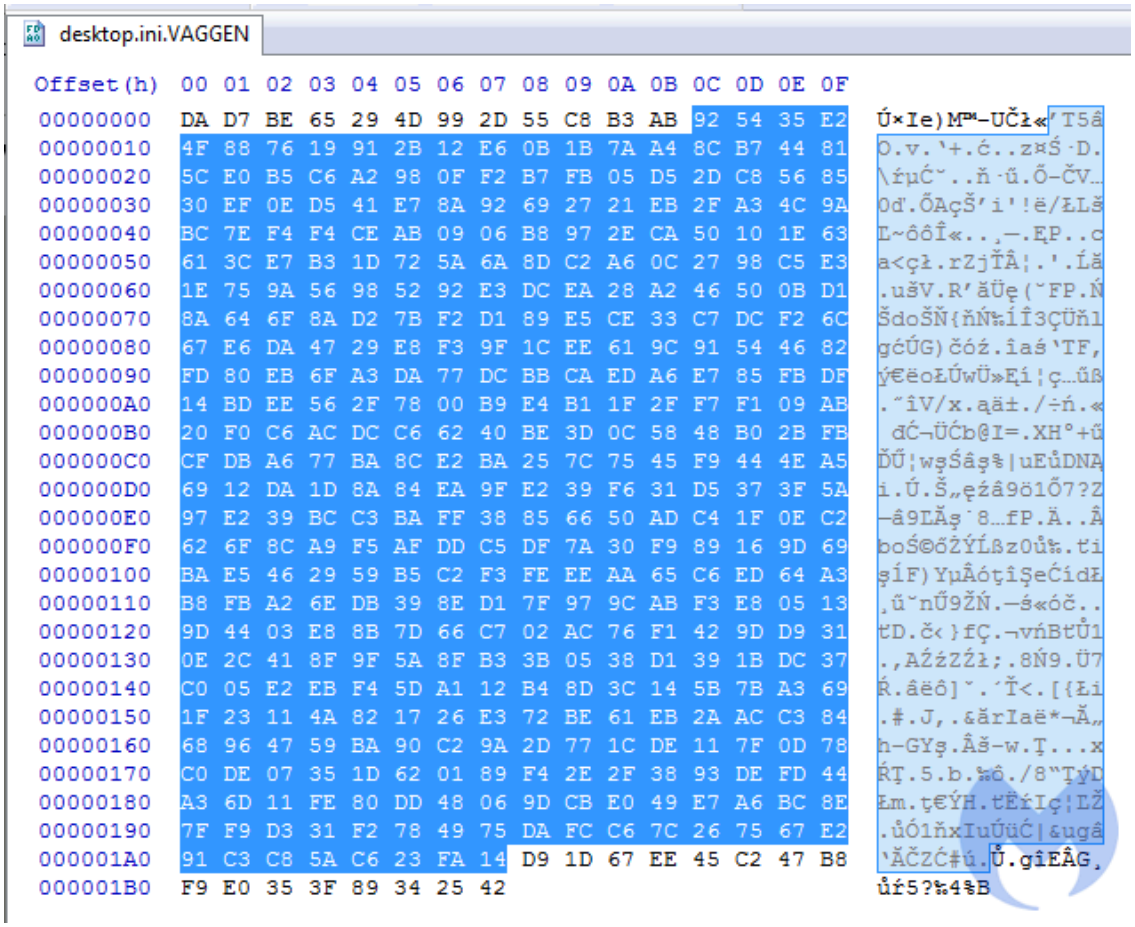
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



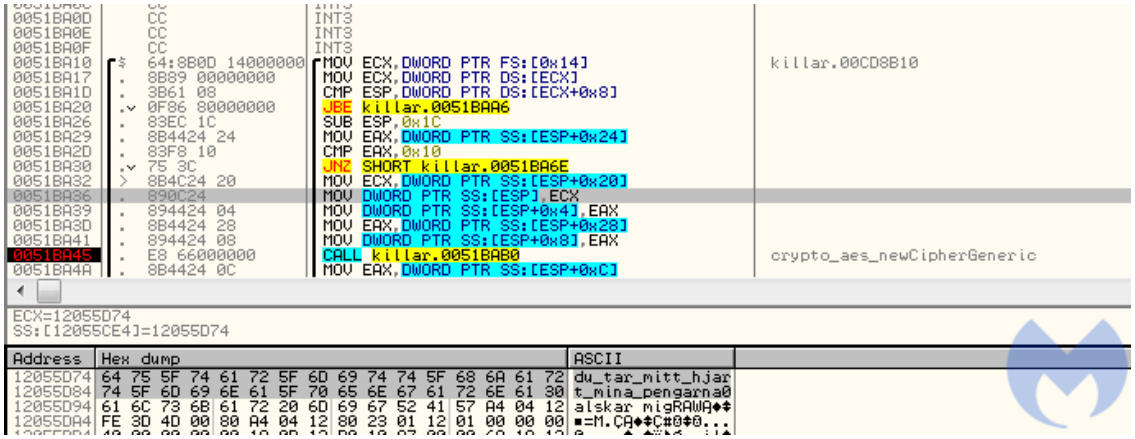
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

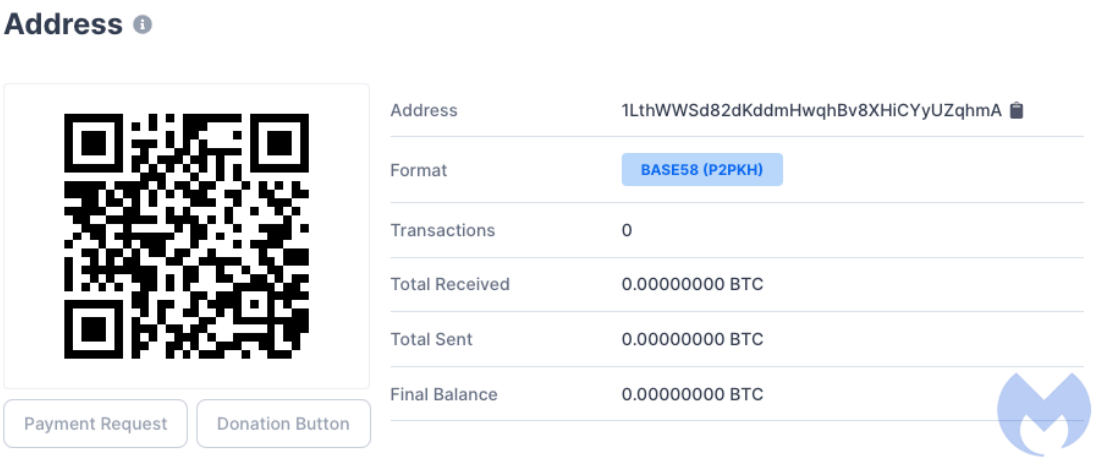
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

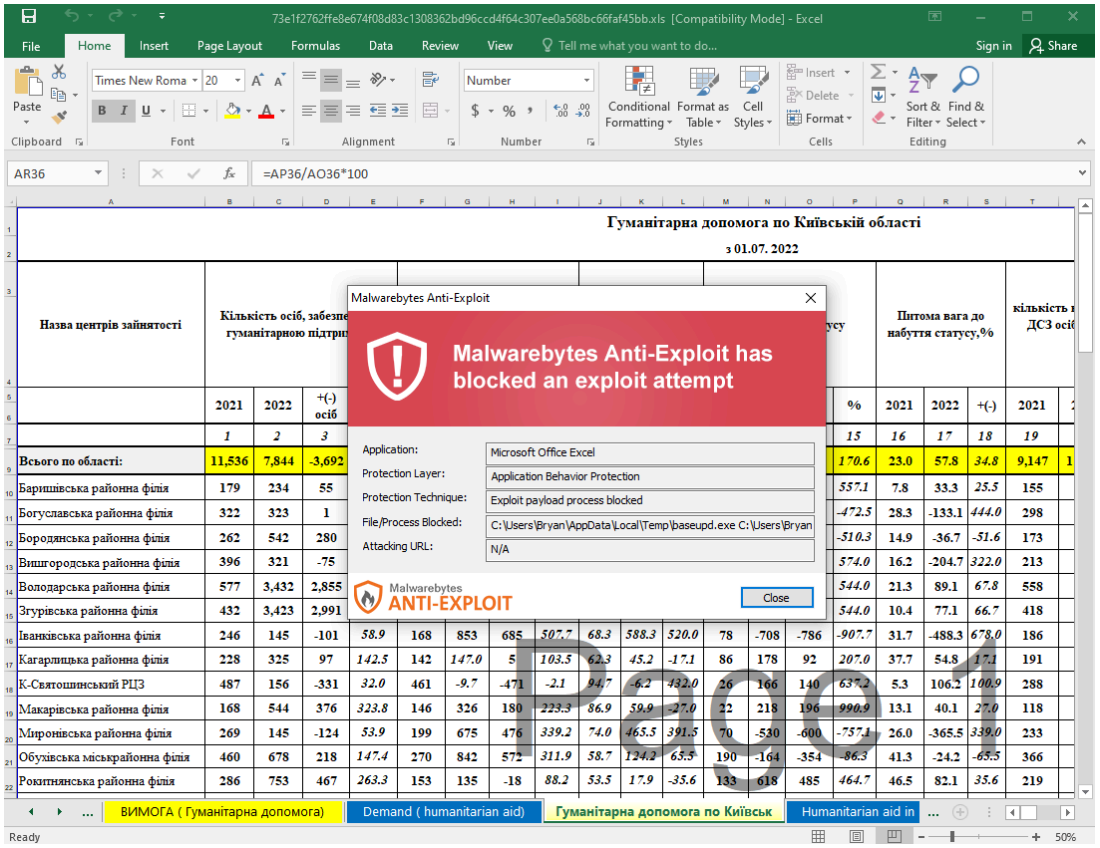
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
46  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
47    runtime_morestack_noctxt();
48  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
49  ((void (*)(void))loc_45CDA0)();
50  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
51  v38 = v4;
52  v30 = v5;
53  v31 = v6;
54  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
55  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
56  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
57  v32 = v10;
58  v11 = *(void (__cdecl **)(int))(v10 + 16);
59  v39 = v12;
60  v11(v12);
61  v29 = v13;
62  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
63  v40 = v15;
64  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
65  v17 = v27;
66  if ( v27 )
67  {
68    if ( v27 )
69      v17 = *(_DWORD *)(v27 + 4);
70    v43 = v17;
71    v44 = v28;
72    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
73  }
```

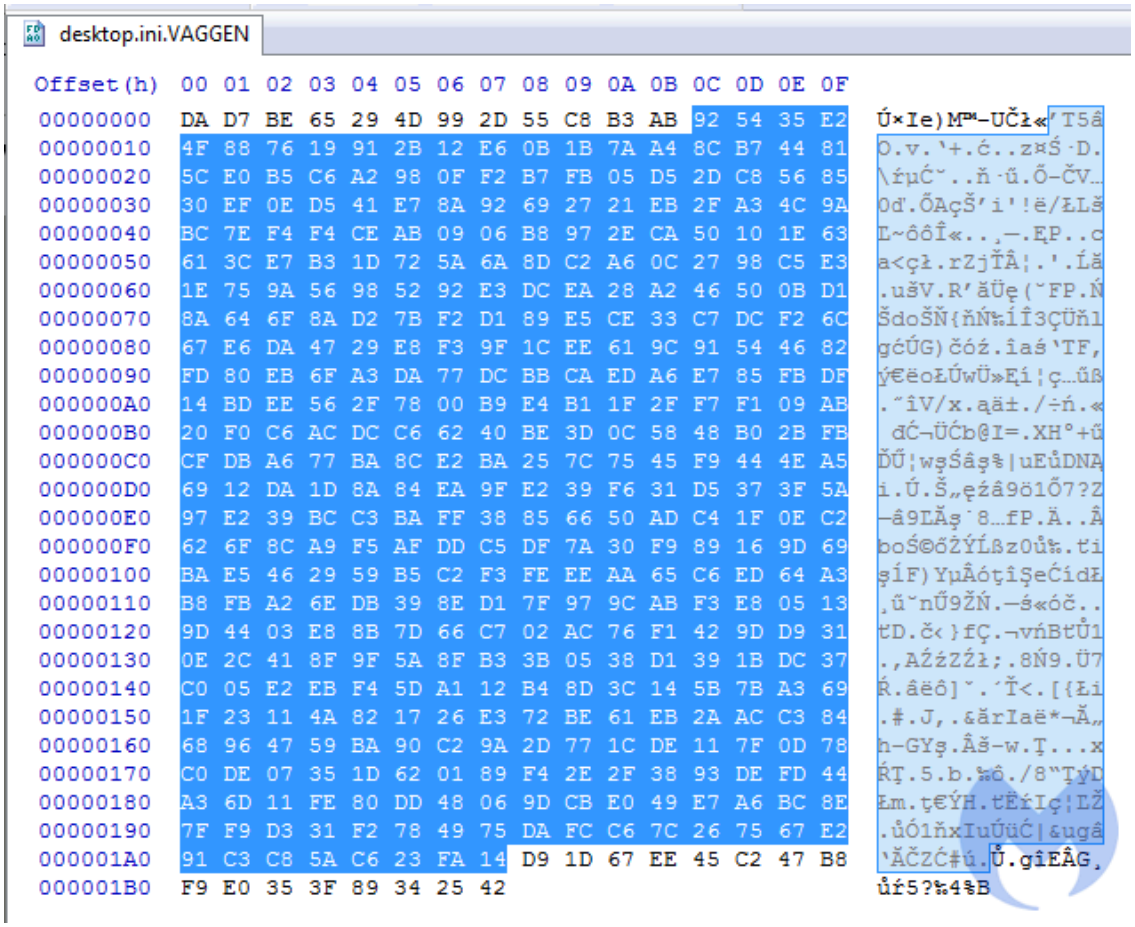


The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

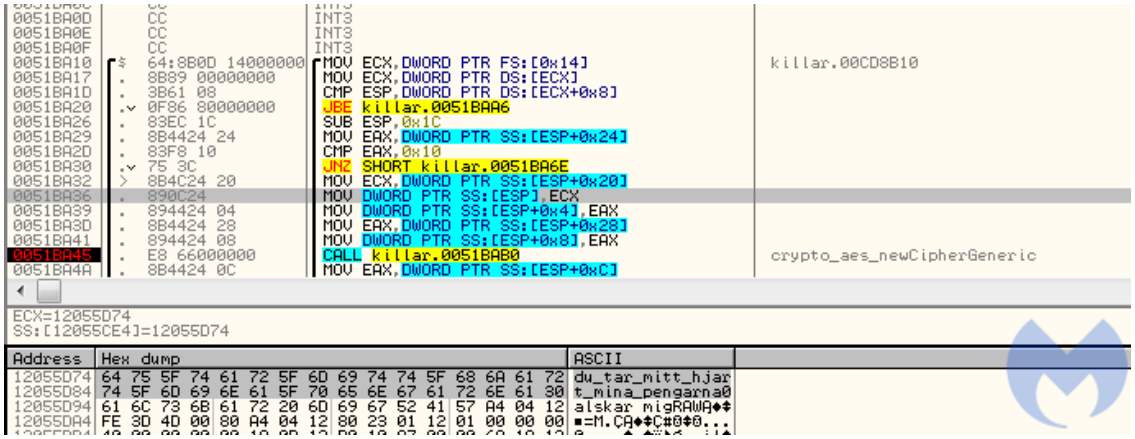


The content of the output file (with .VAGGEN extension) contains:

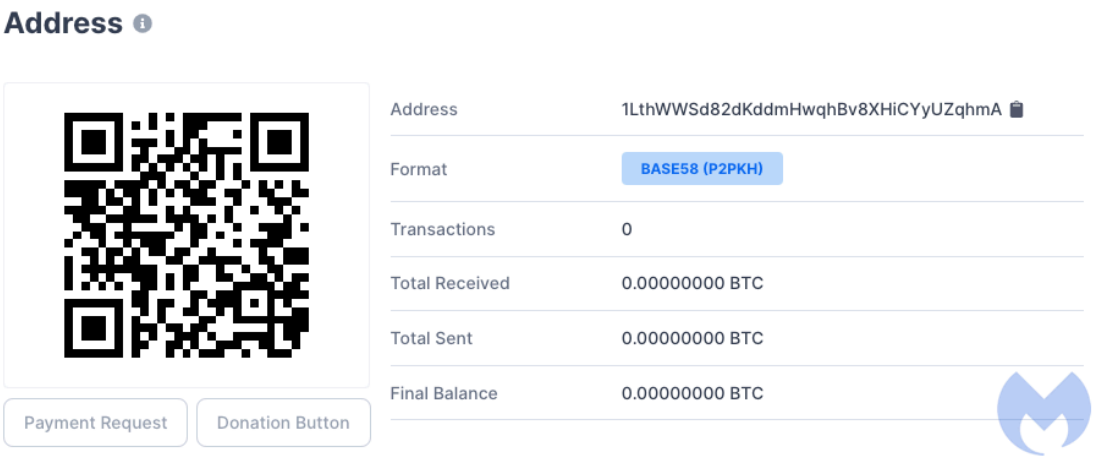
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

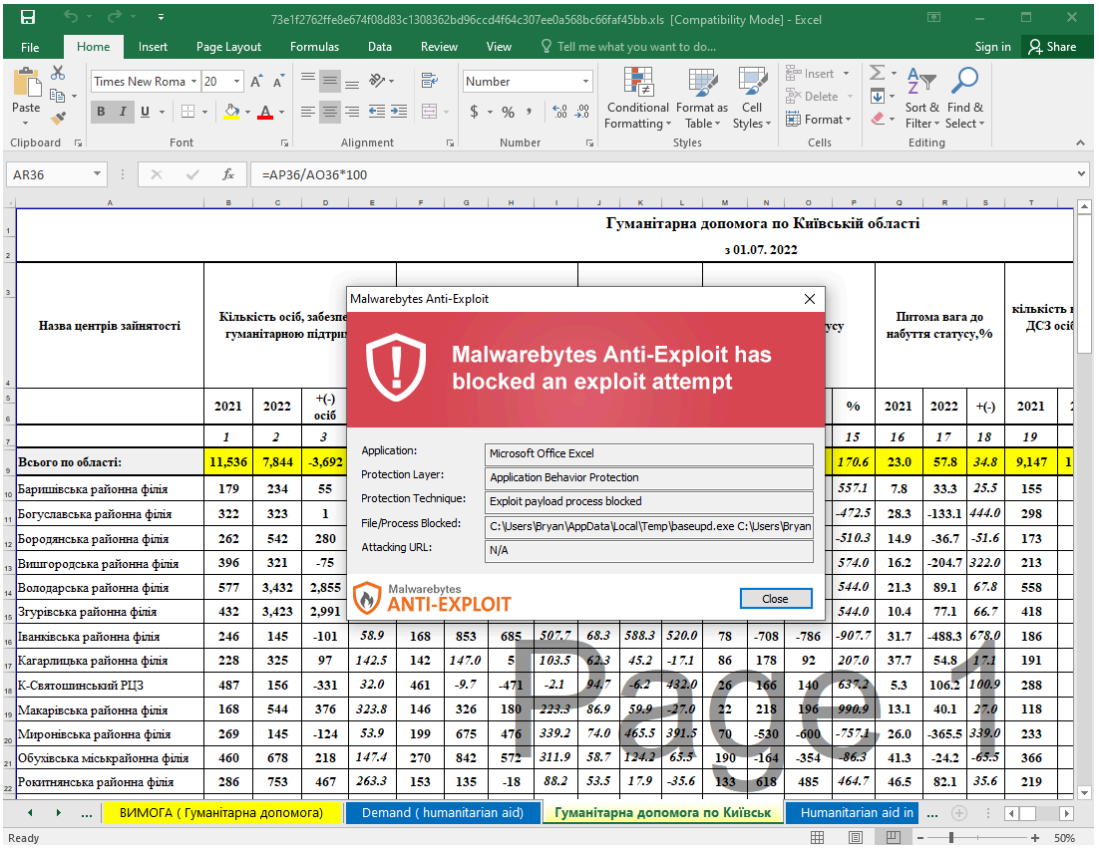
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

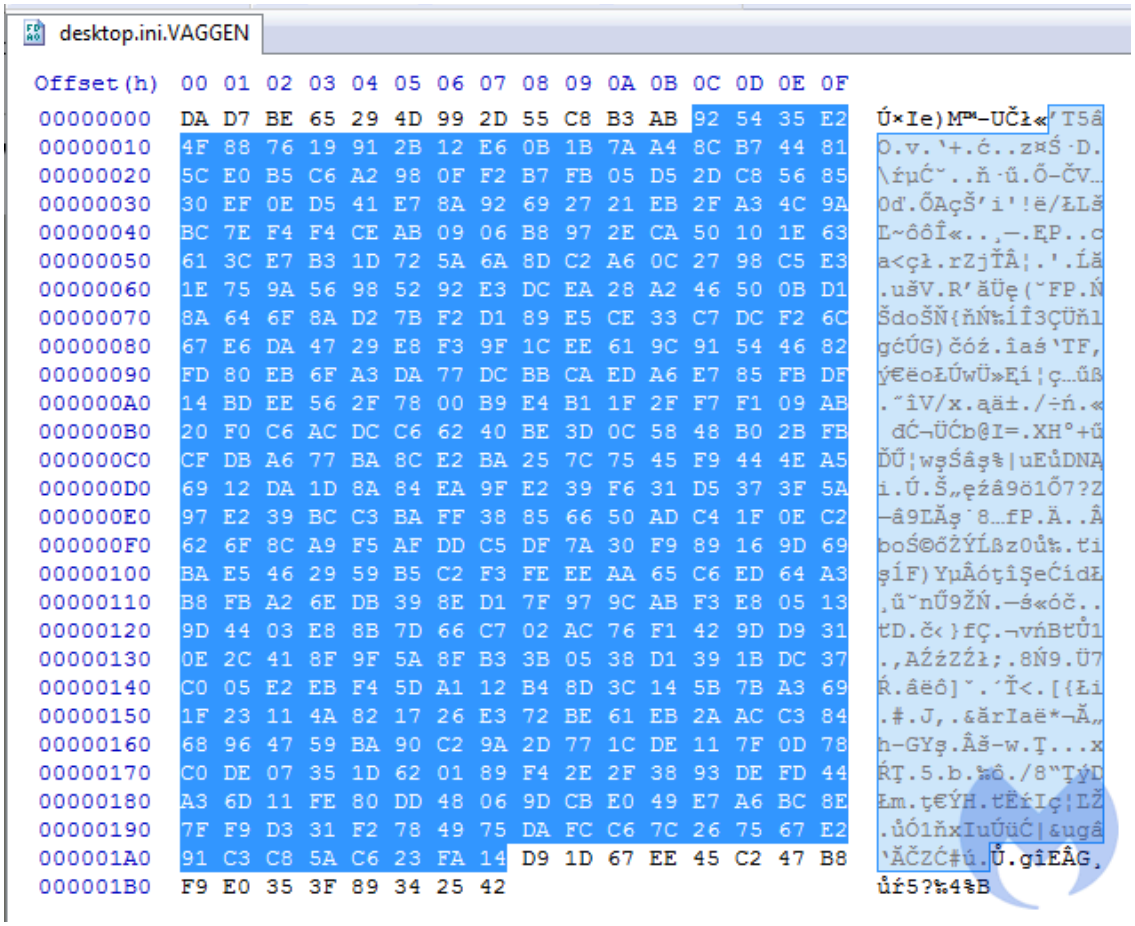
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

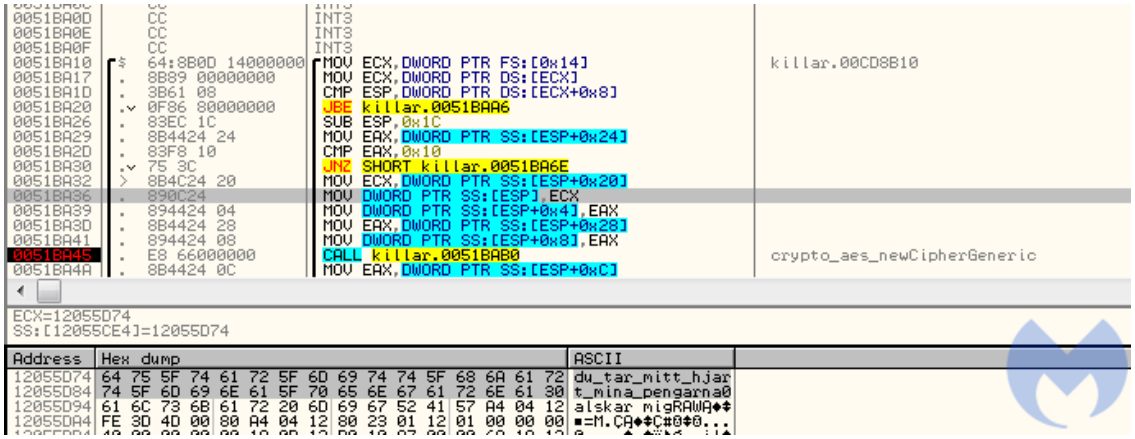


The content of the output file (with .VAGGEN extension) contains:

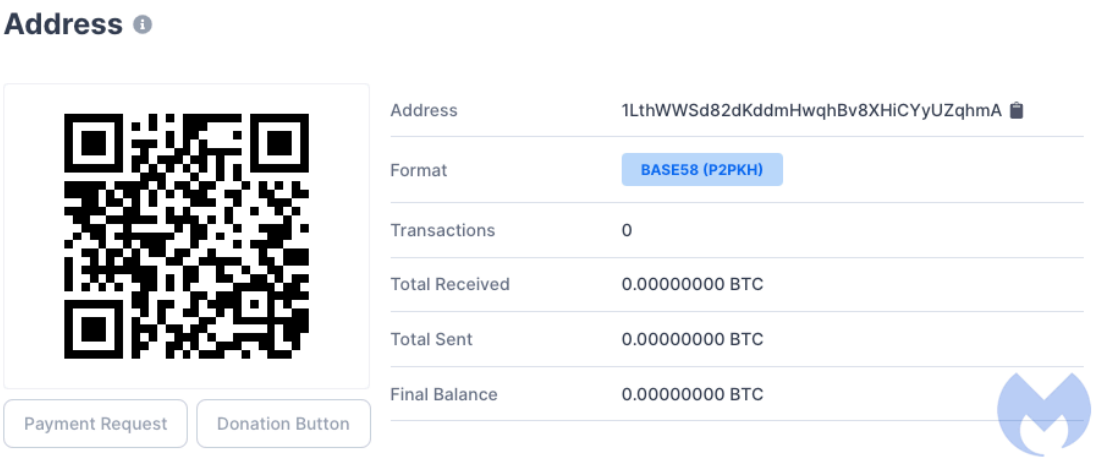
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

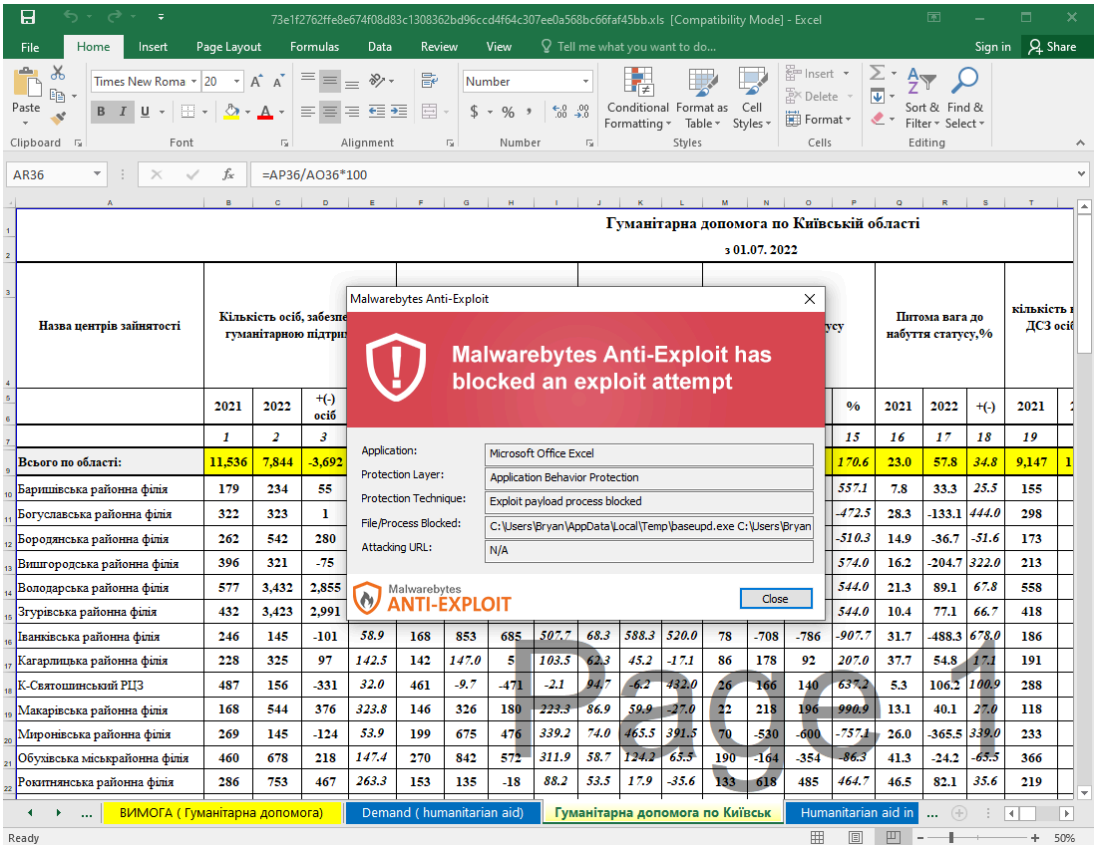
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

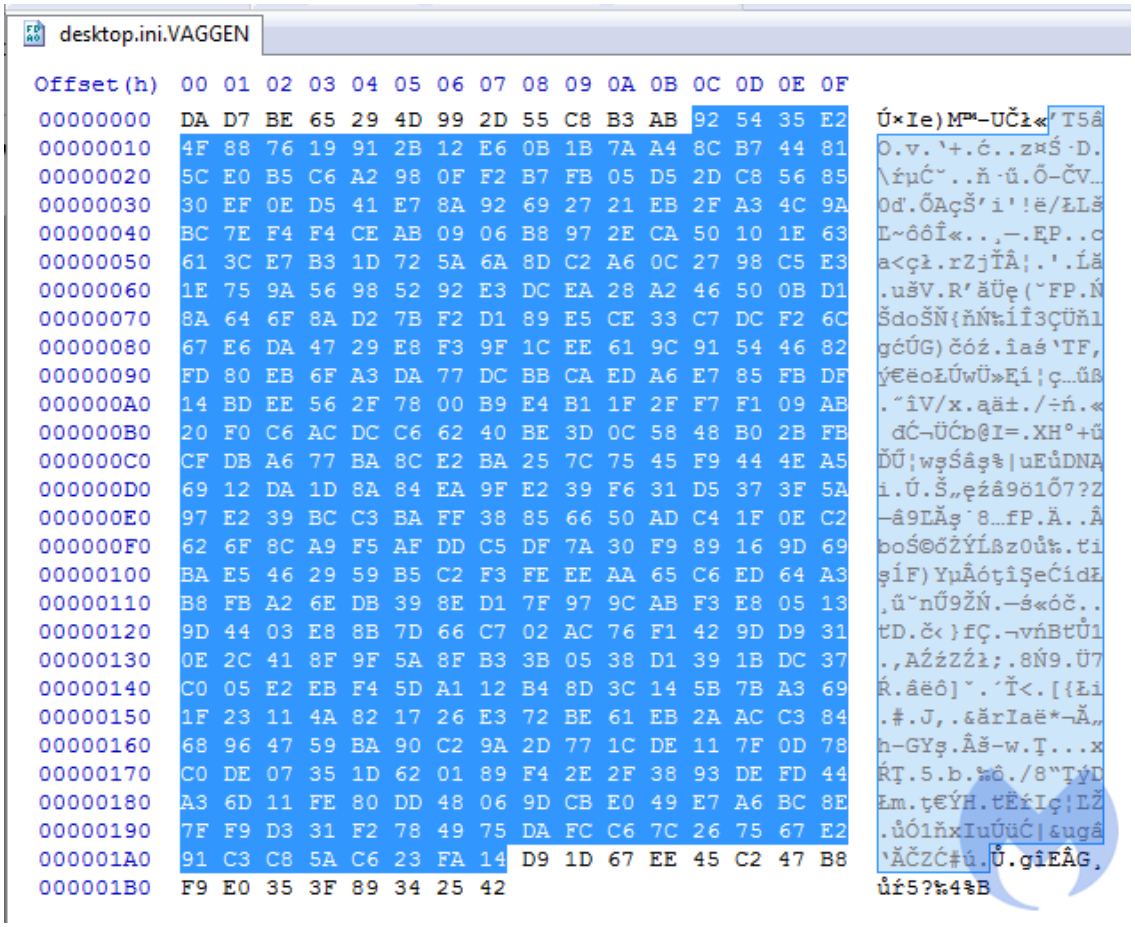
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

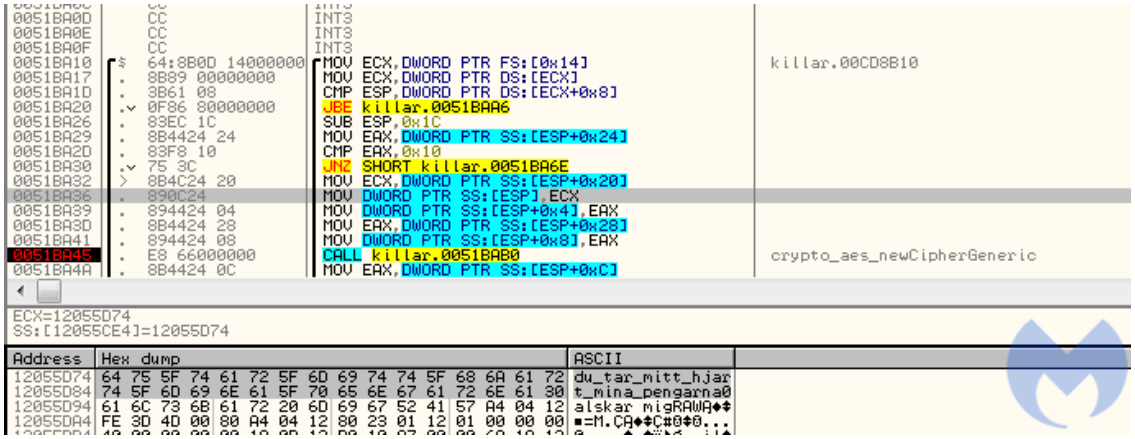


The content of the output file (with .VAGGEN extension) contains:

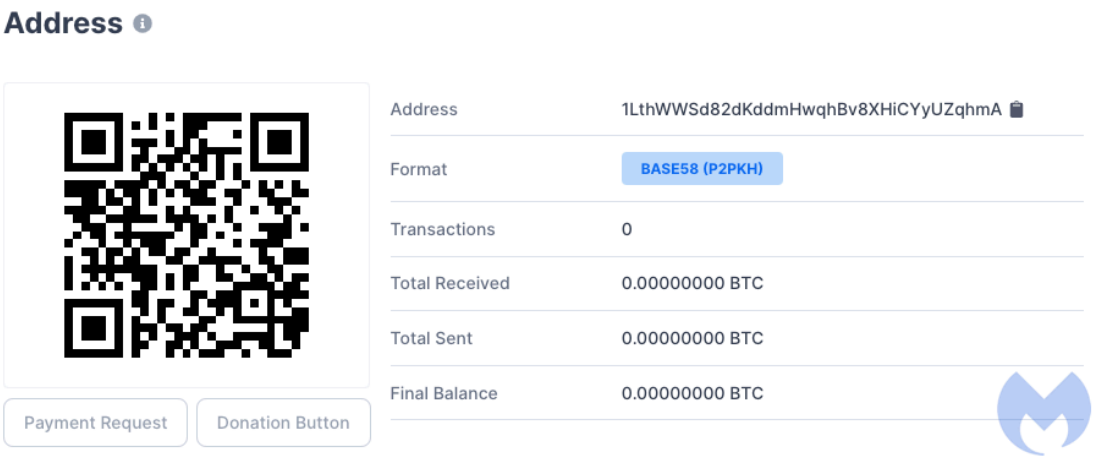
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

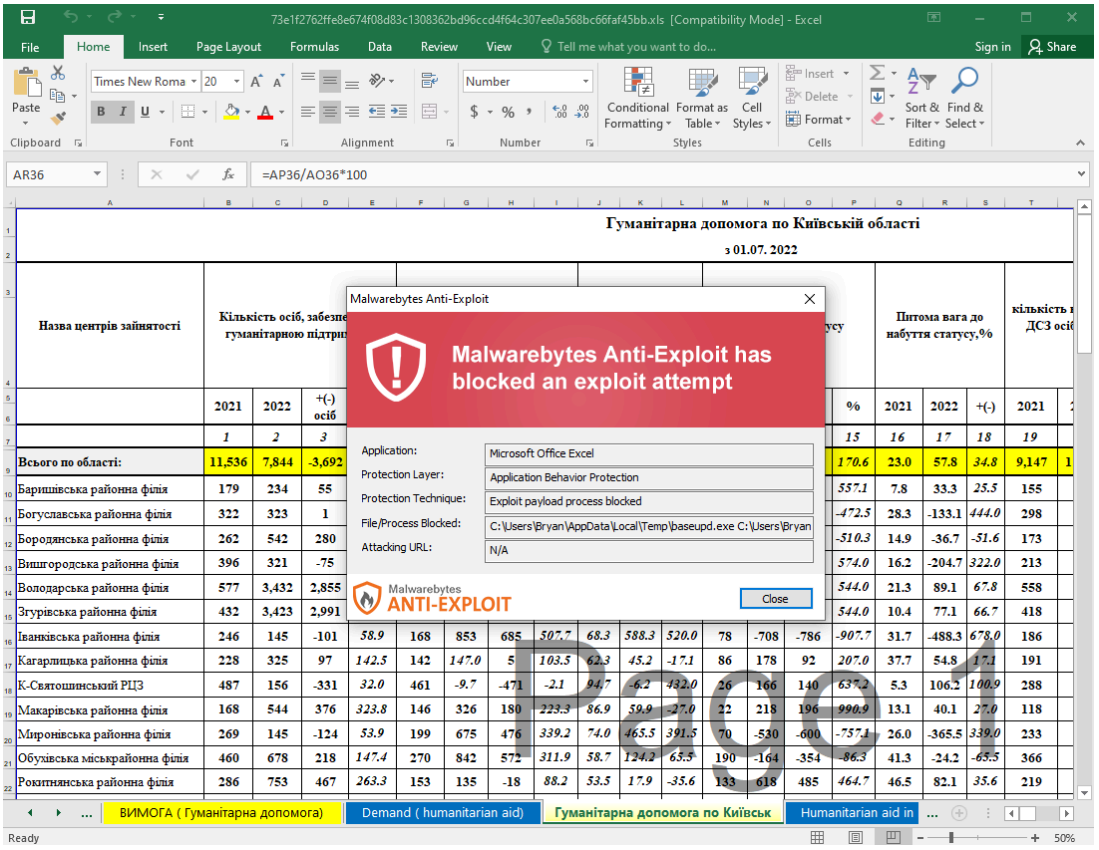
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

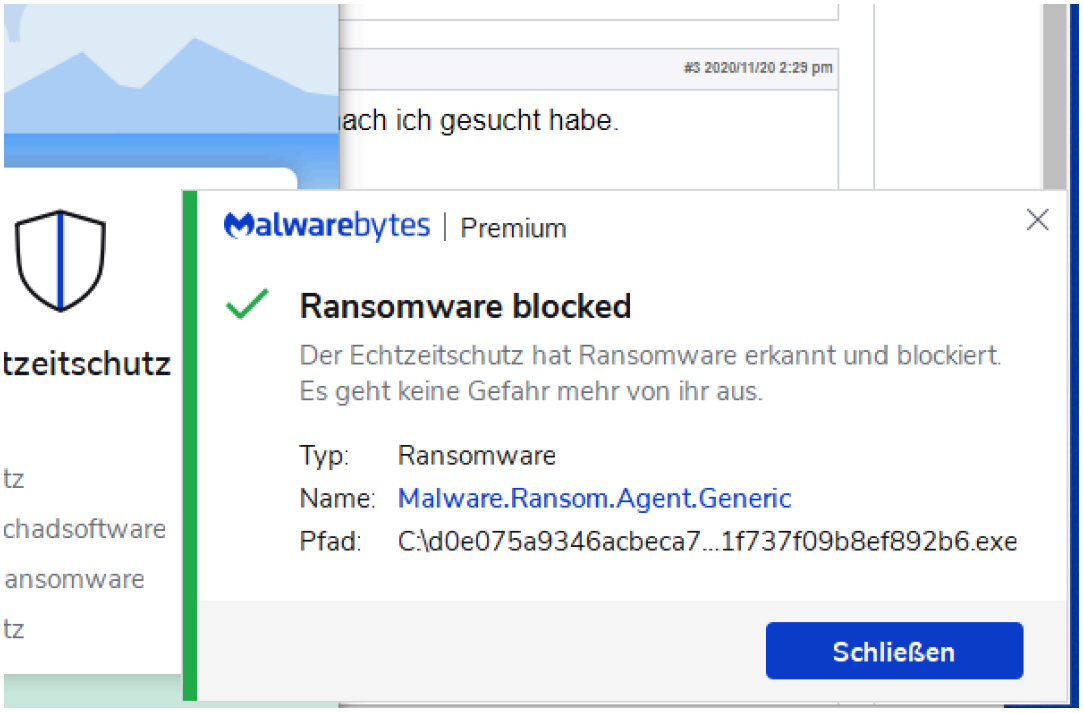
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```



Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

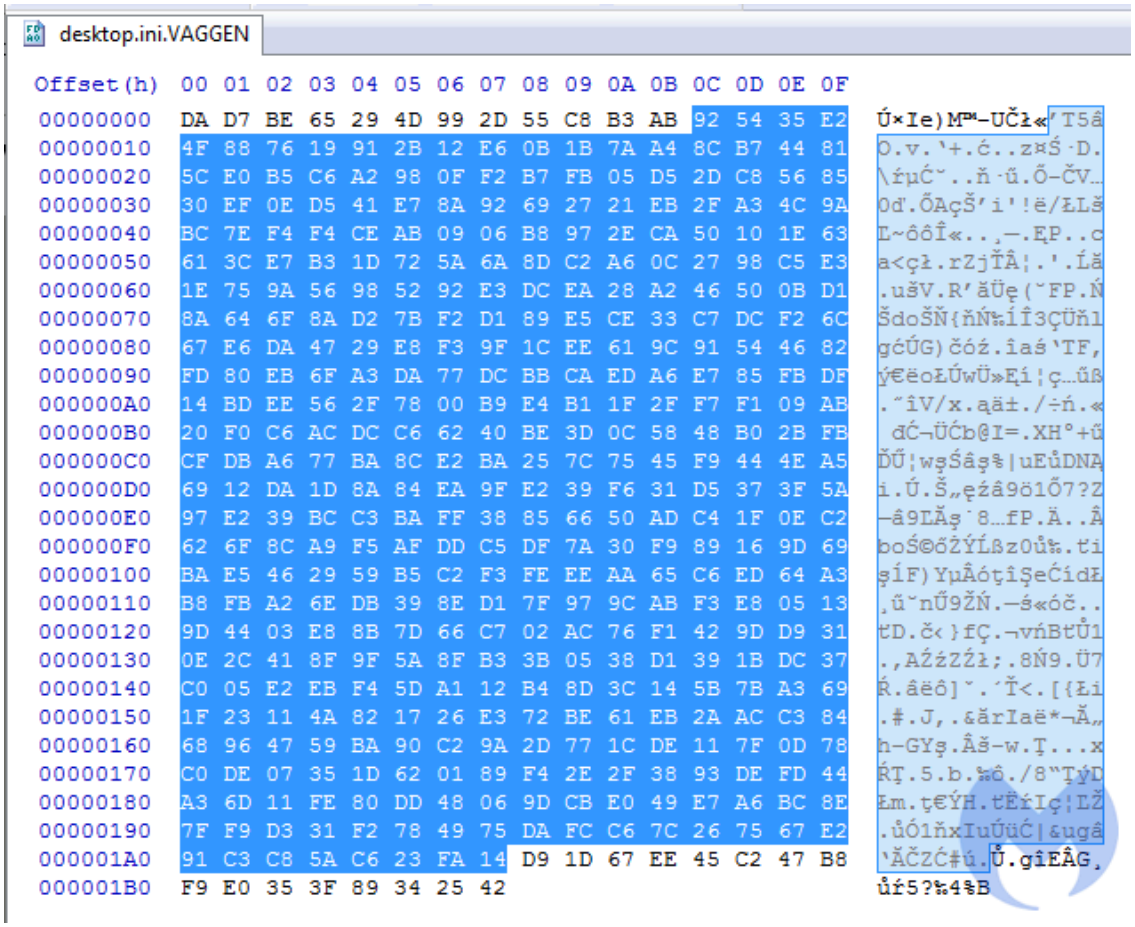


The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

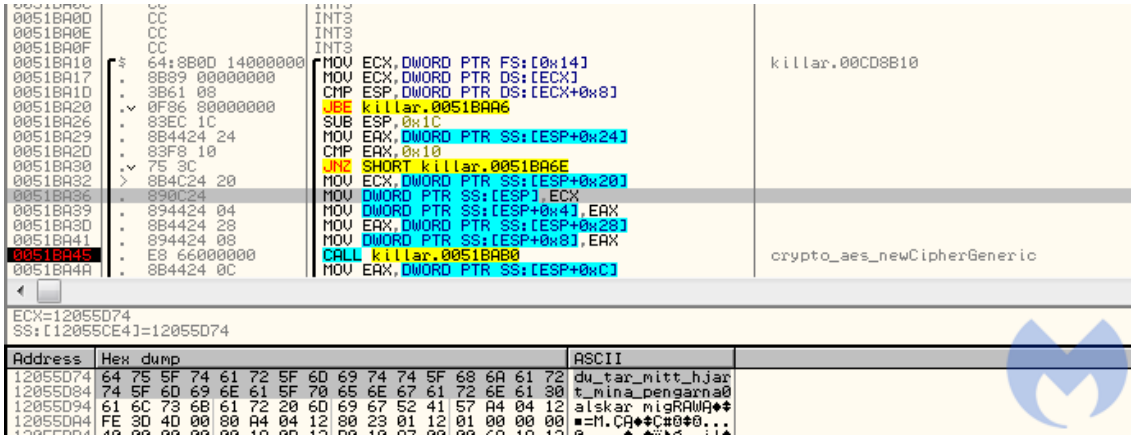


The content of the output file (with .VAGGEN extension) contains:

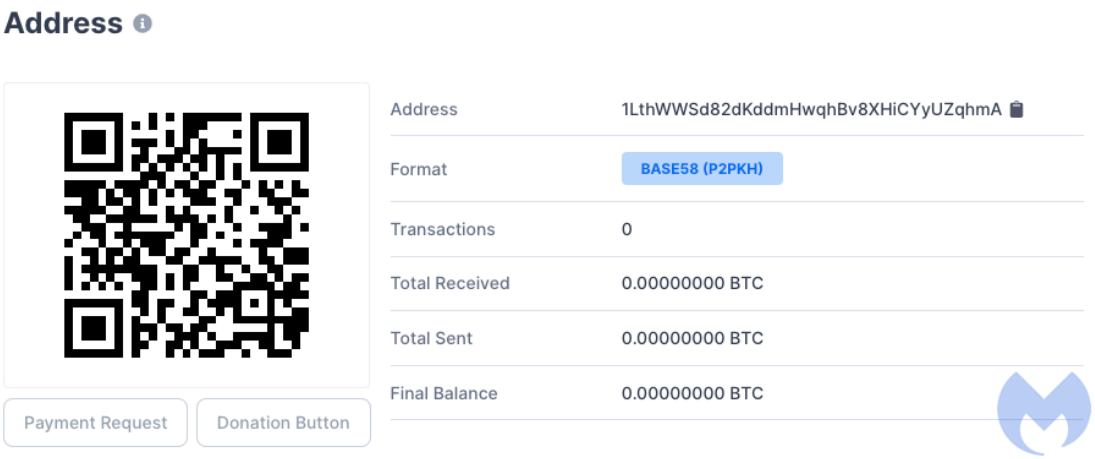
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

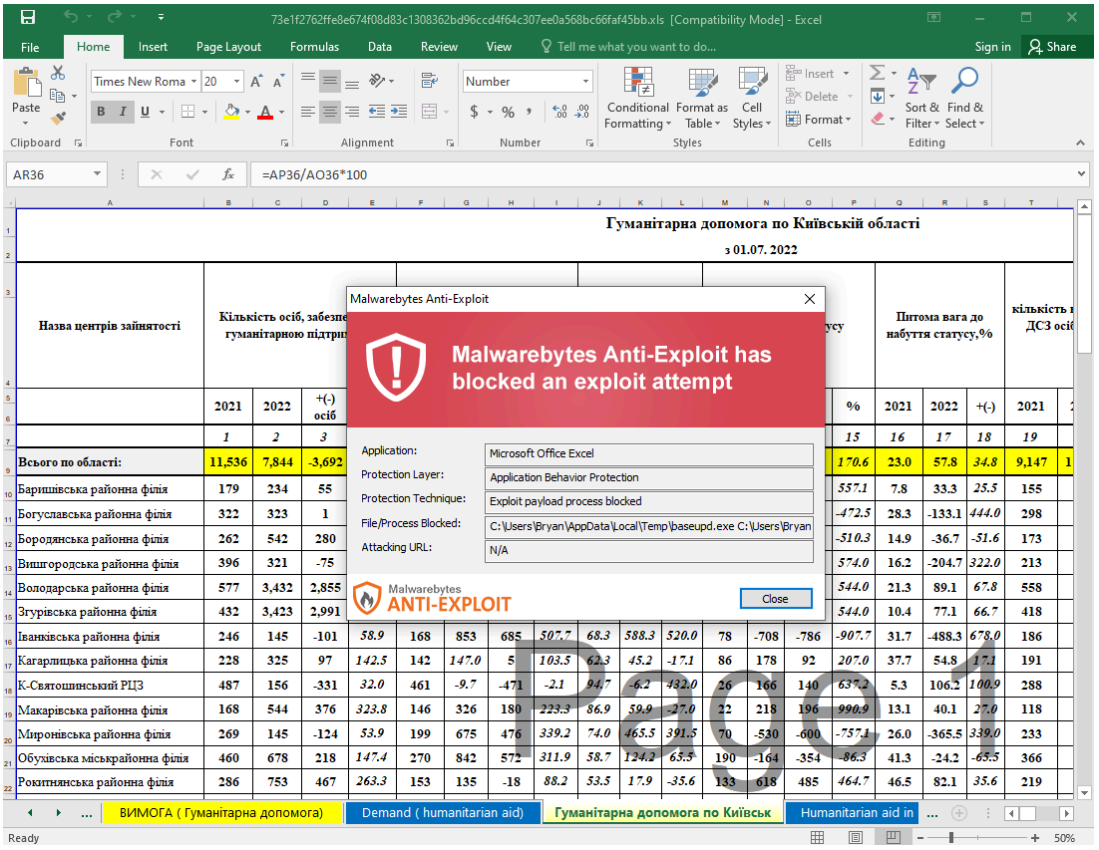
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

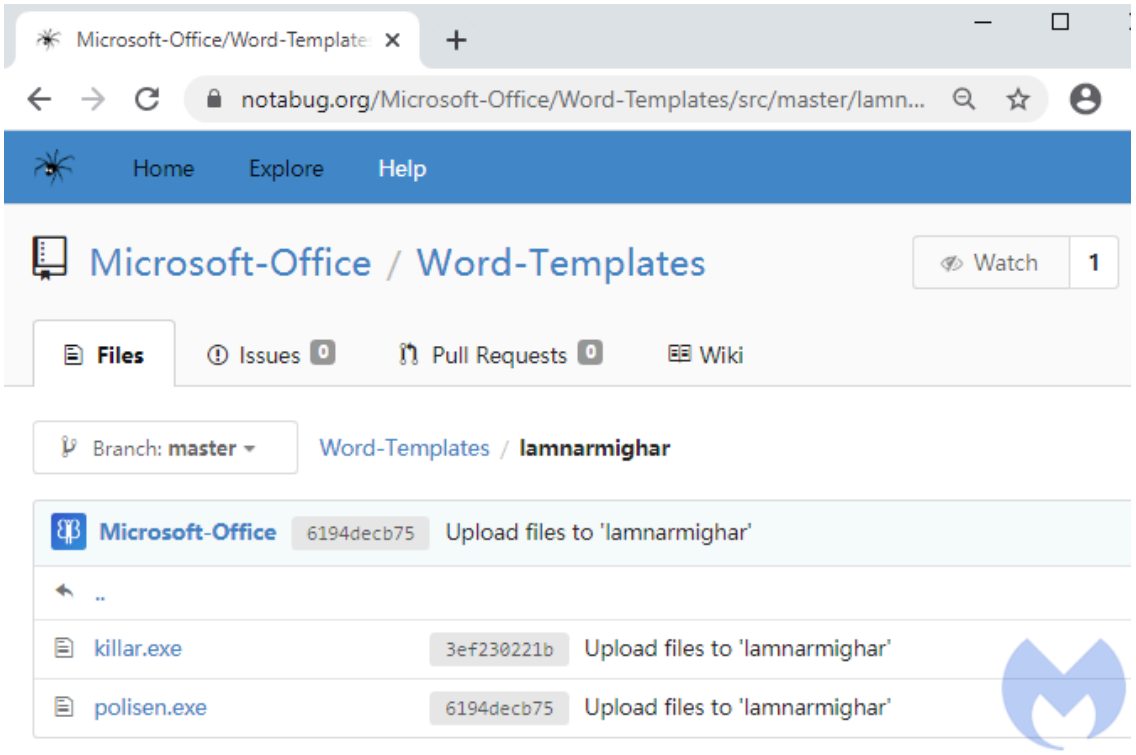
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



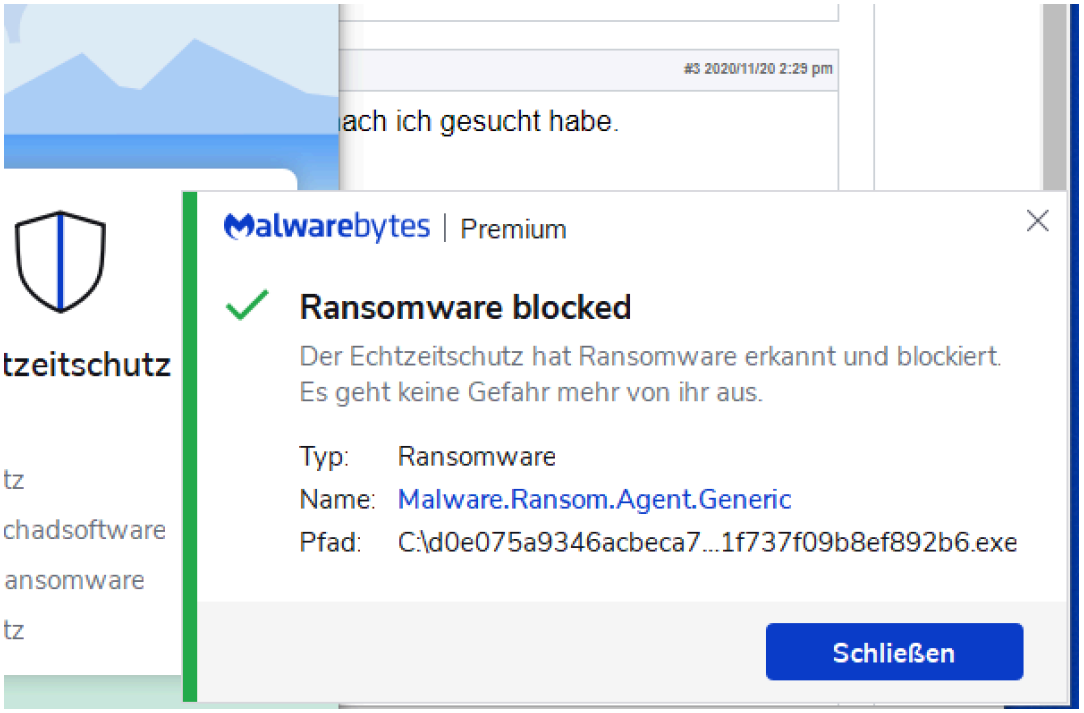
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

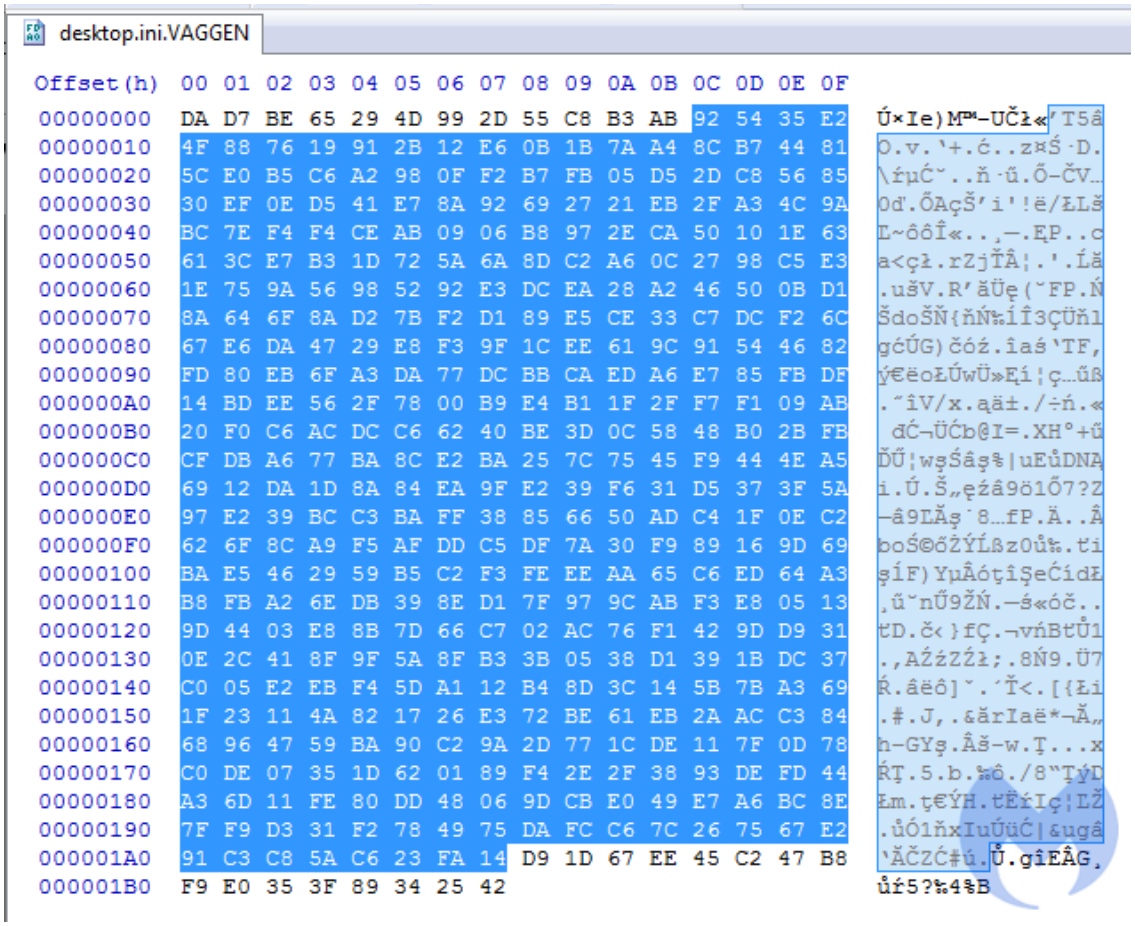
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

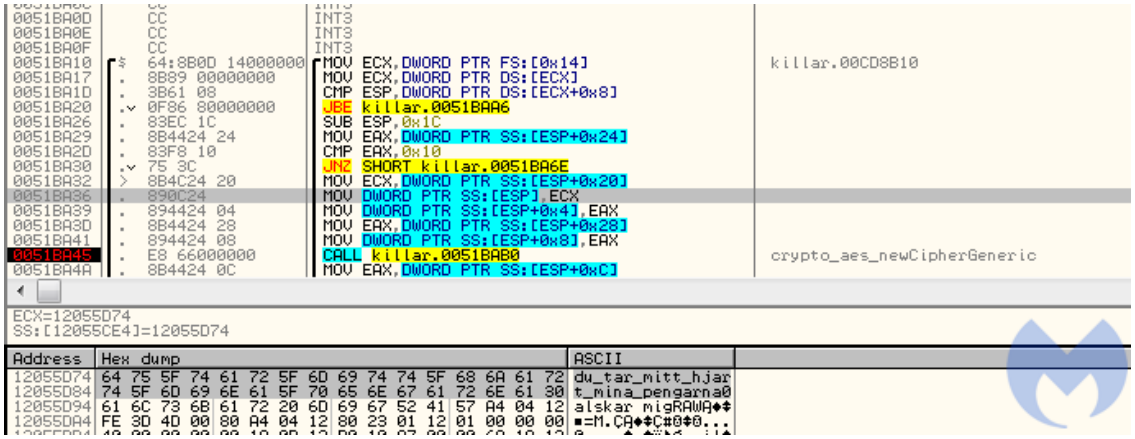


The content of the output file (with .VAGGEN extension) contains:

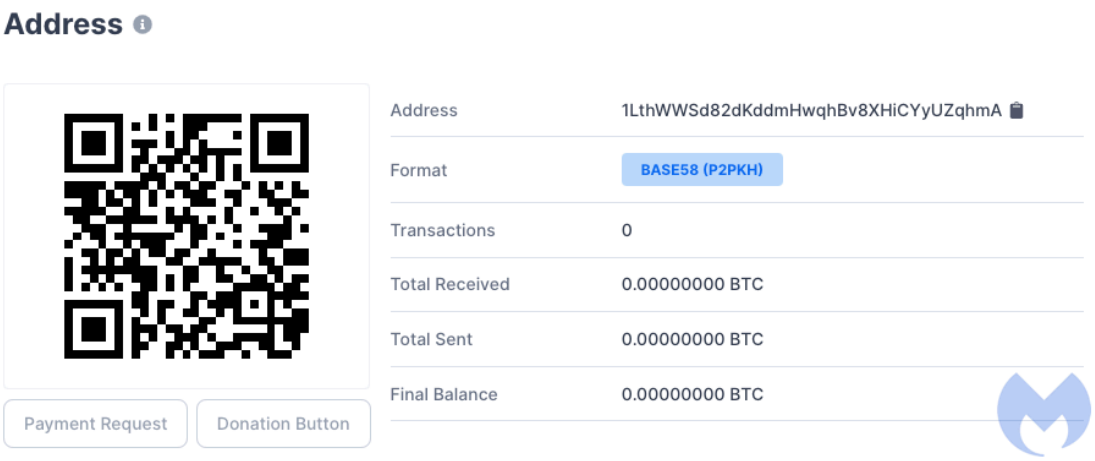
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

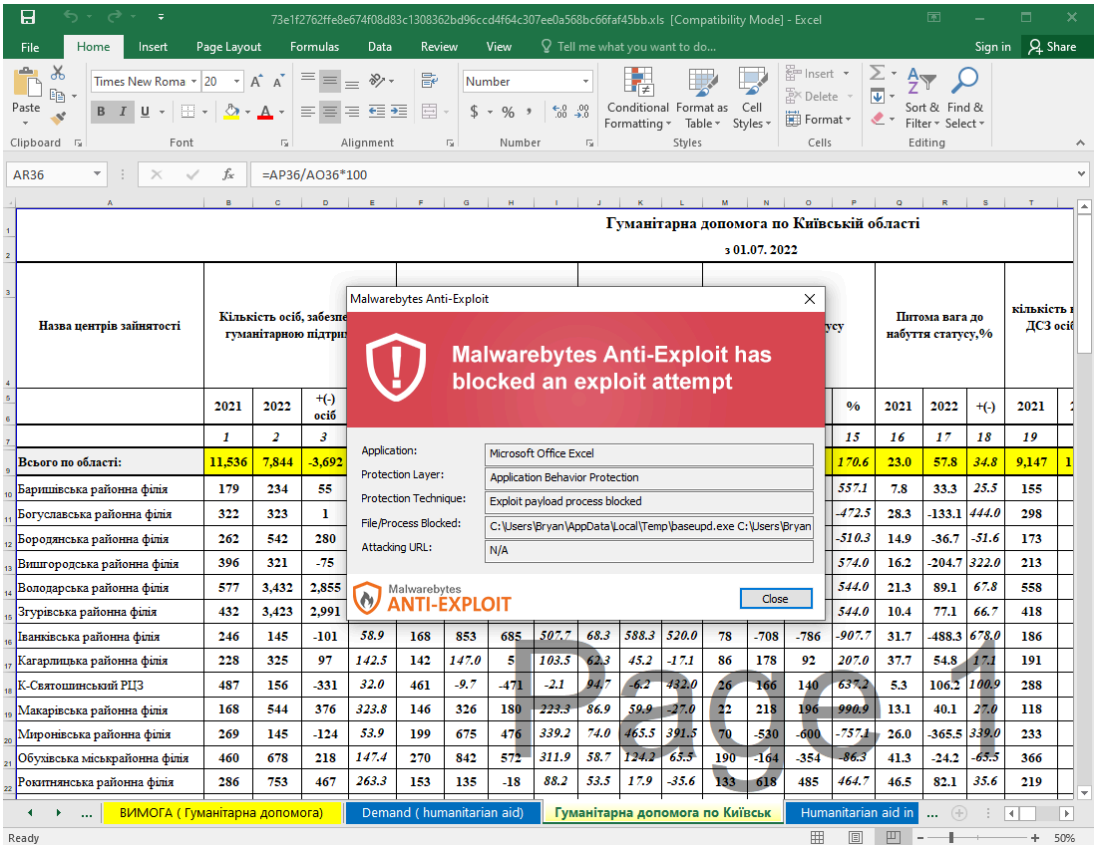
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

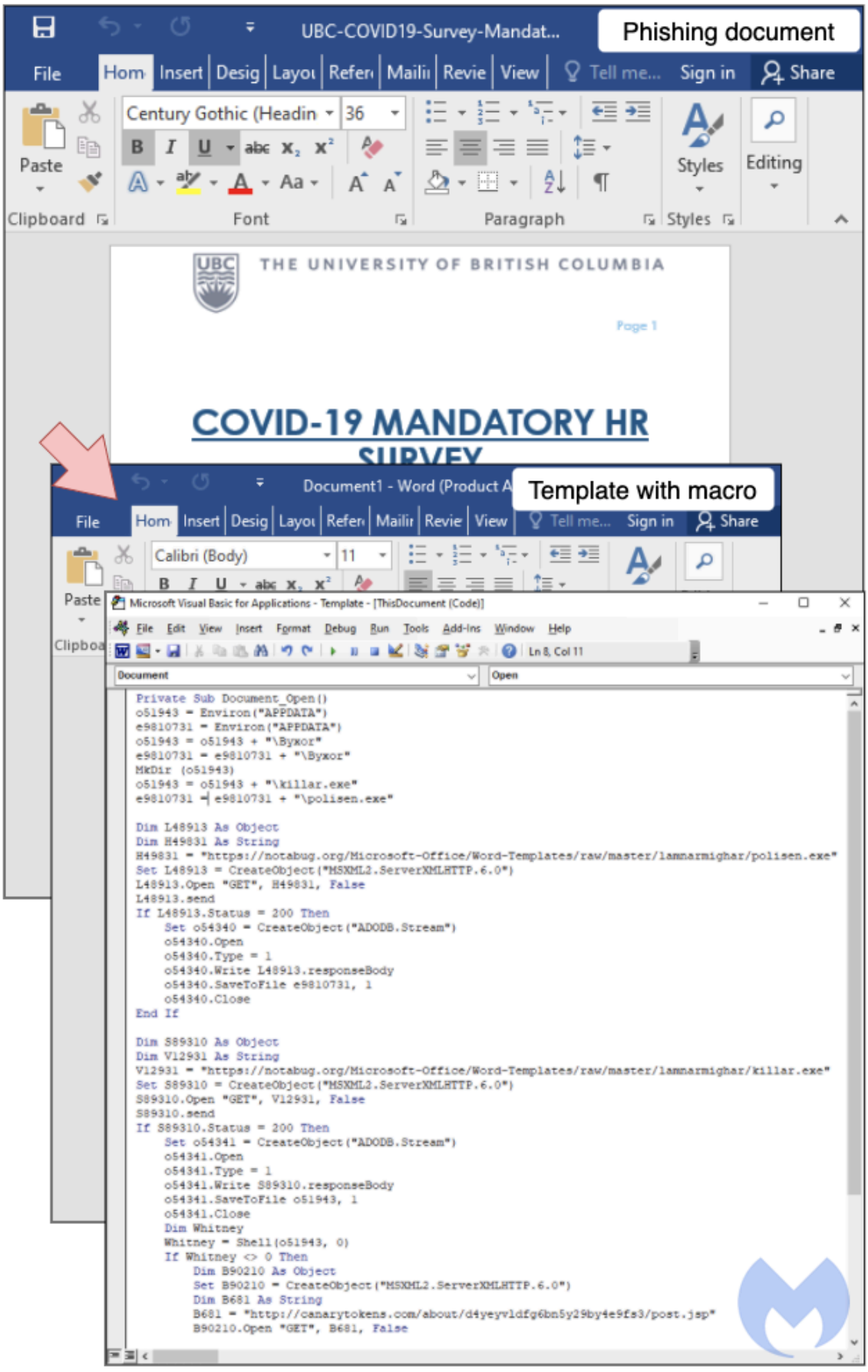
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

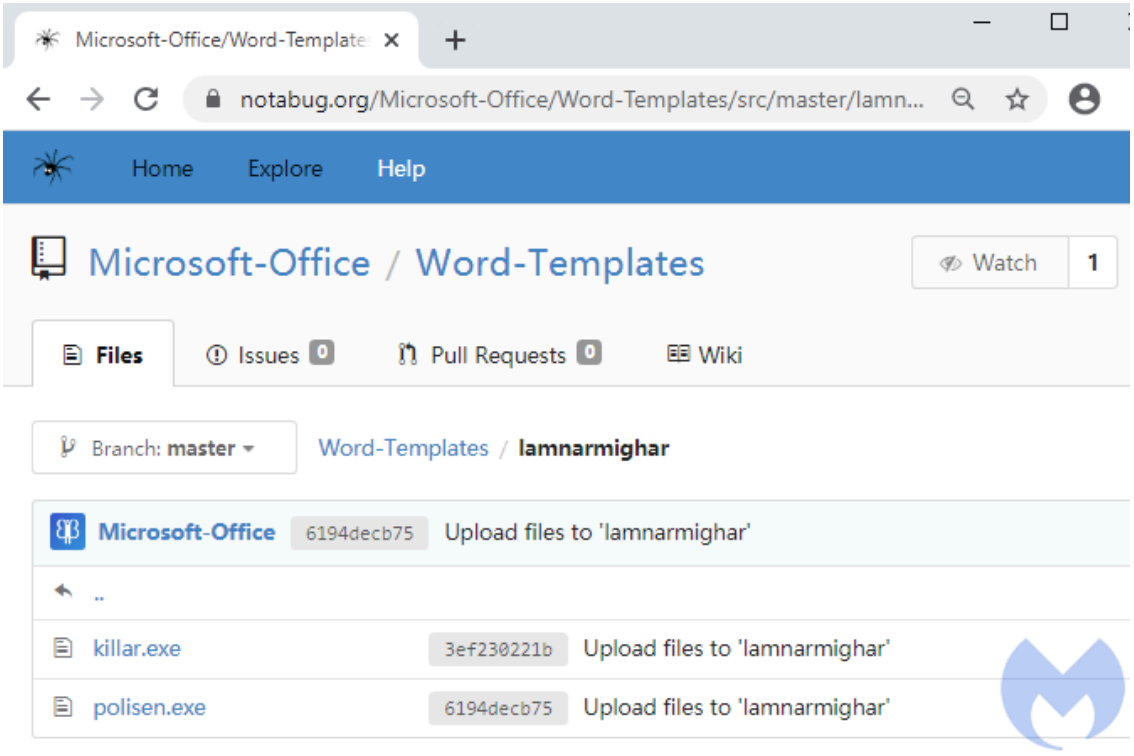
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



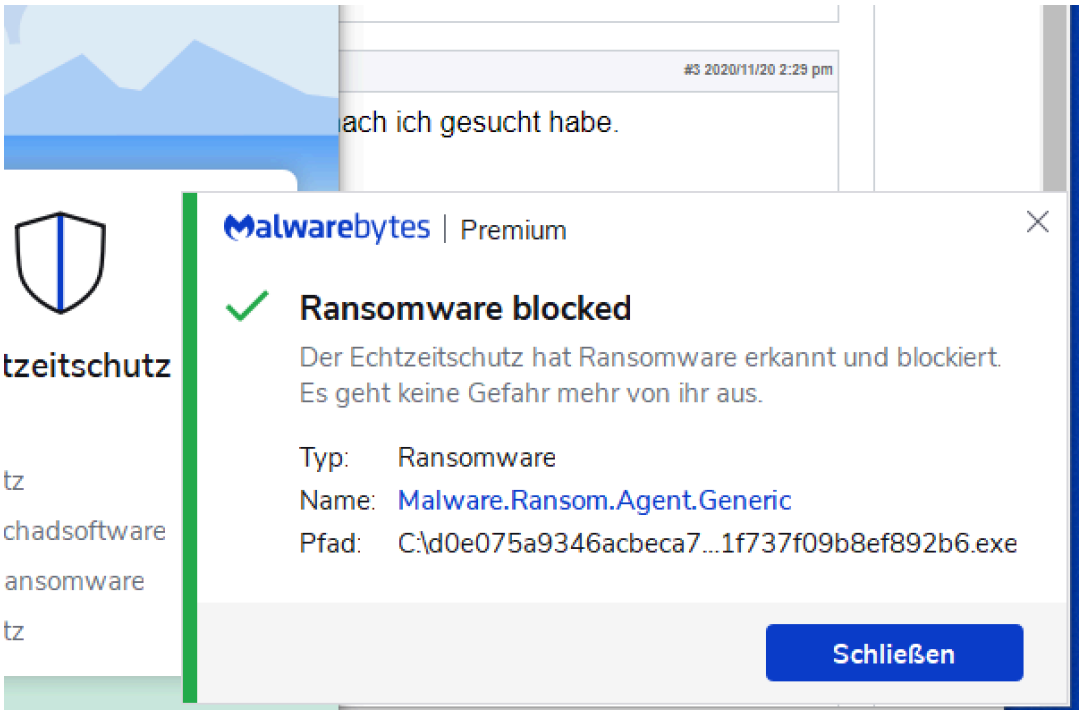
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

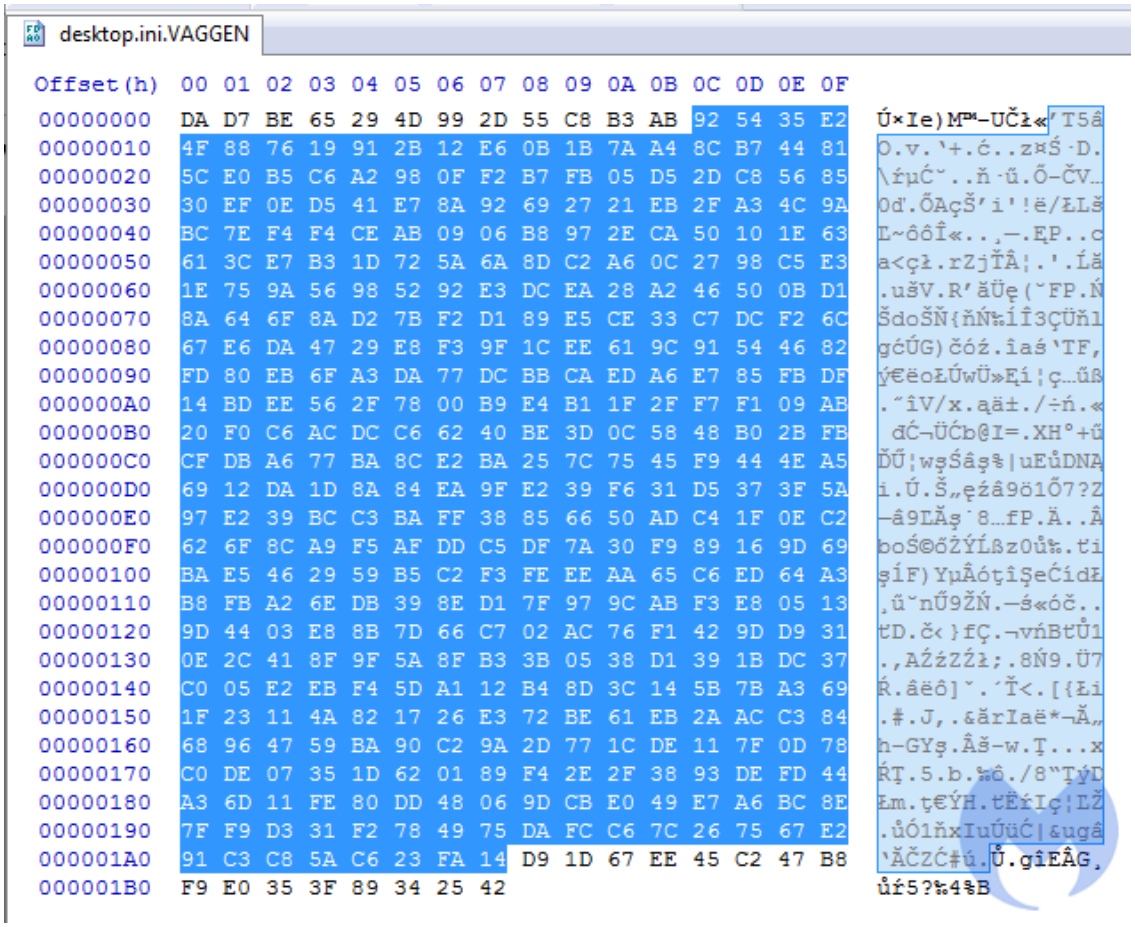
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

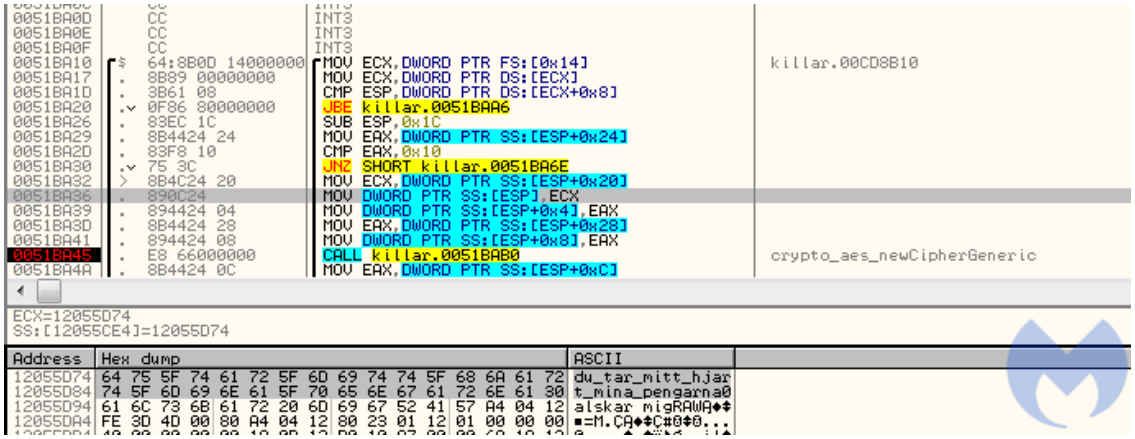


The content of the output file (with .VAGGEN extension) contains:

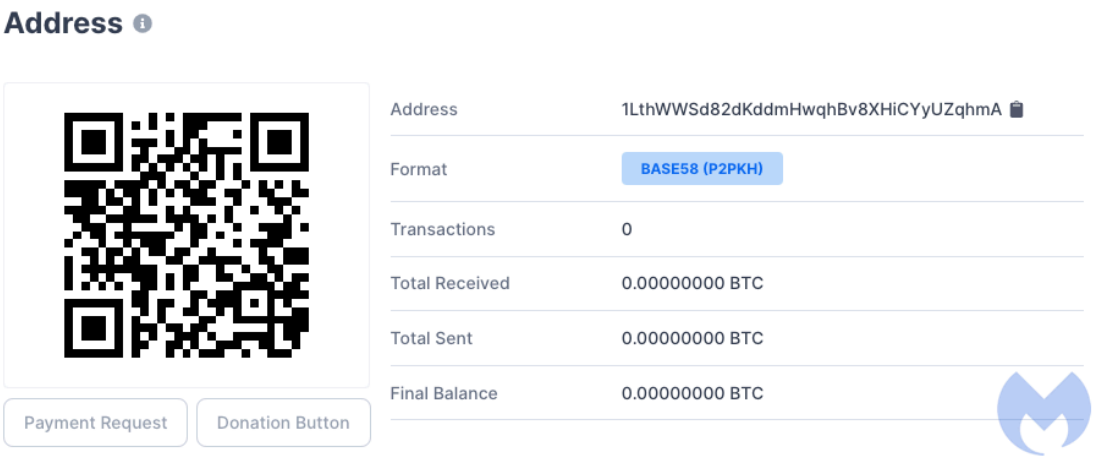
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

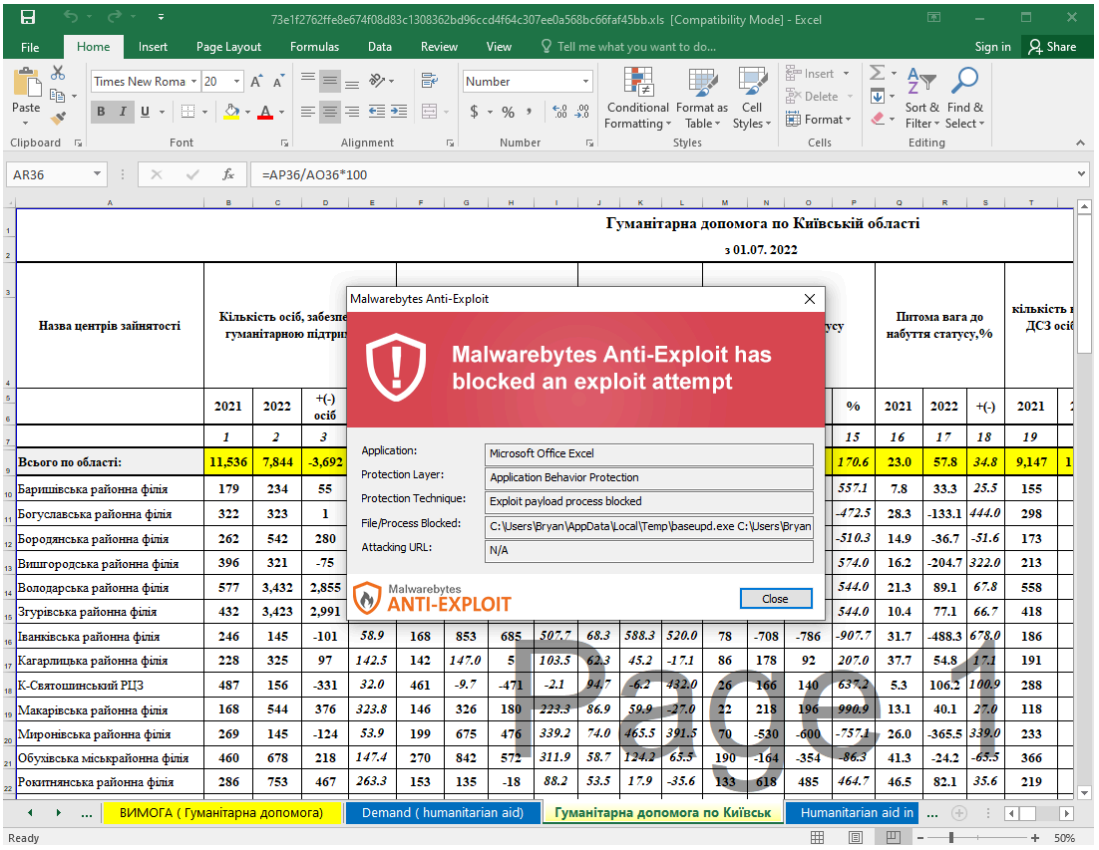
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

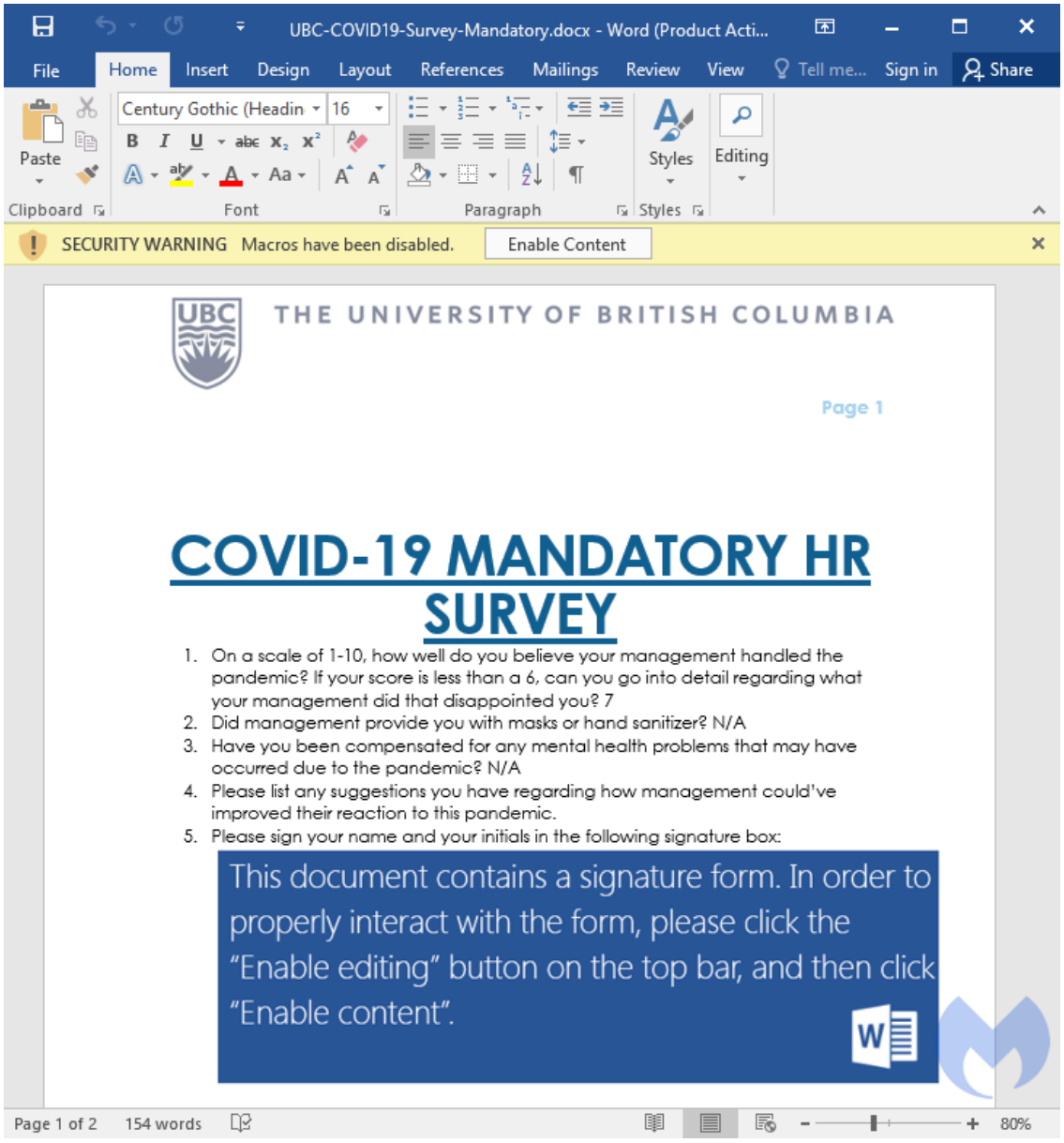
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

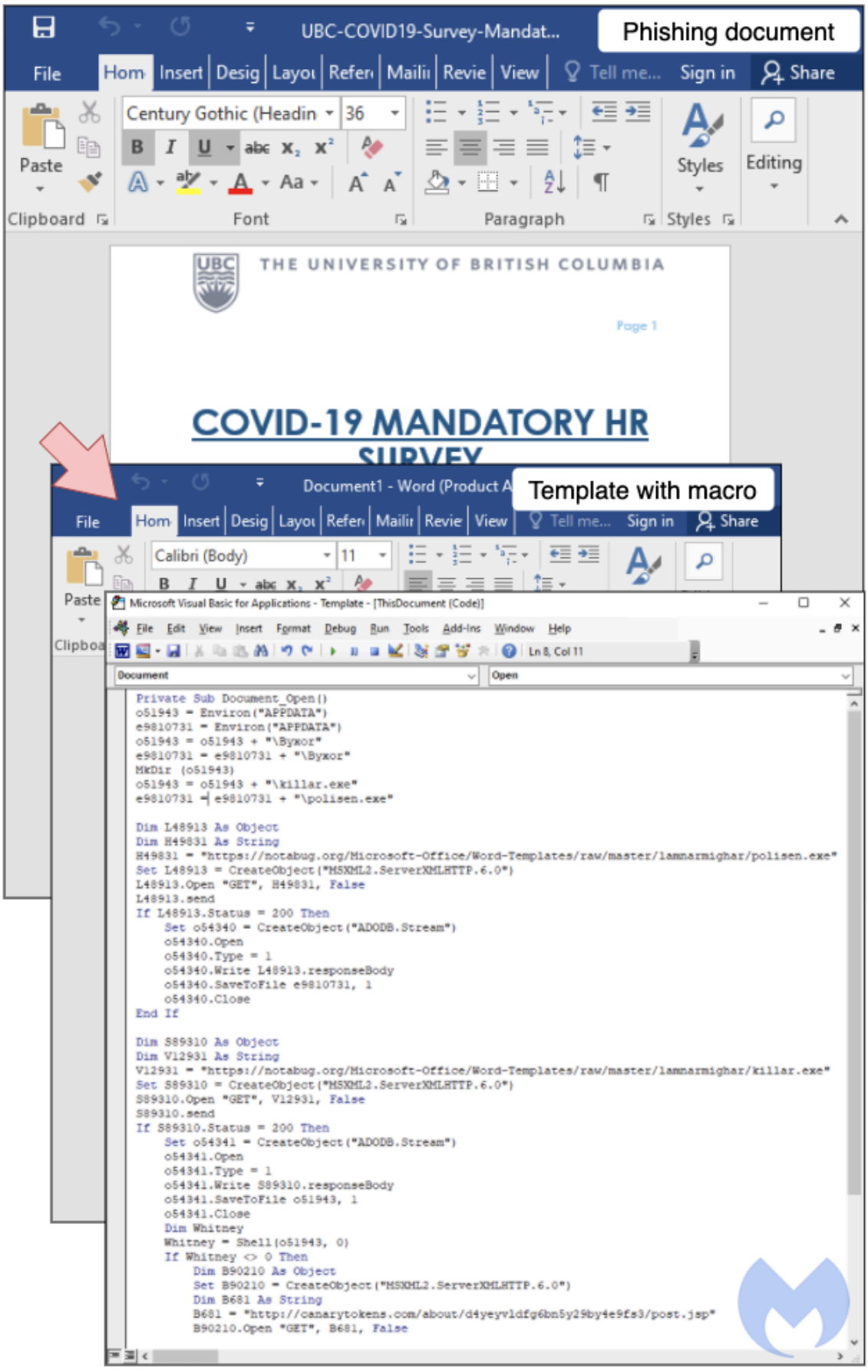
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

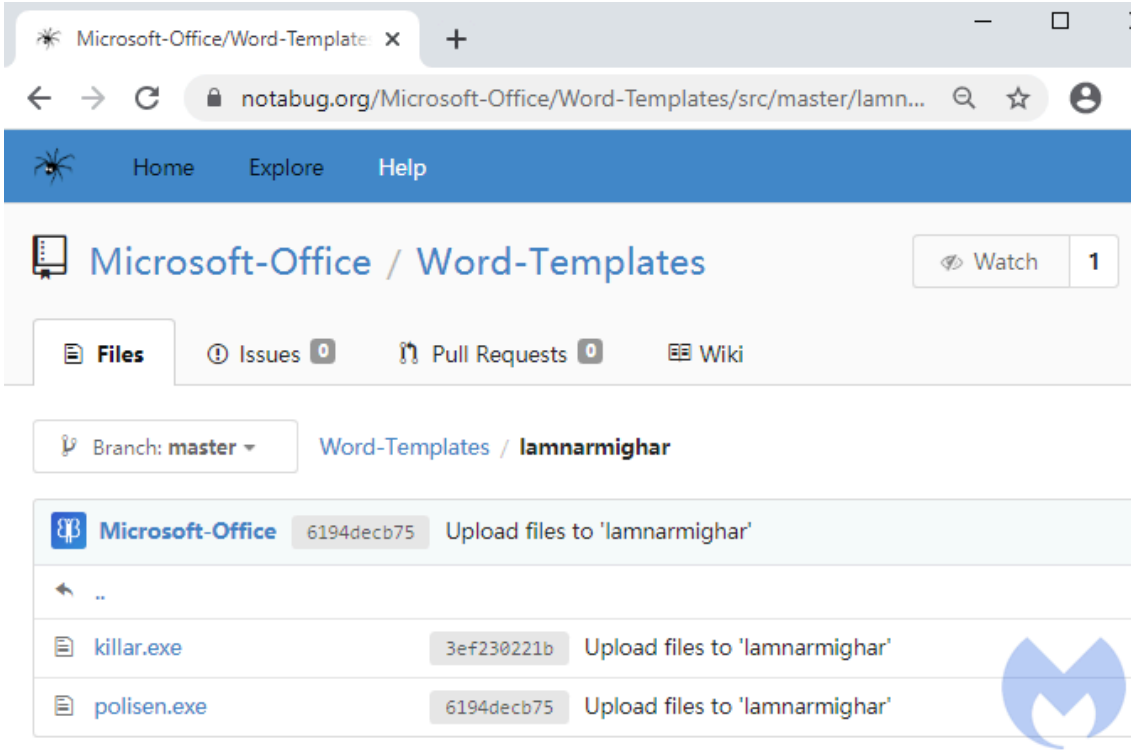
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



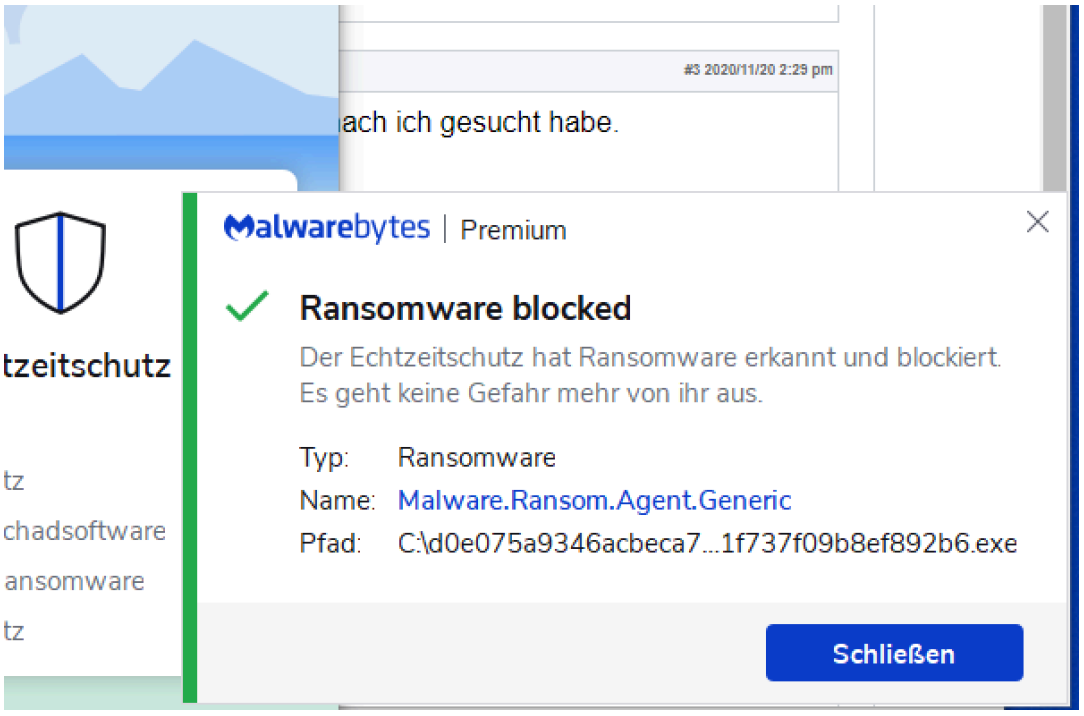
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

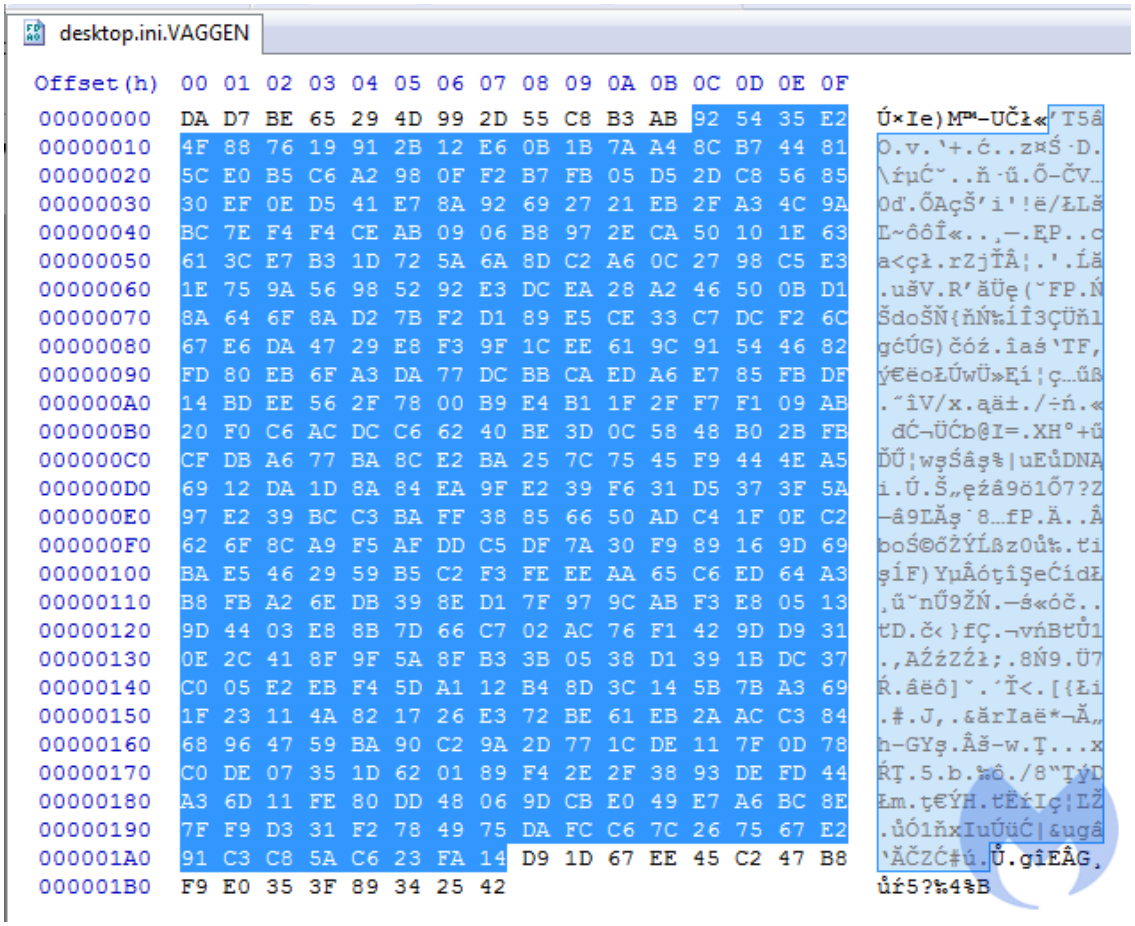
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

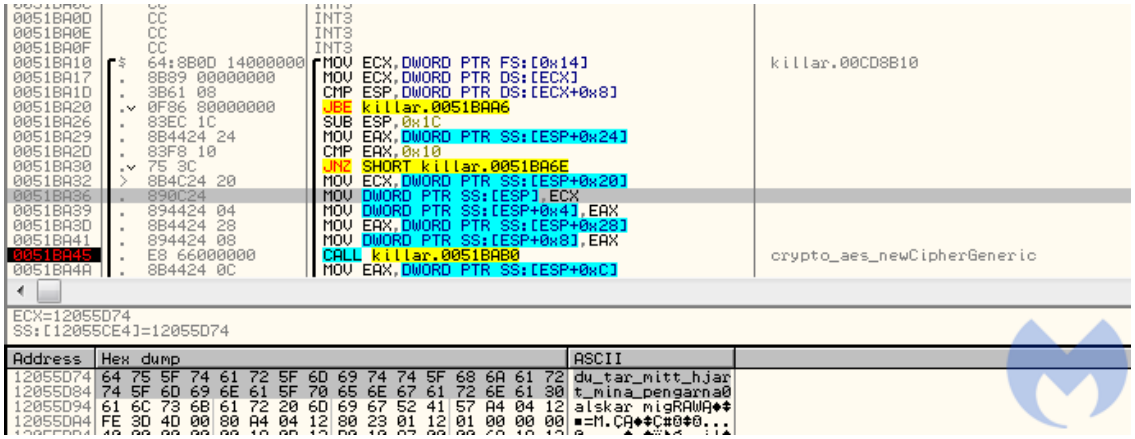


The content of the output file (with .VAGGEN extension) contains:

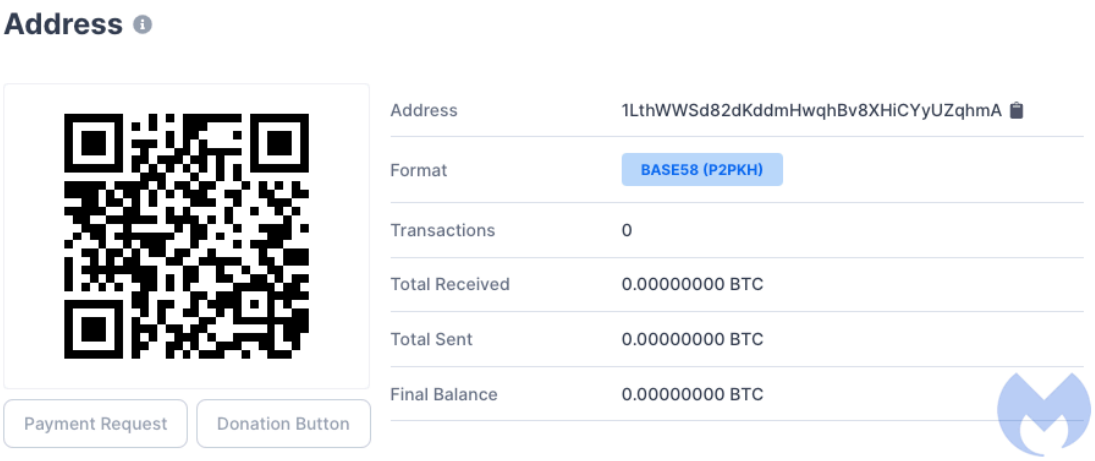
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

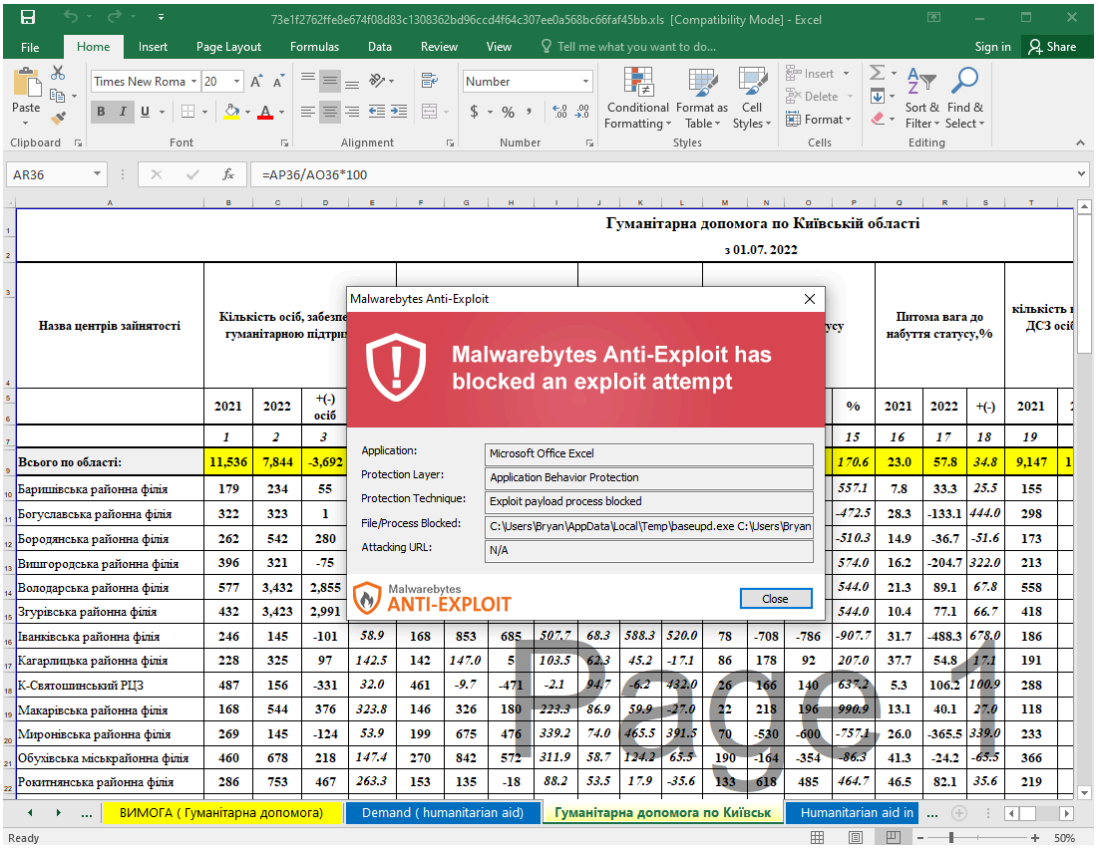
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

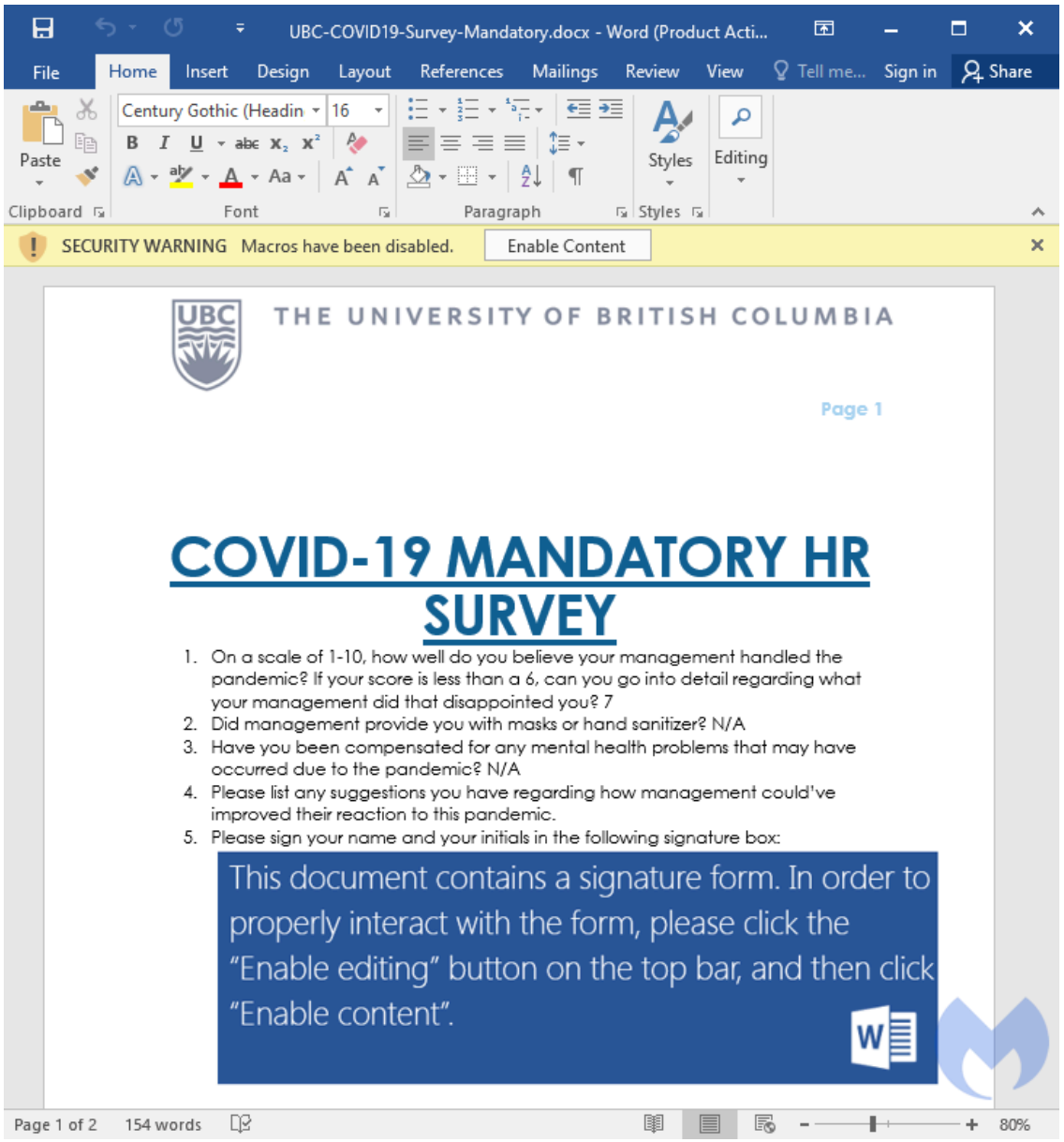
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

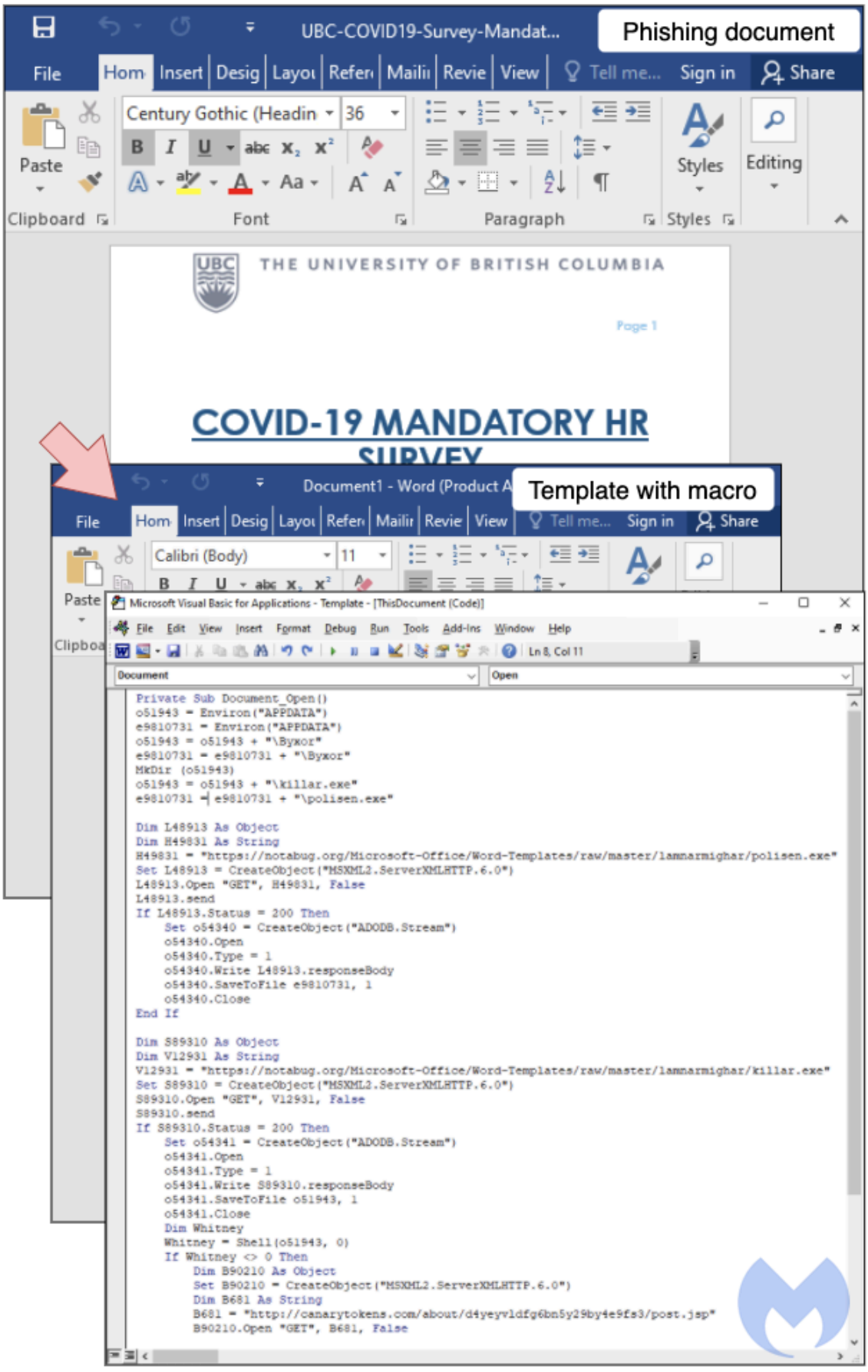
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

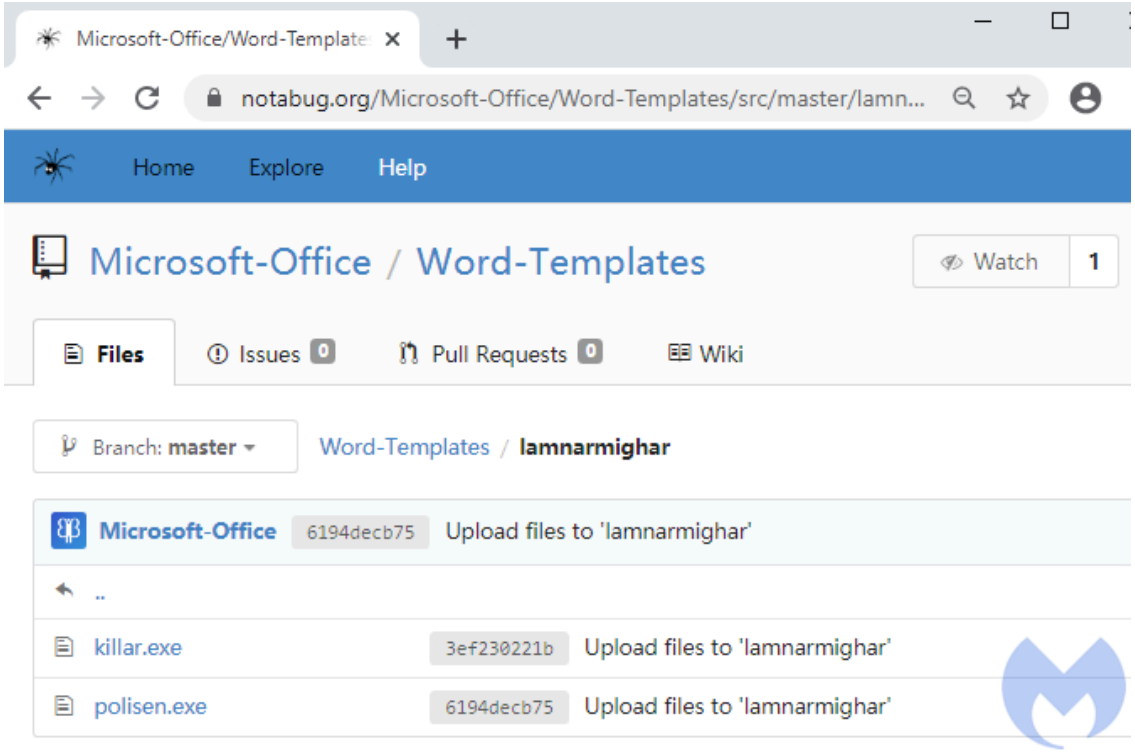
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



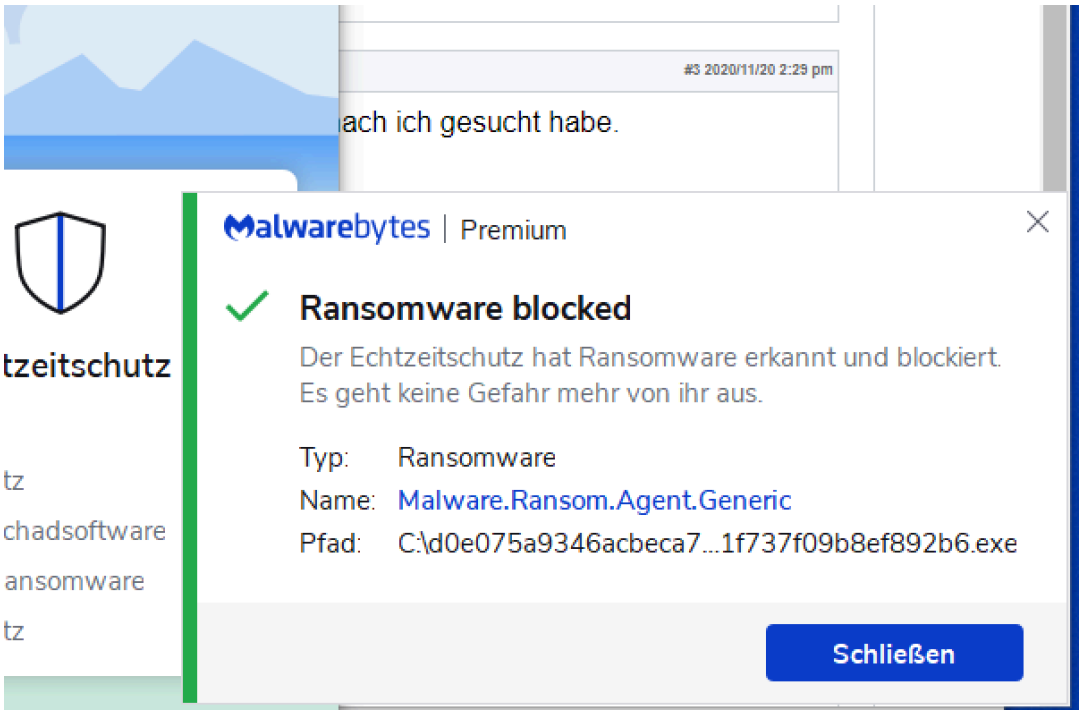
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

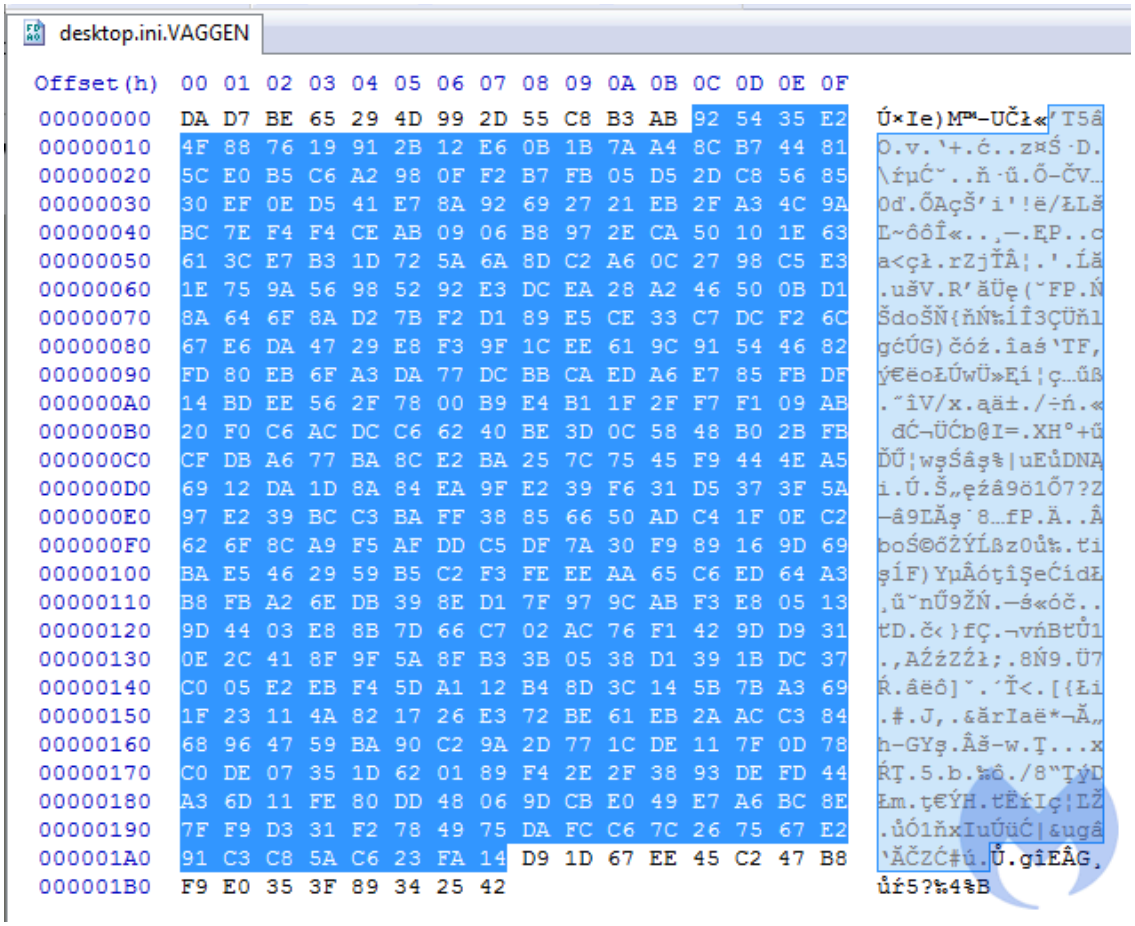
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

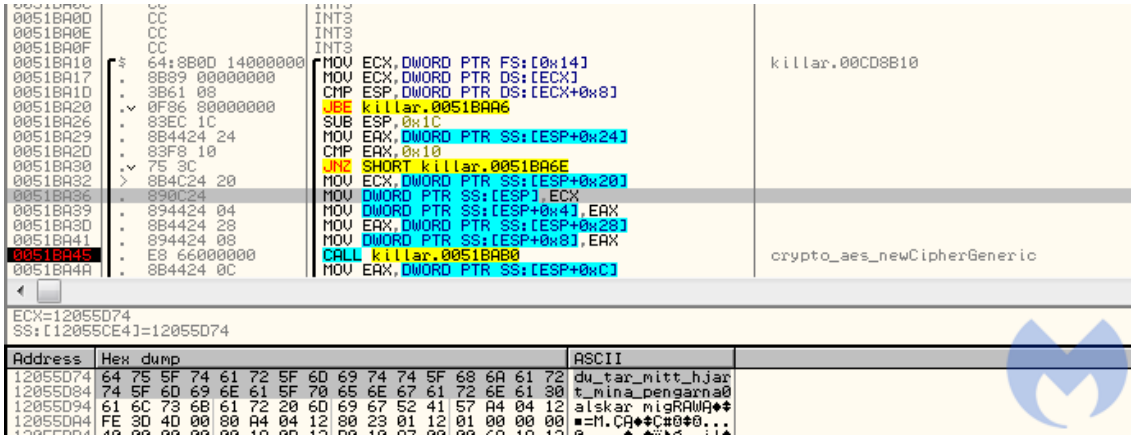


The content of the output file (with .VAGGEN extension) contains:

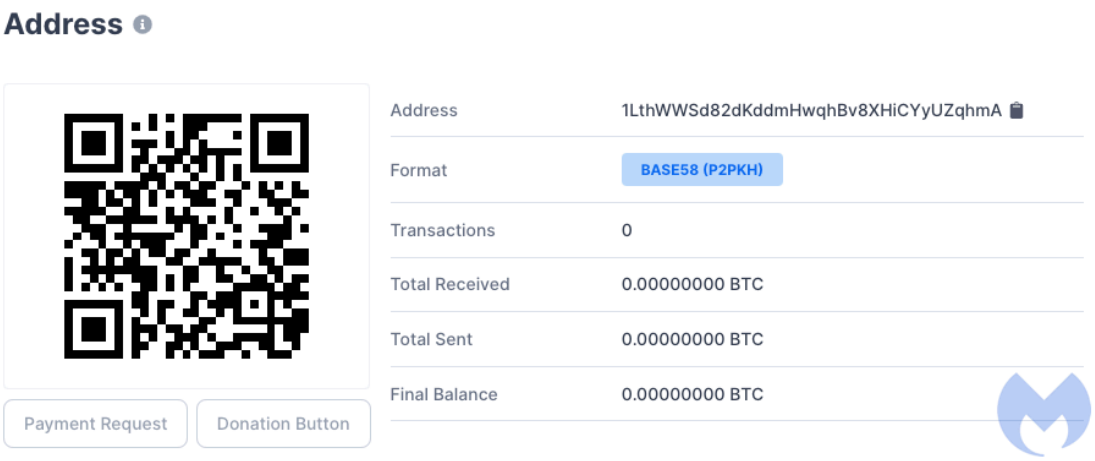
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

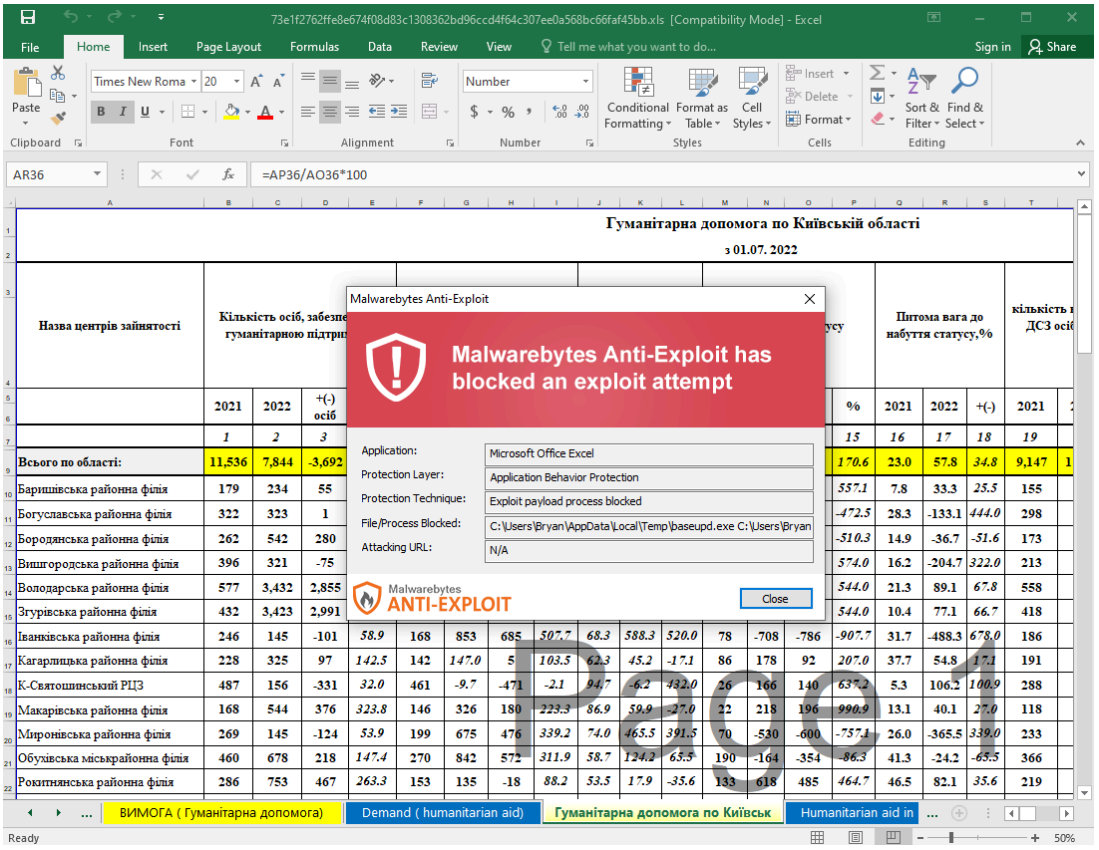
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

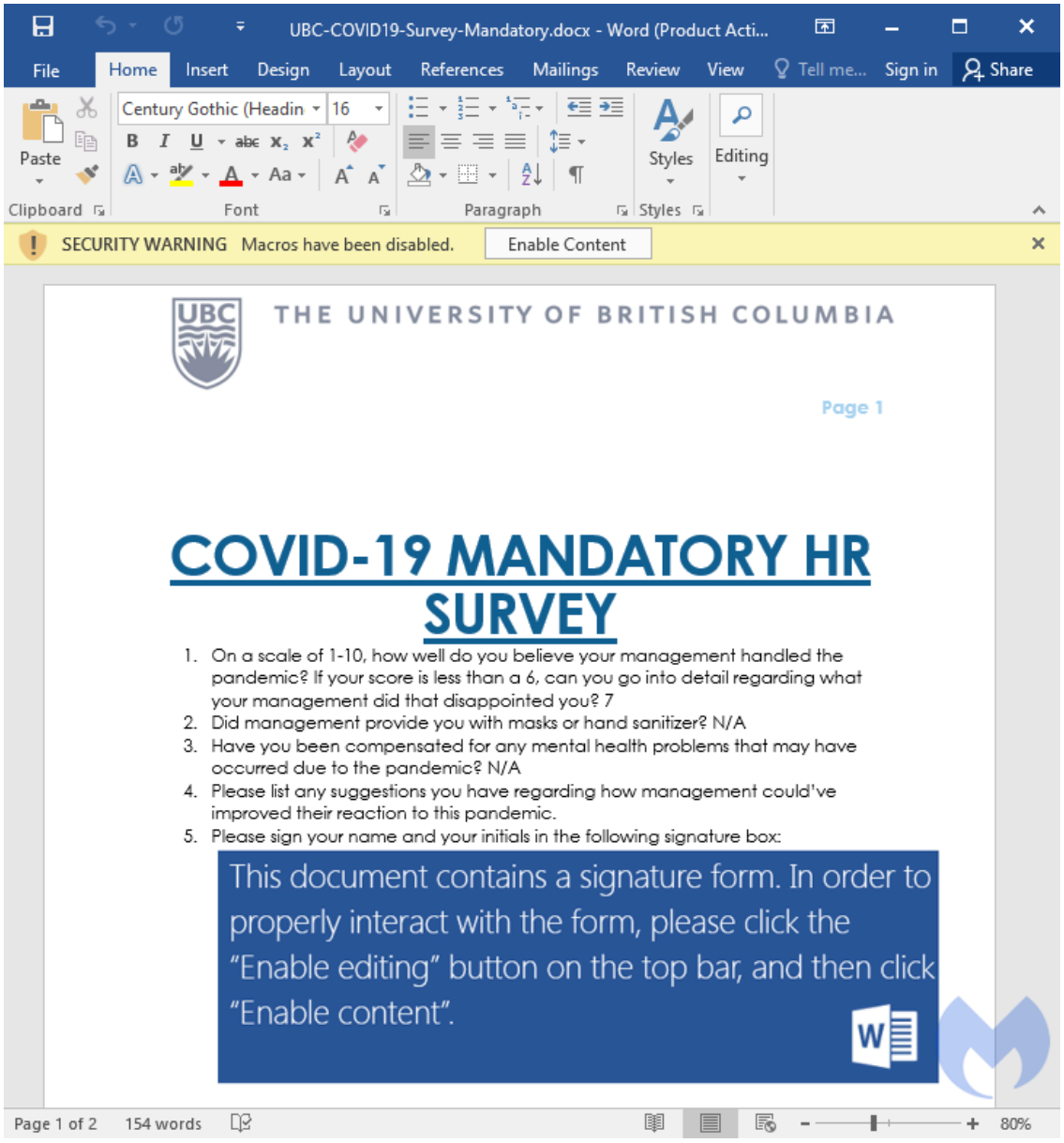
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

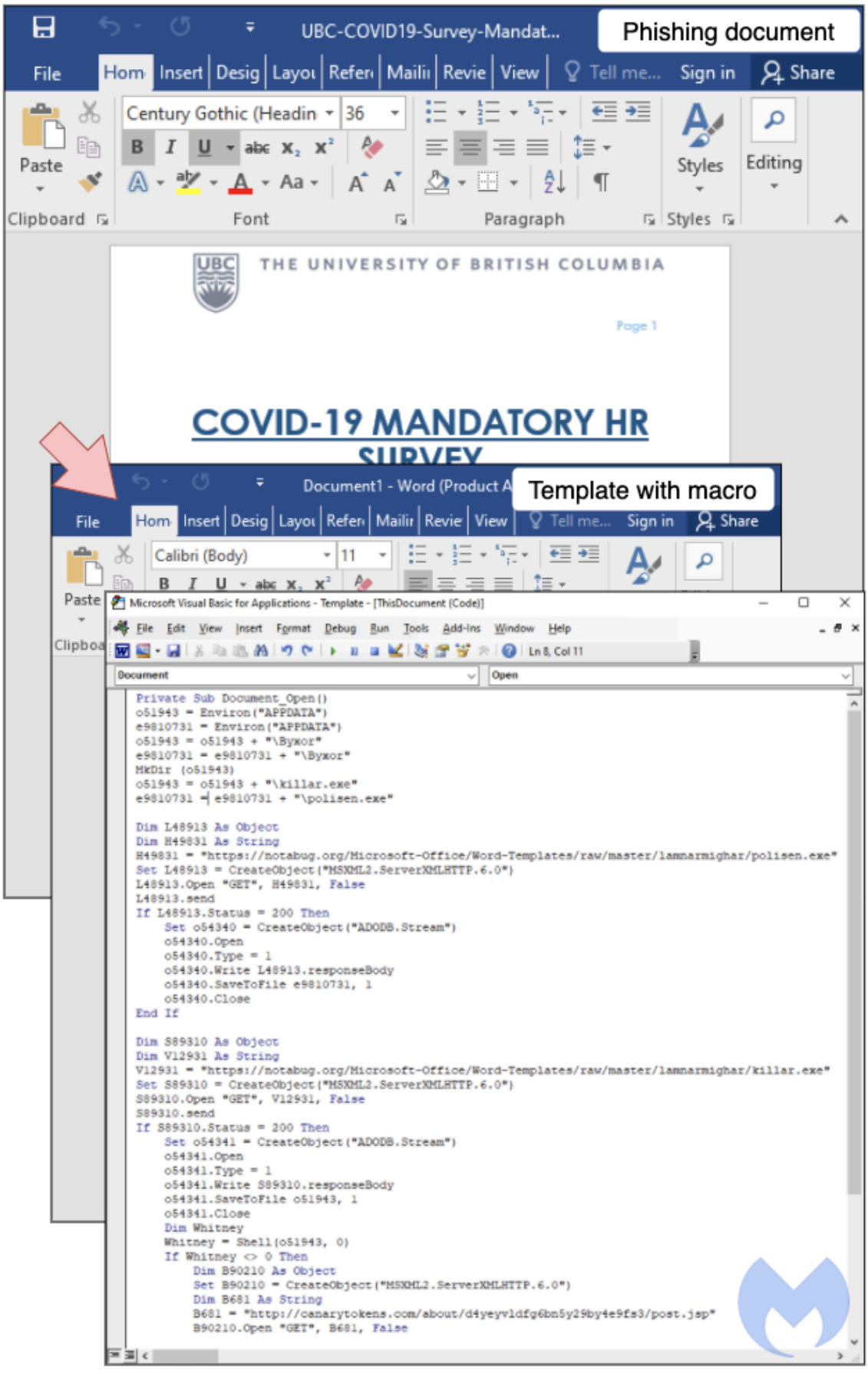
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

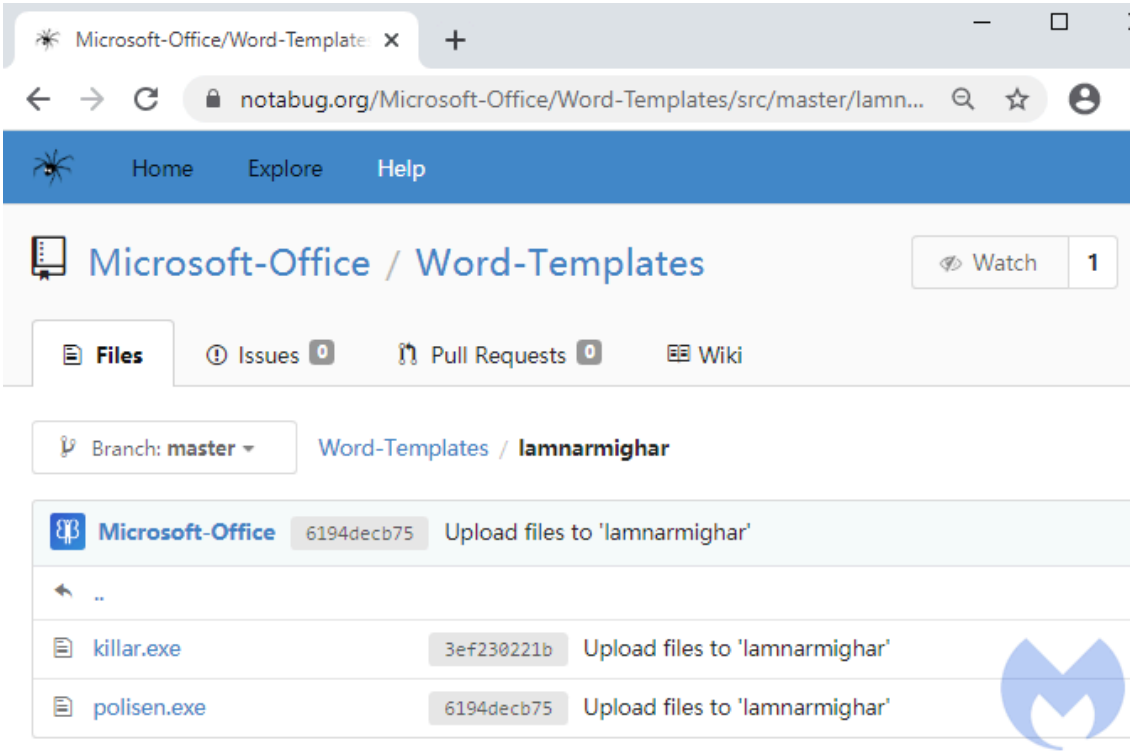
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



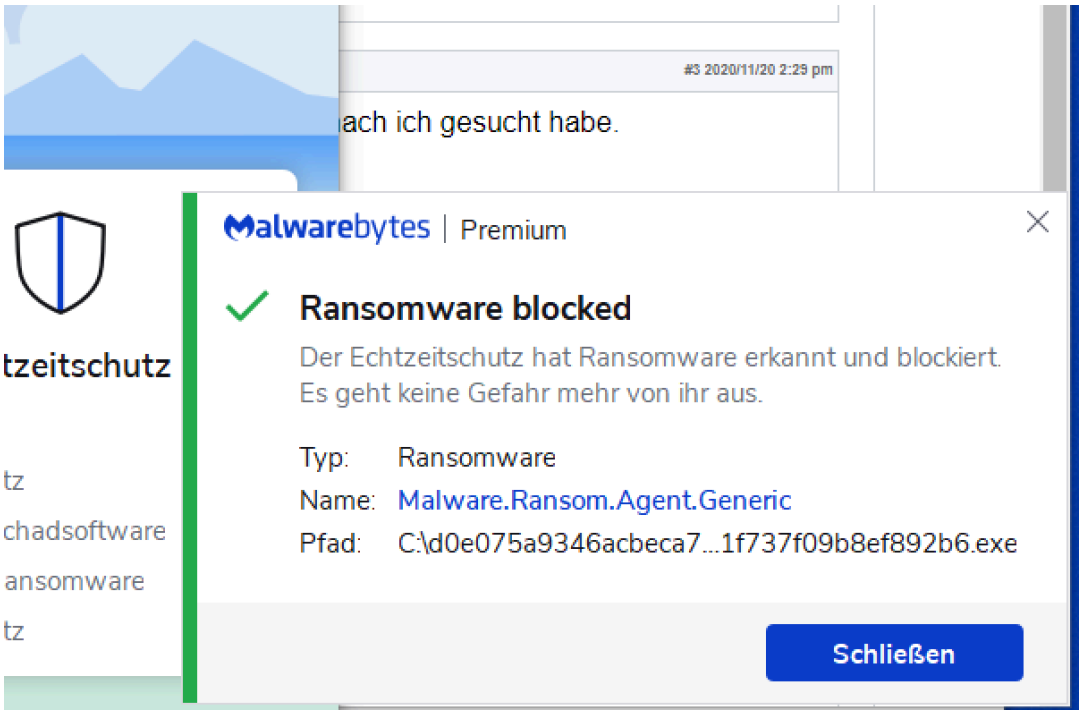
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```



Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```



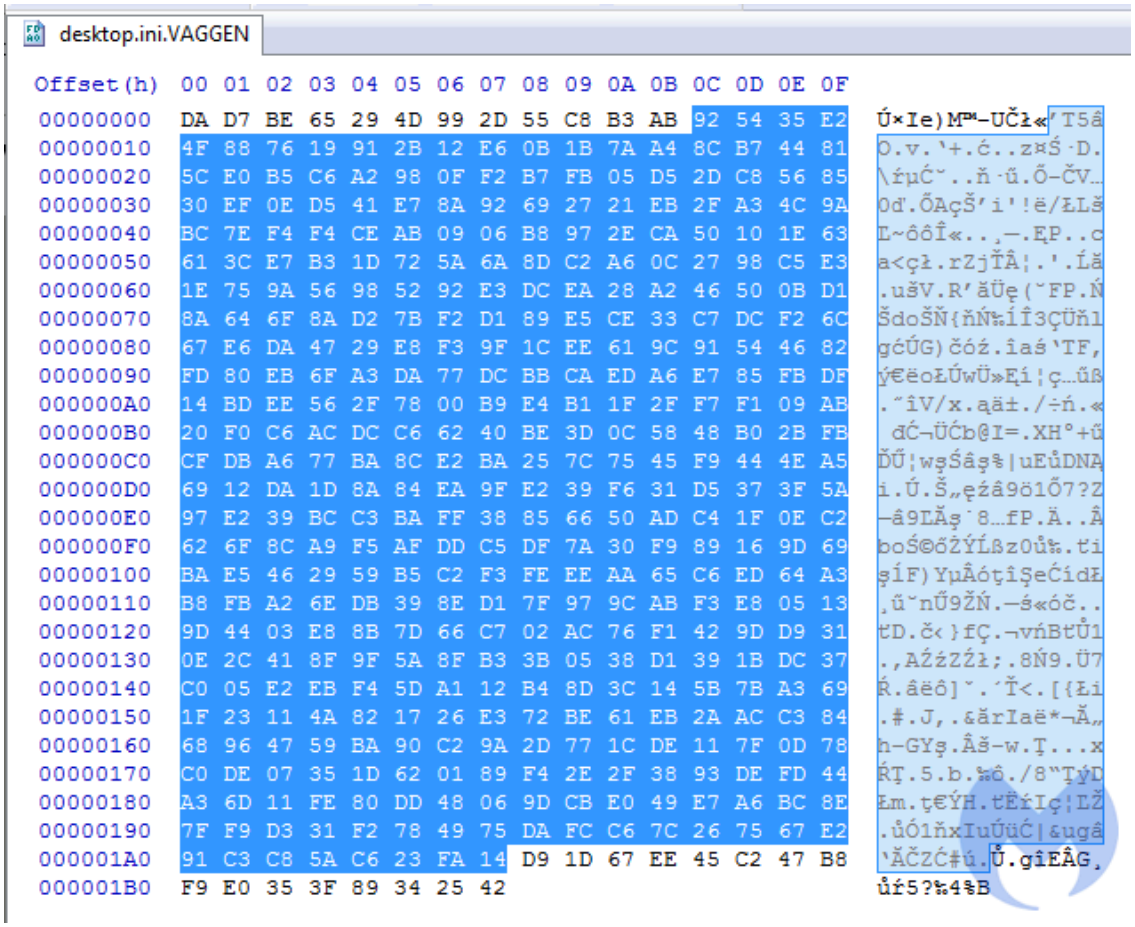
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



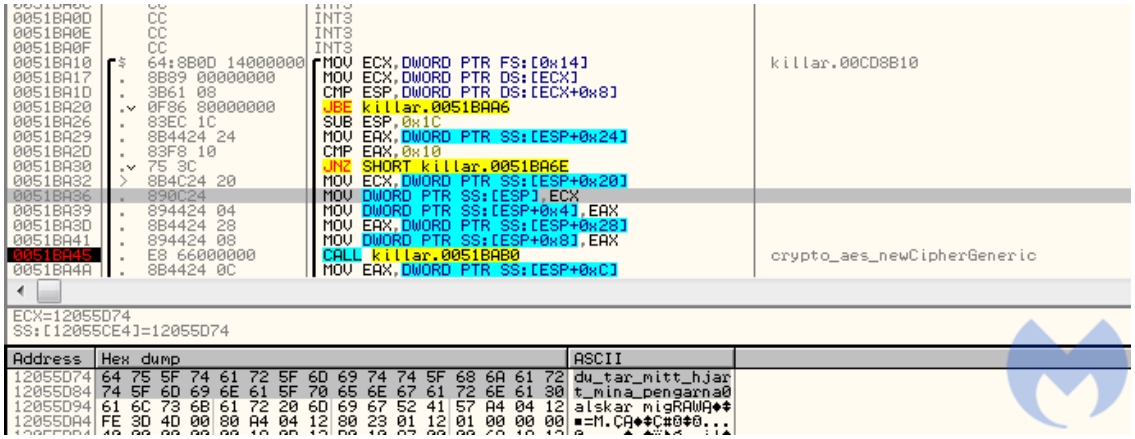
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

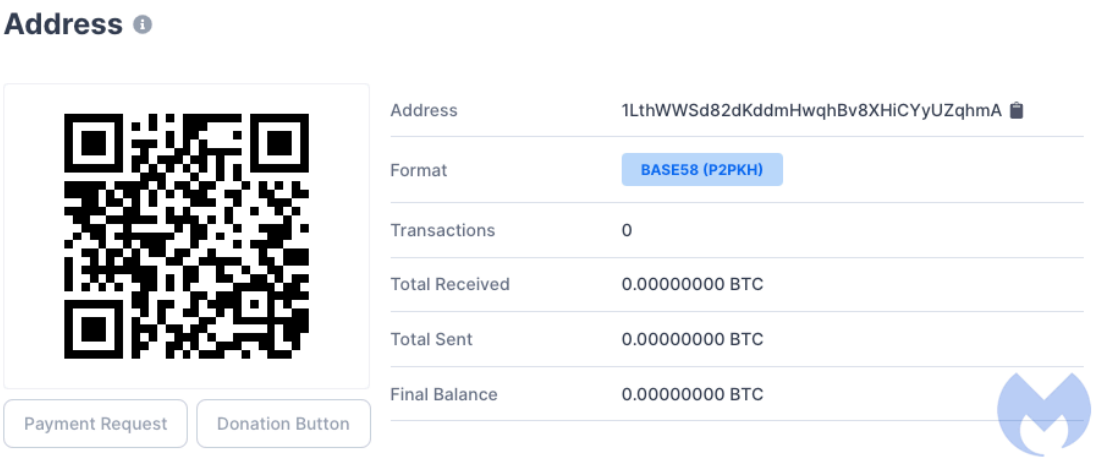
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

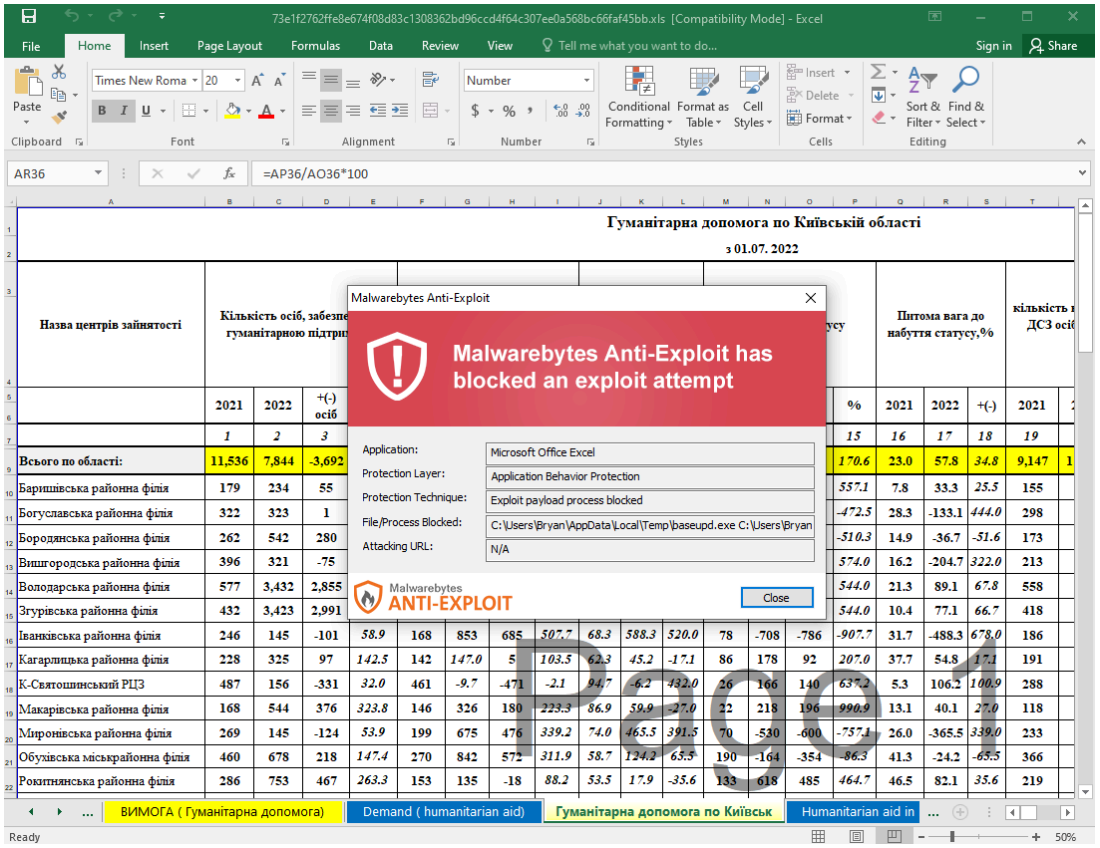
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

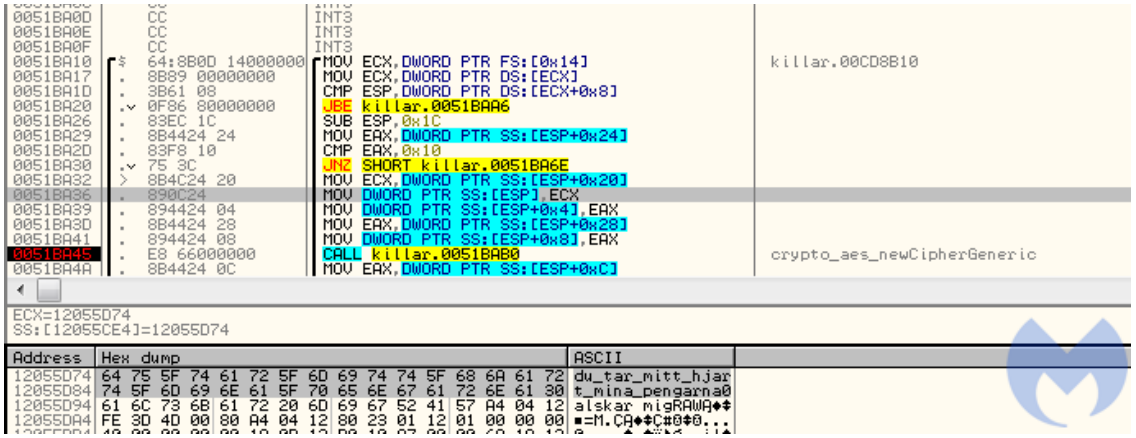
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

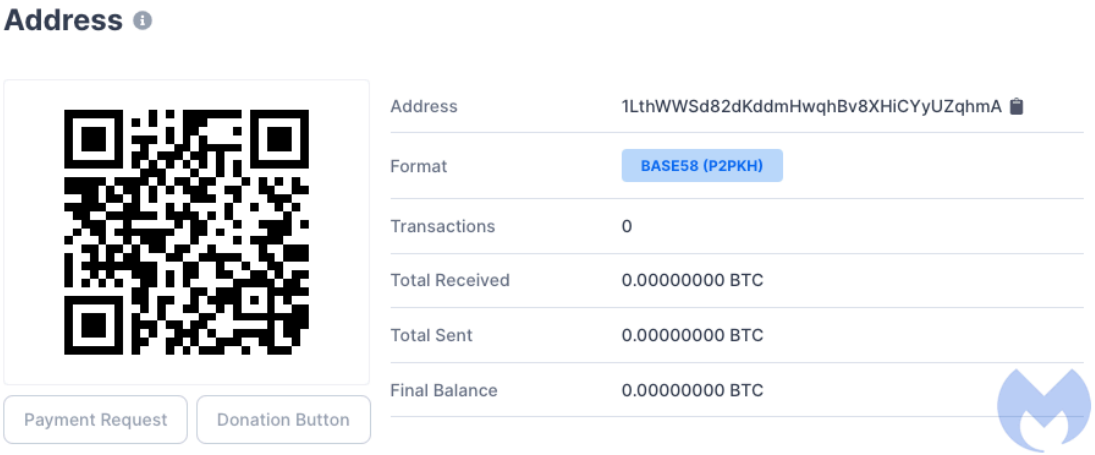
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

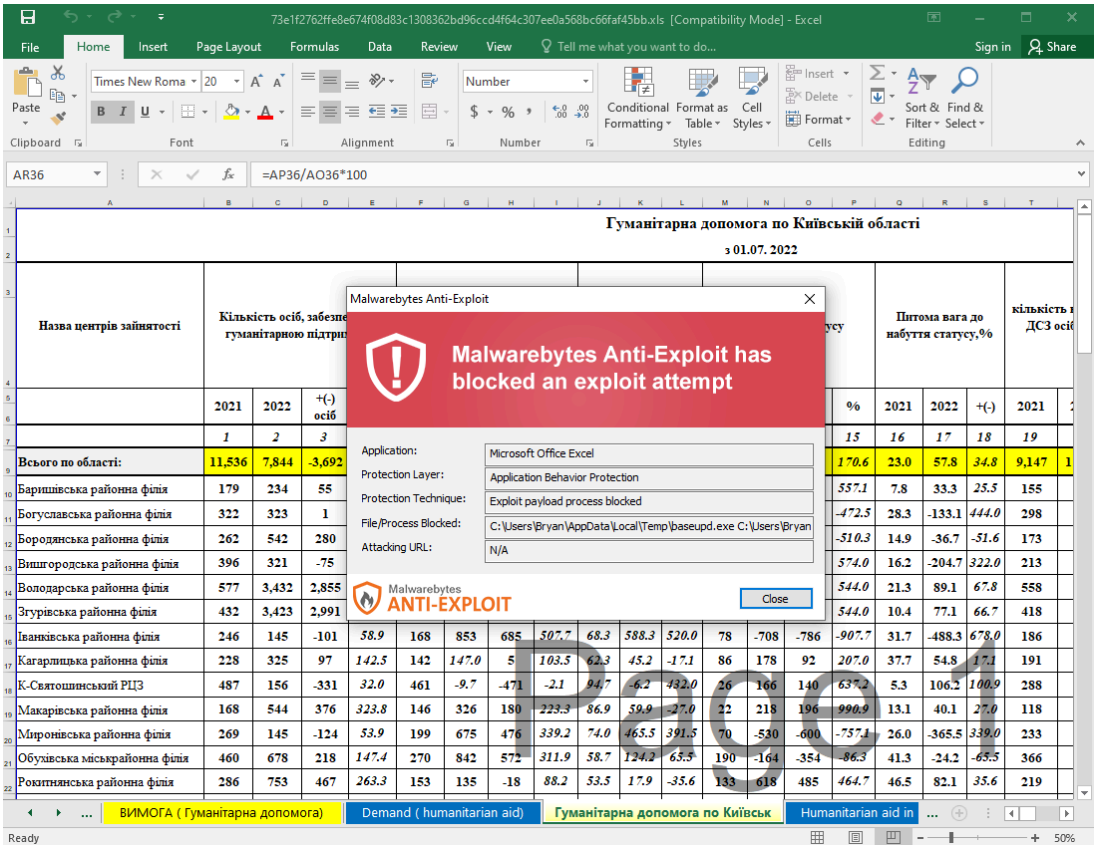
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

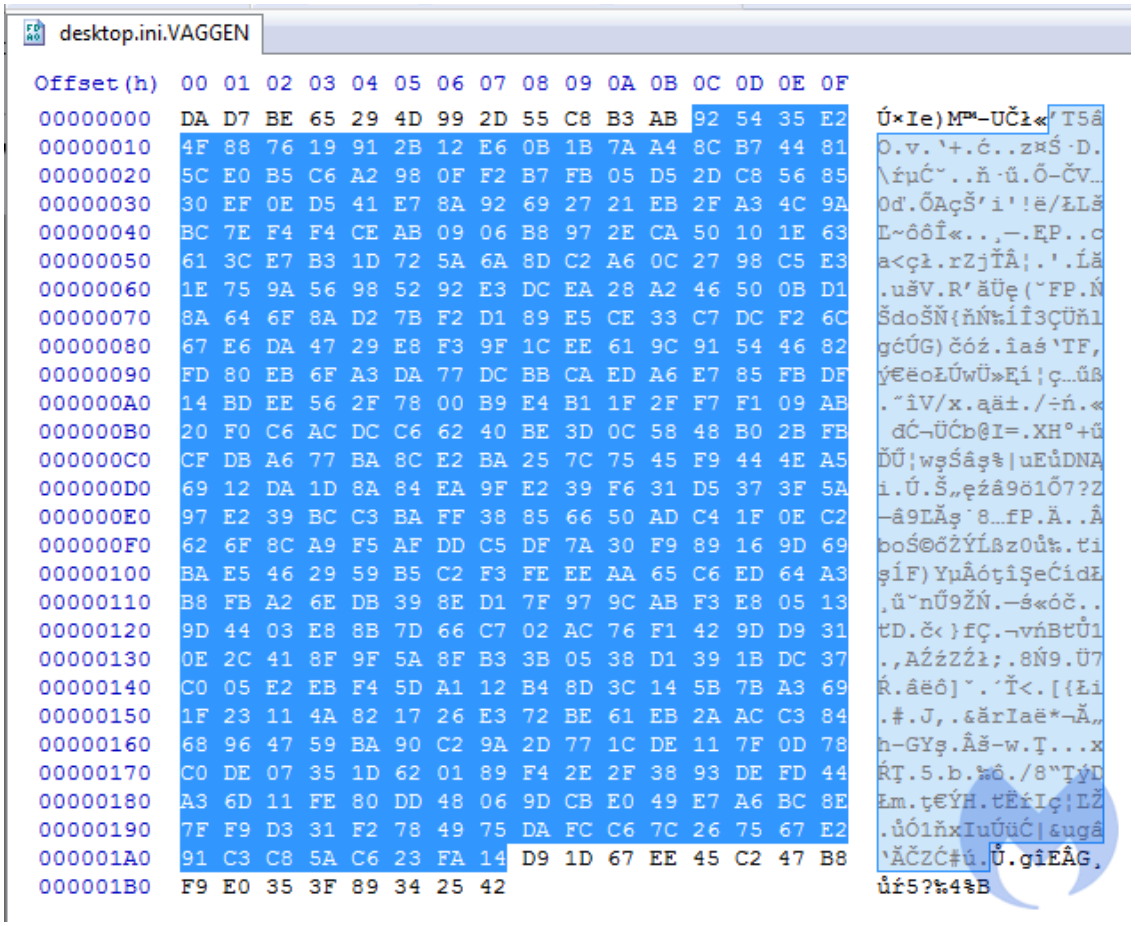
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

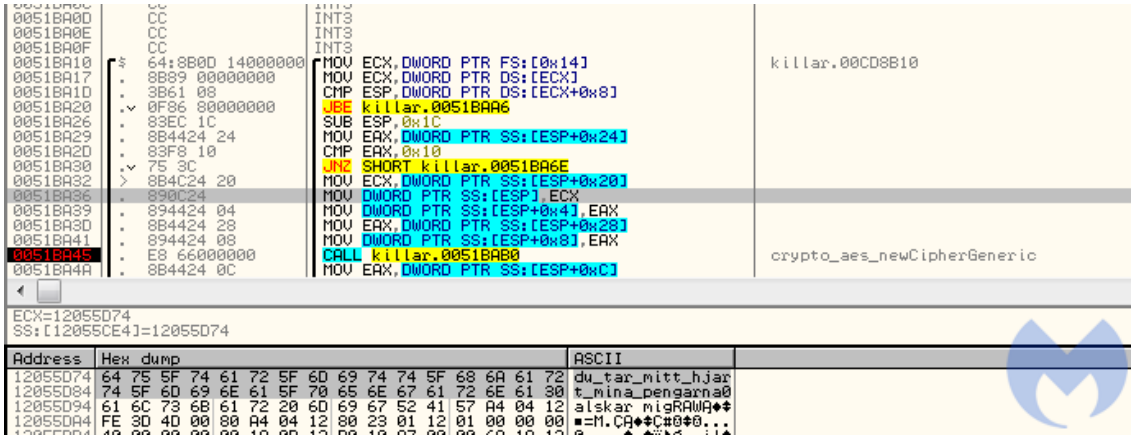
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

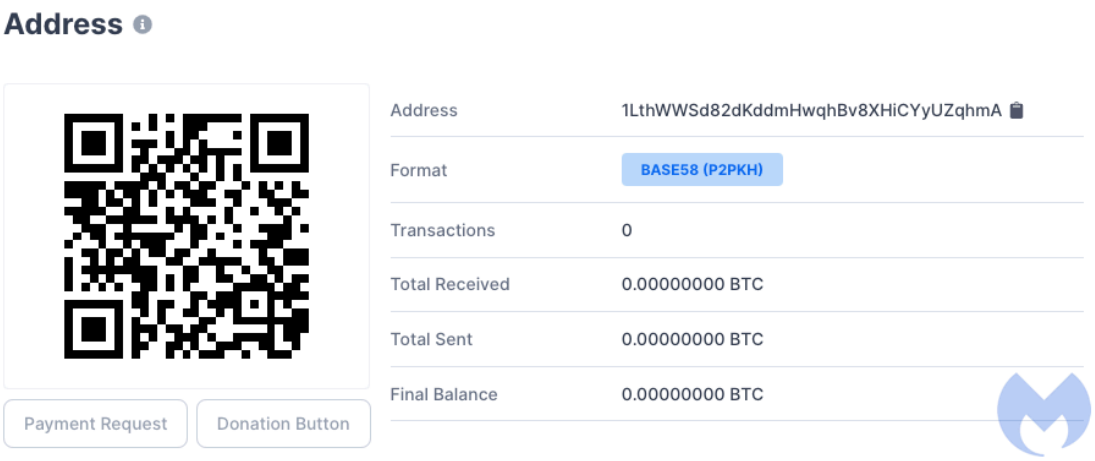
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

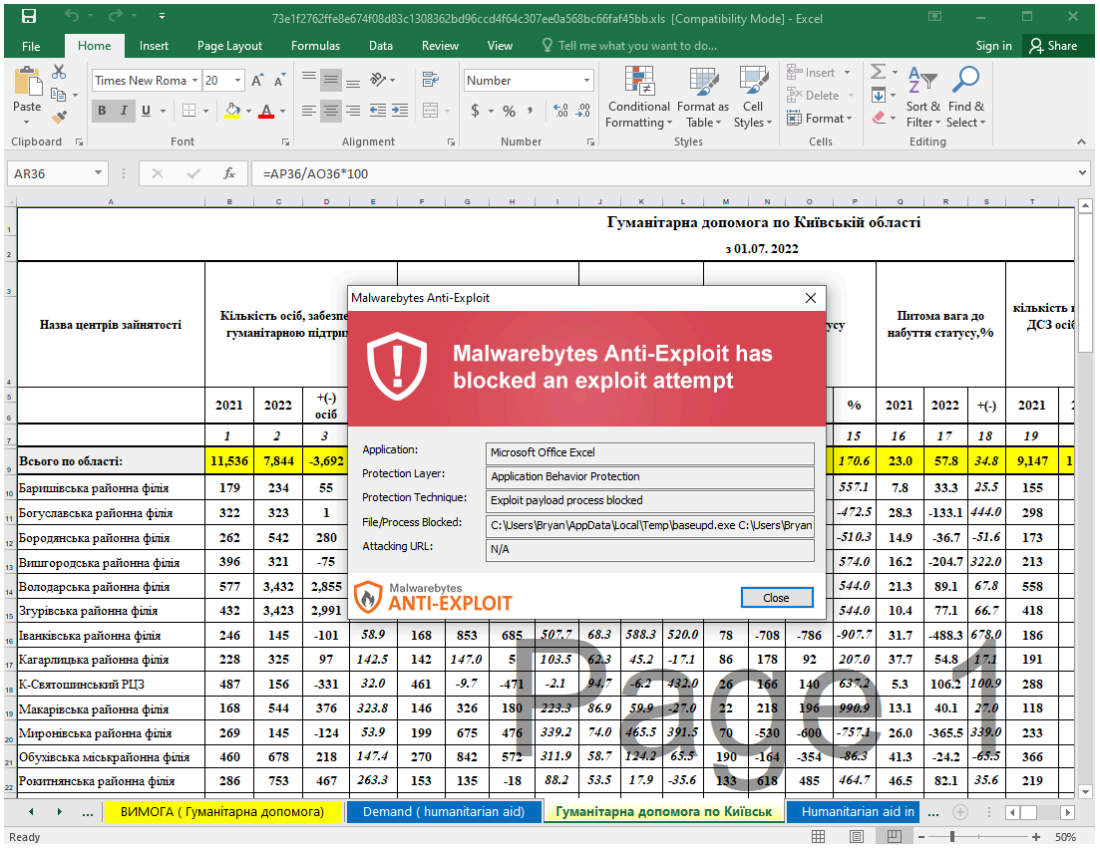
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

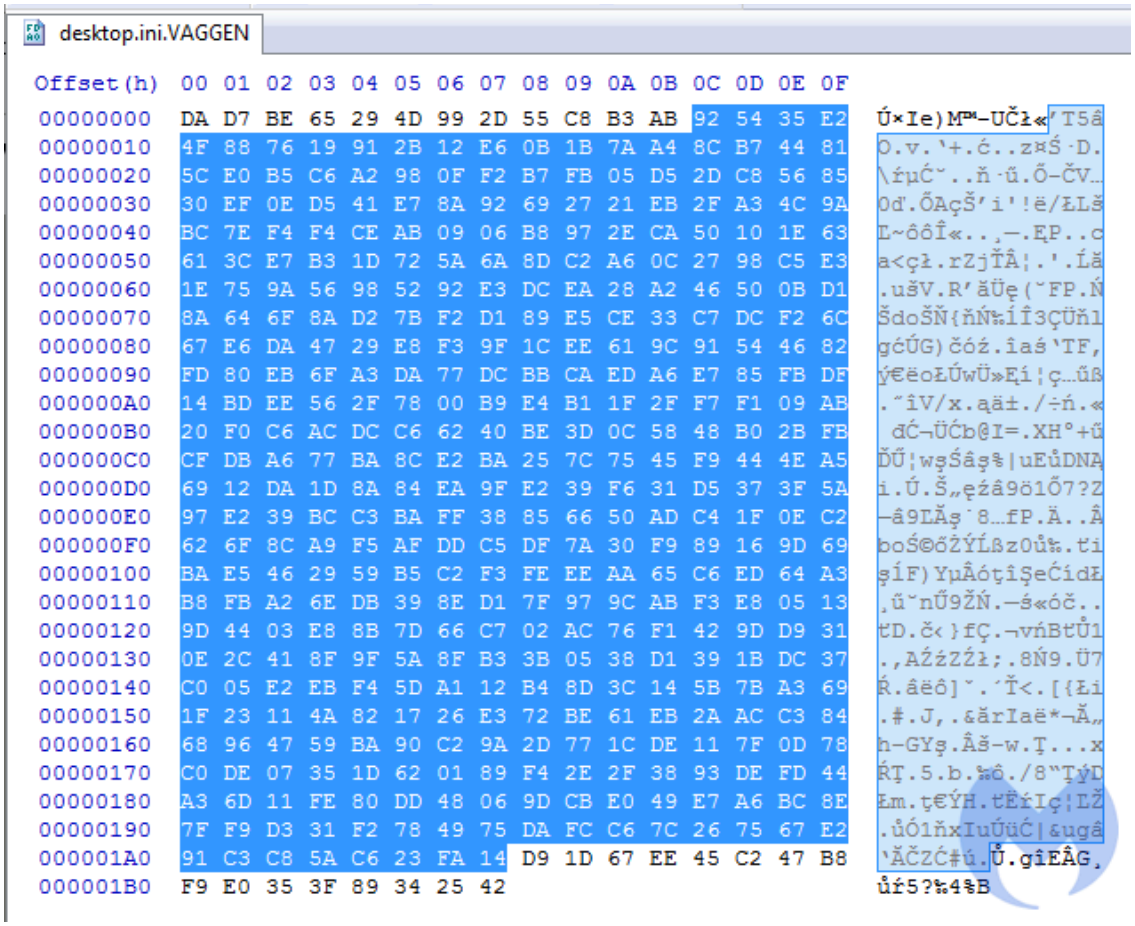
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

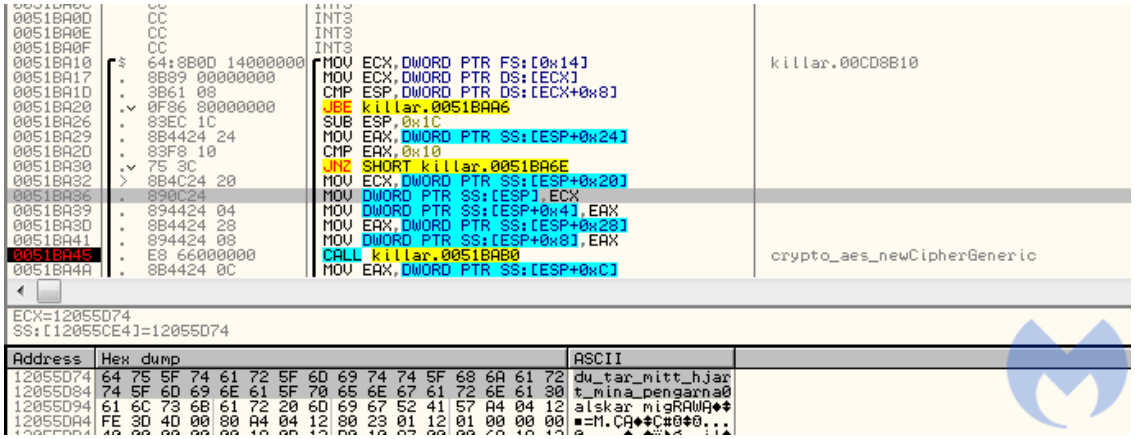


The content of the output file (with .VAGGEN extension) contains:

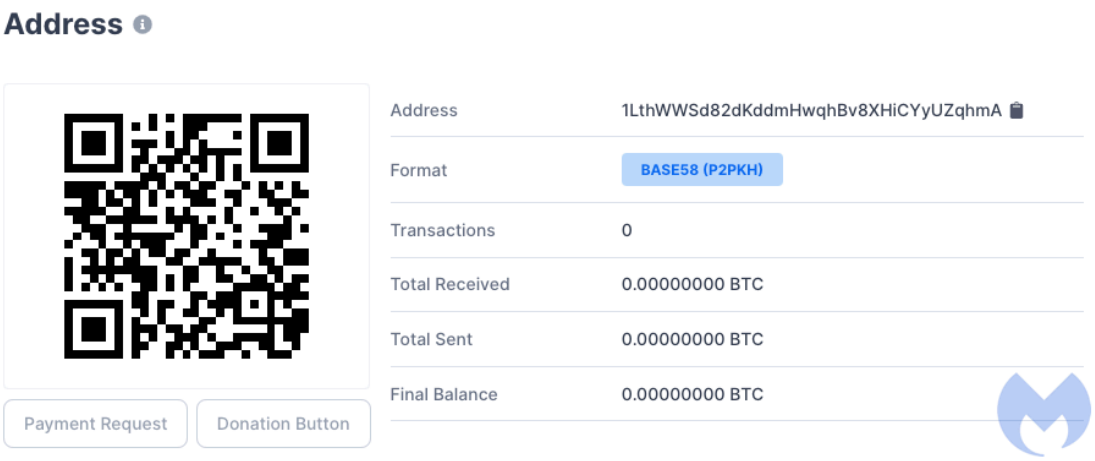
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

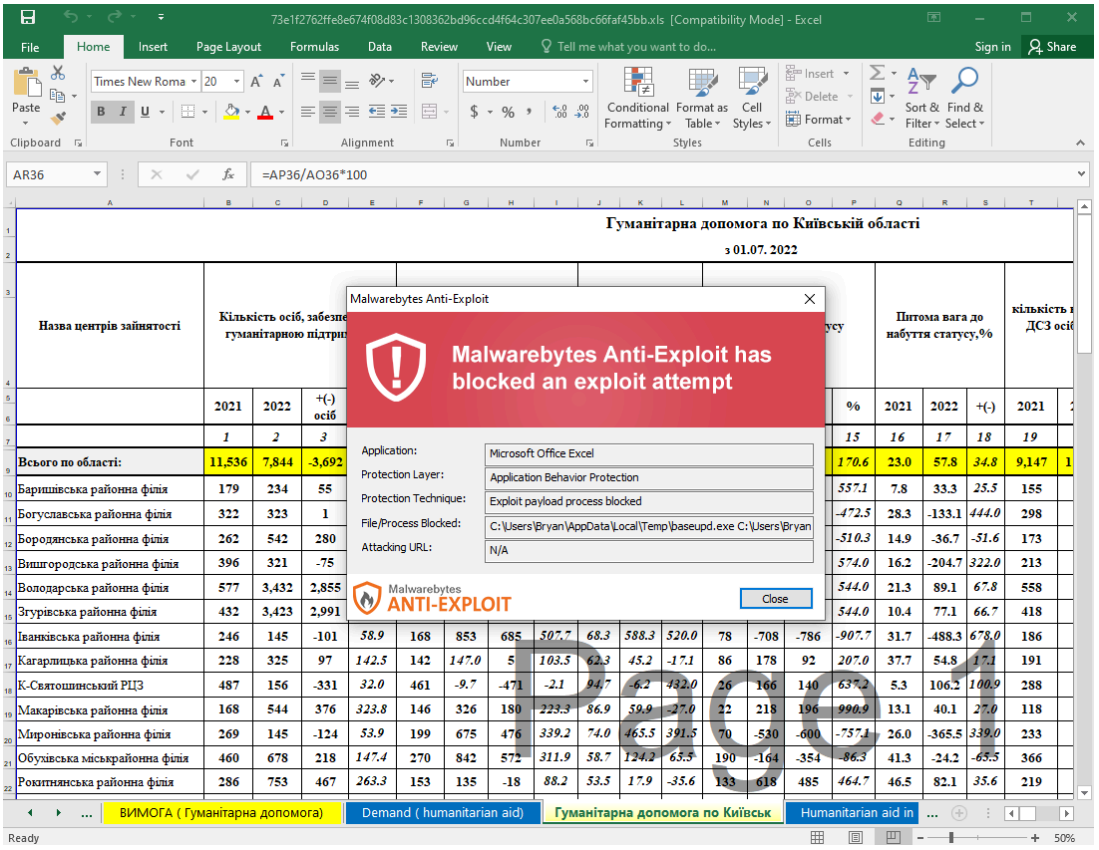
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
46  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
47    runtime_morestack_noctxt();
48  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
49  ((void (*)(void))loc_45CDA0)();
50  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
51  v38 = v4;
52  v30 = v5;
53  v31 = v6;
54  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
55  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
56  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
57  v32 = v10;
58  v11 = *(void (__cdecl **)(int))(v10 + 16);
59  v39 = v12;
60  v11(v12);
61  v29 = v13;
62  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
63  v40 = v15;
64  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
65  v17 = v27;
66  if ( v27 )
67  {
68    if ( v27 )
69      v17 = *(_DWORD *)(v27 + 4);
70    v43 = v17;
71    v44 = v28;
72    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
73  }
```

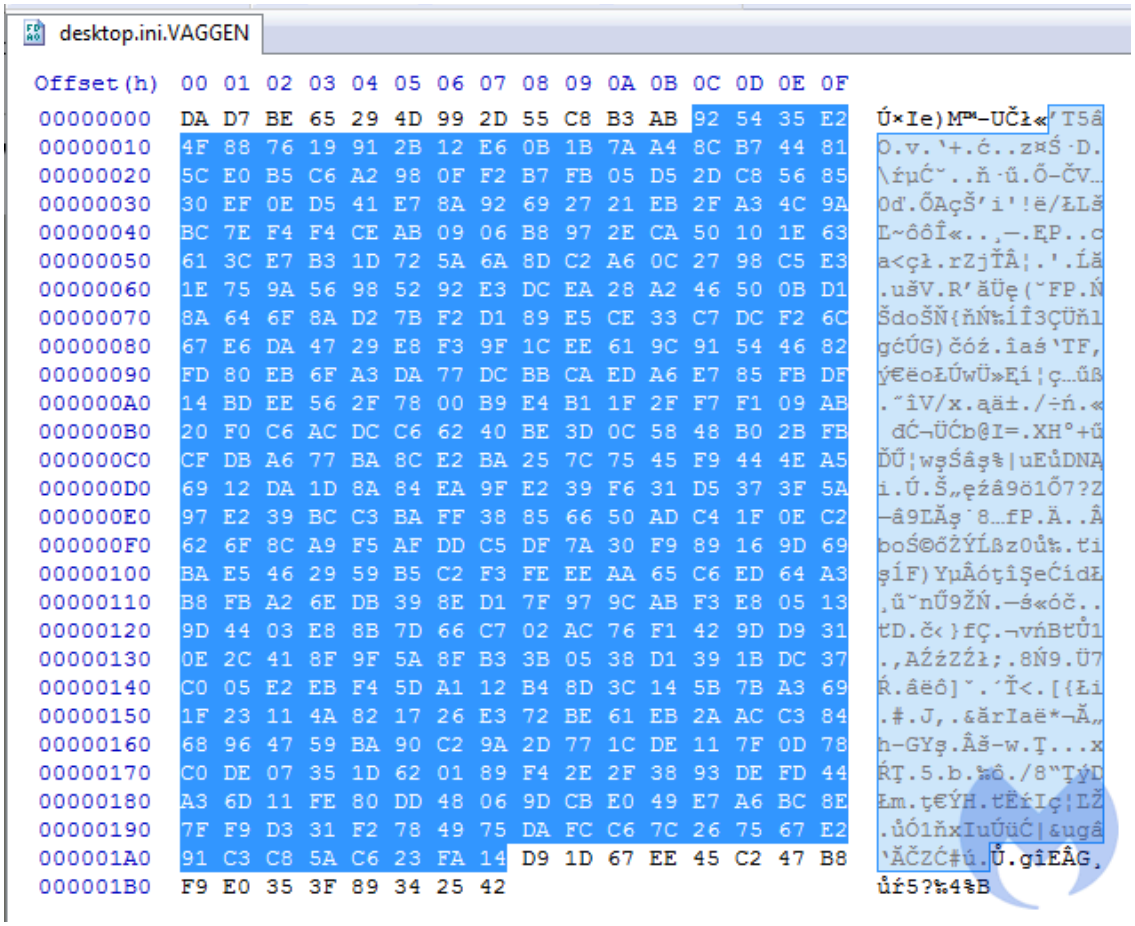


The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

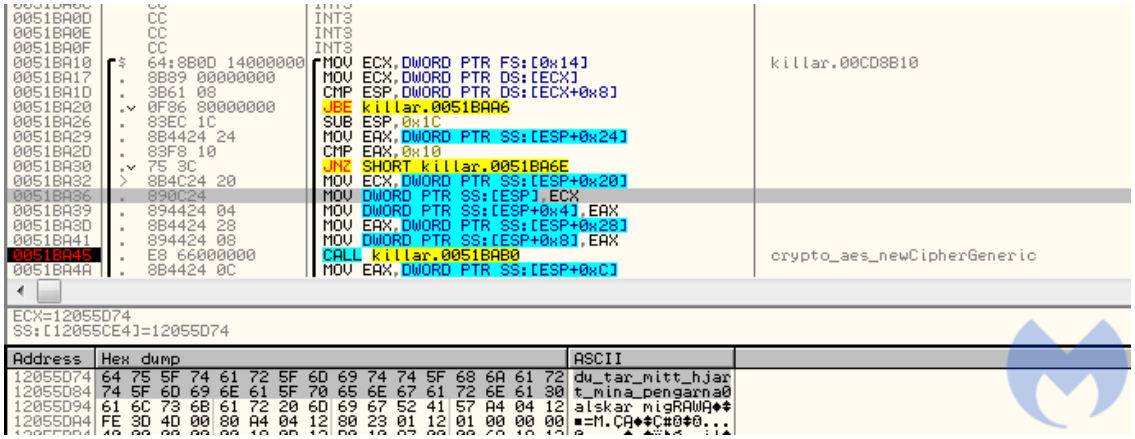


The content of the output file (with .VAGGEN extension) contains:

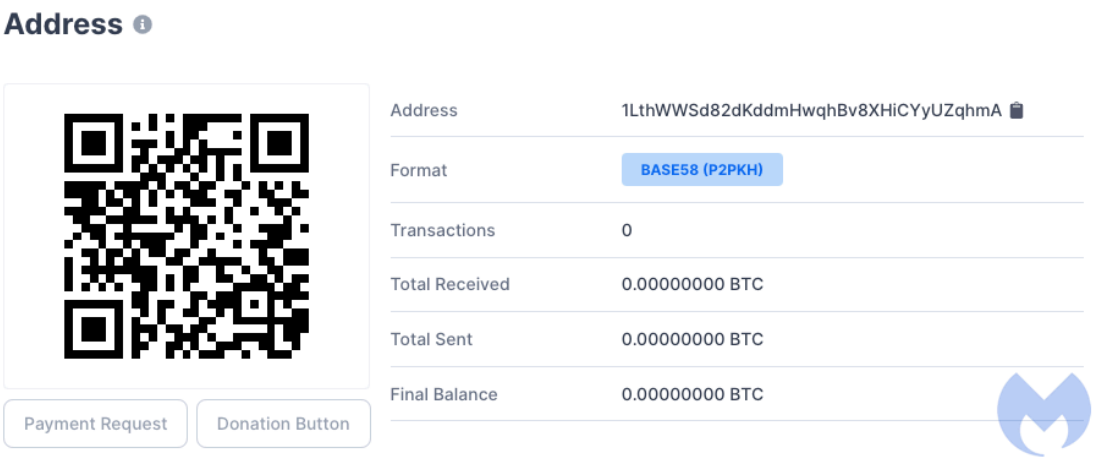
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

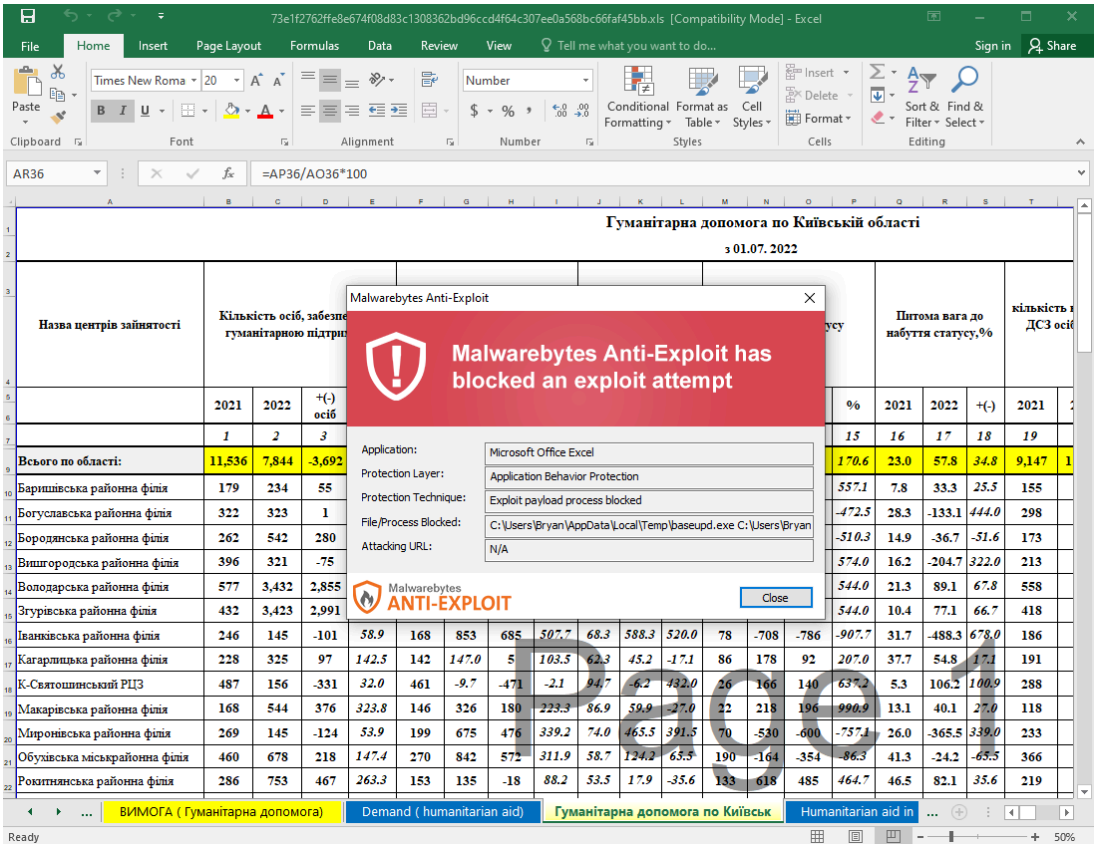
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

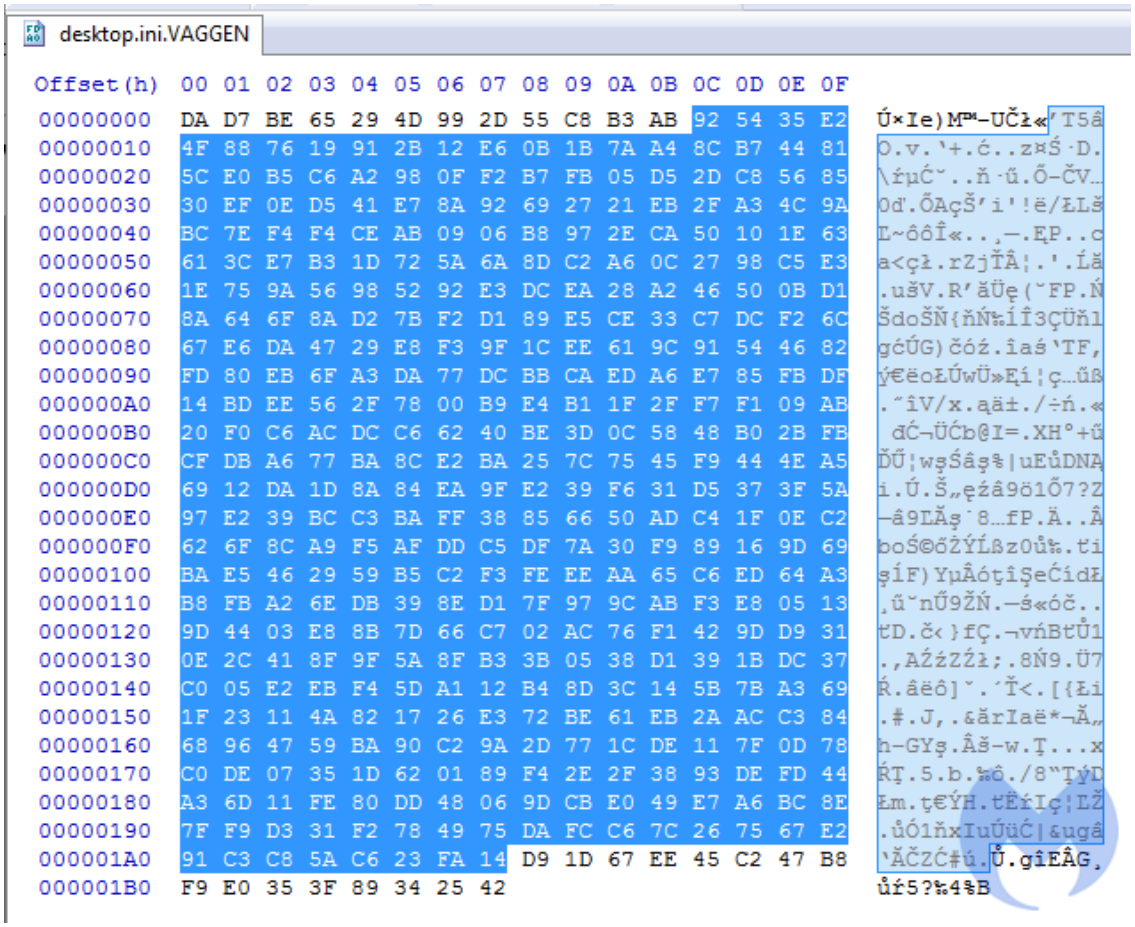
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

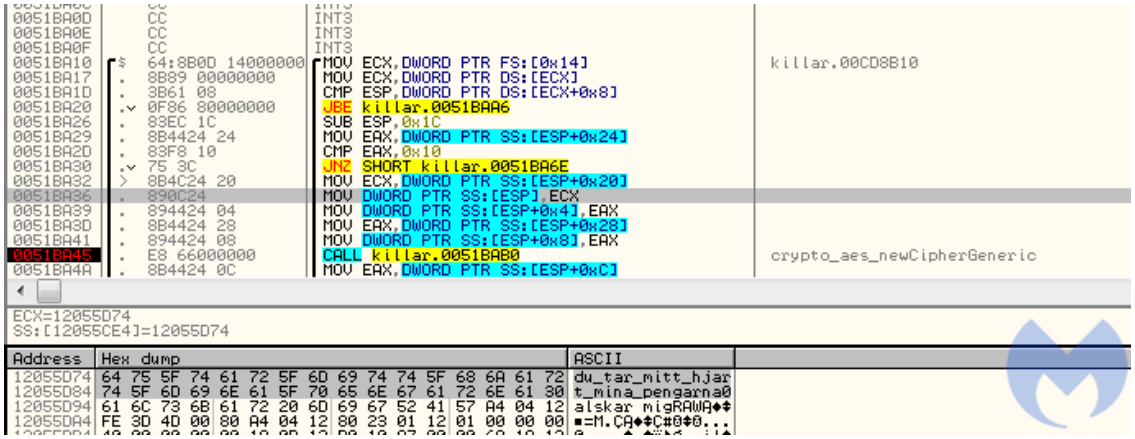


The content of the output file (with .VAGGEN extension) contains:

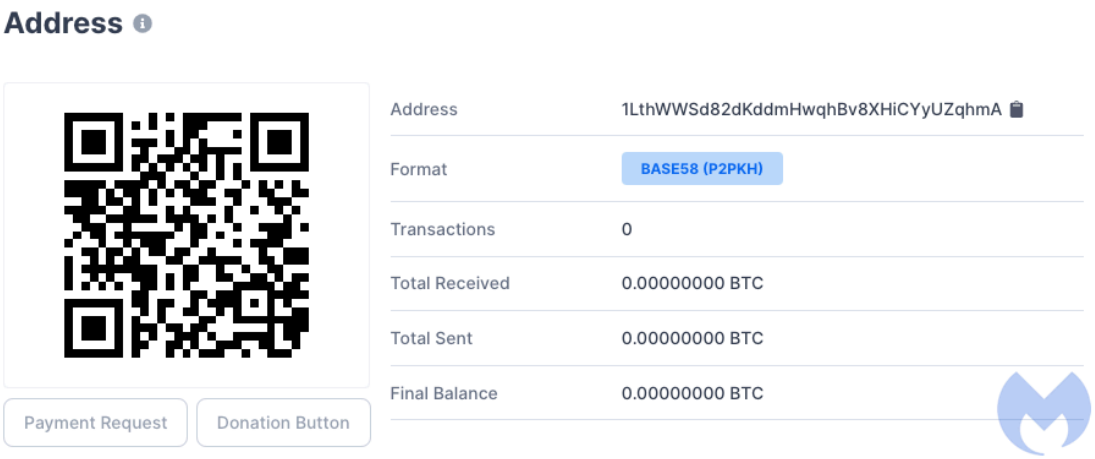
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

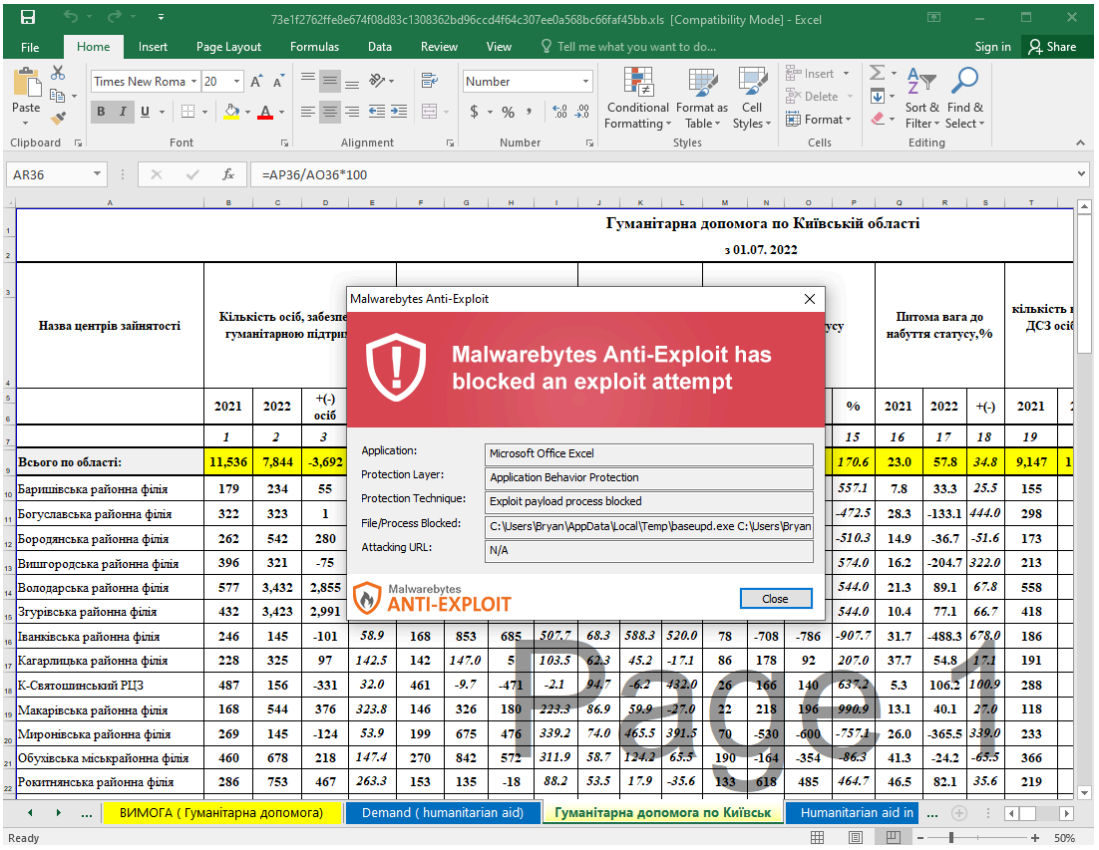
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

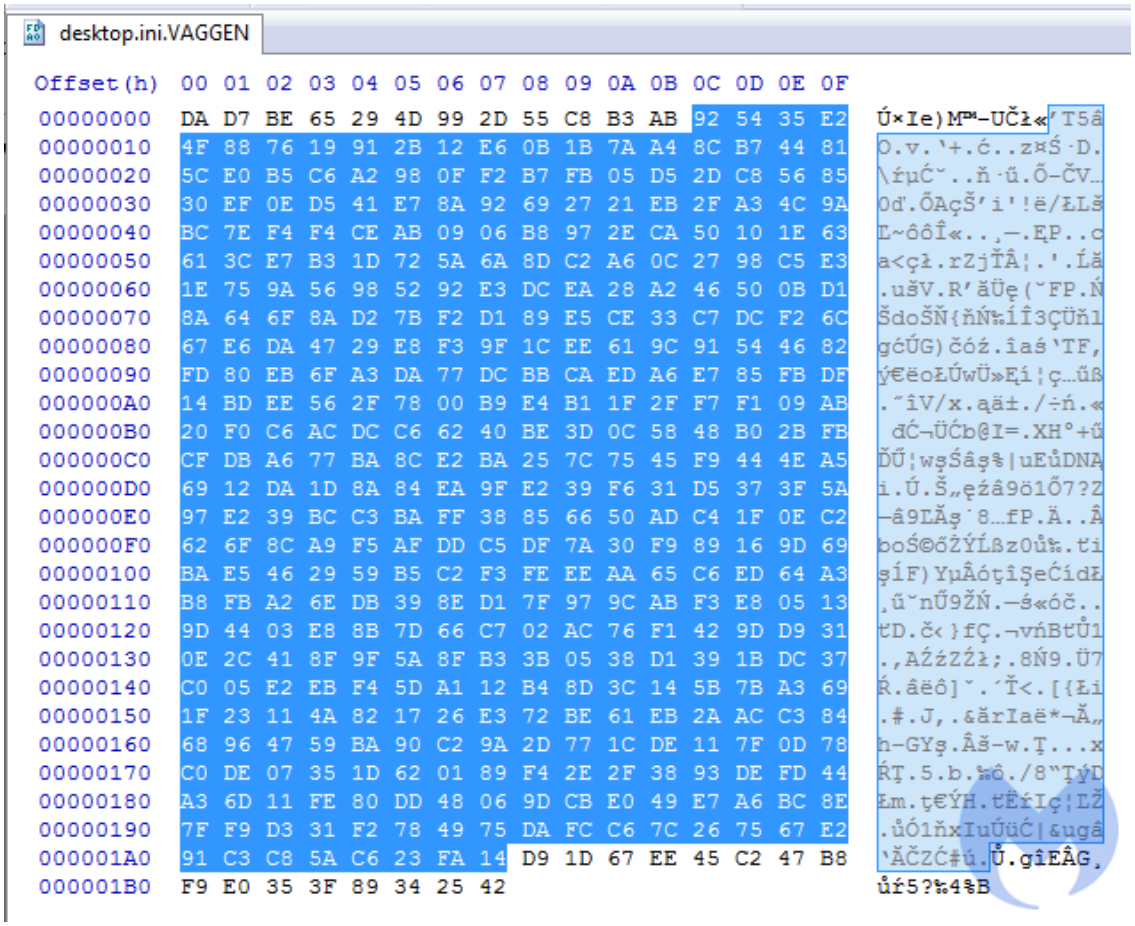
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

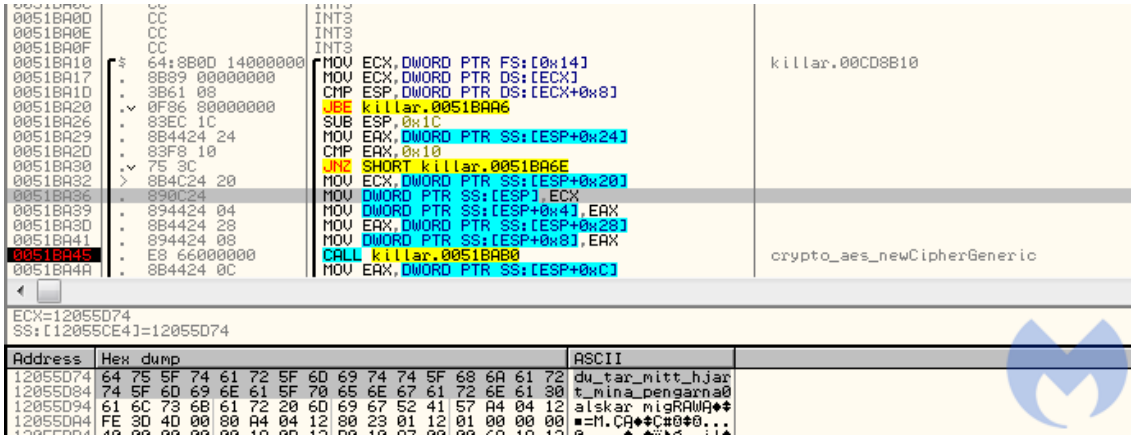


The content of the output file (with .VAGGEN extension) contains:

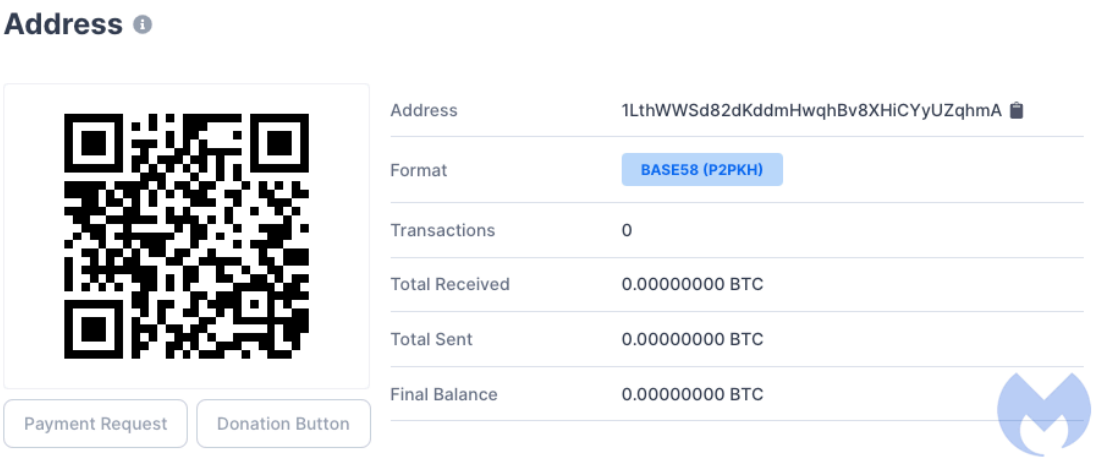
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

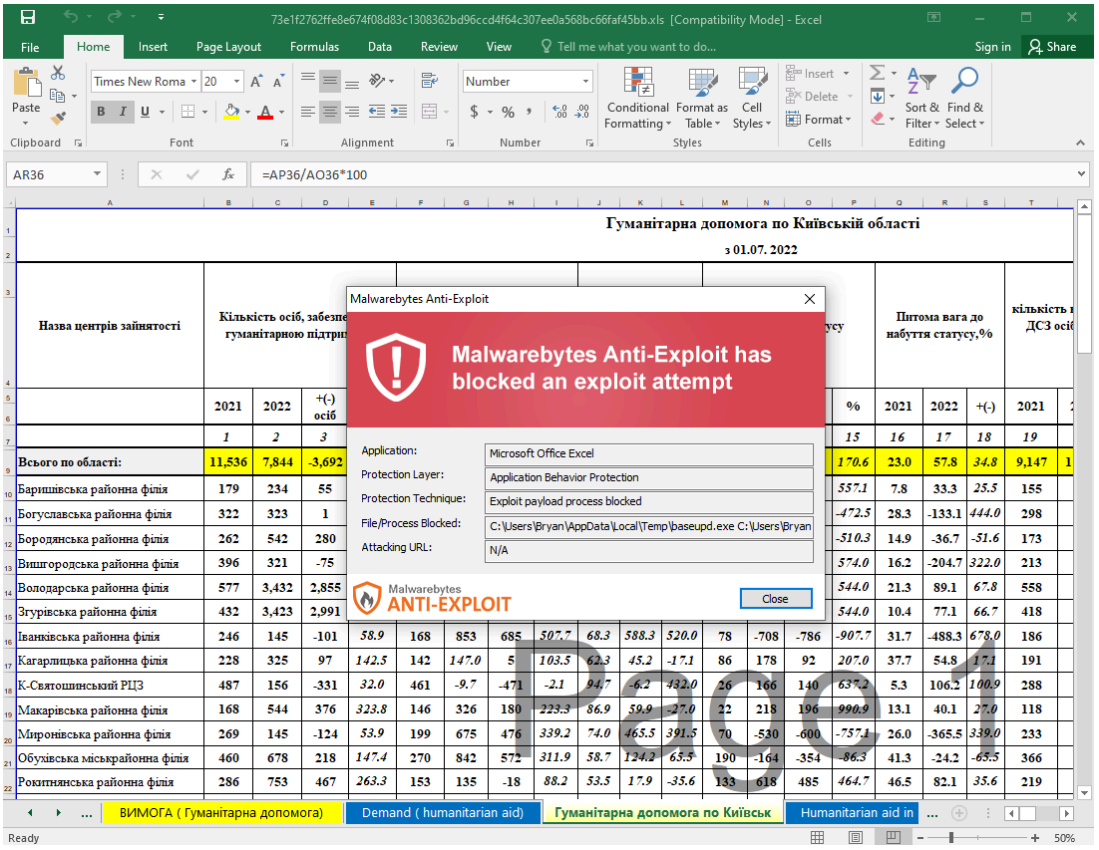
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

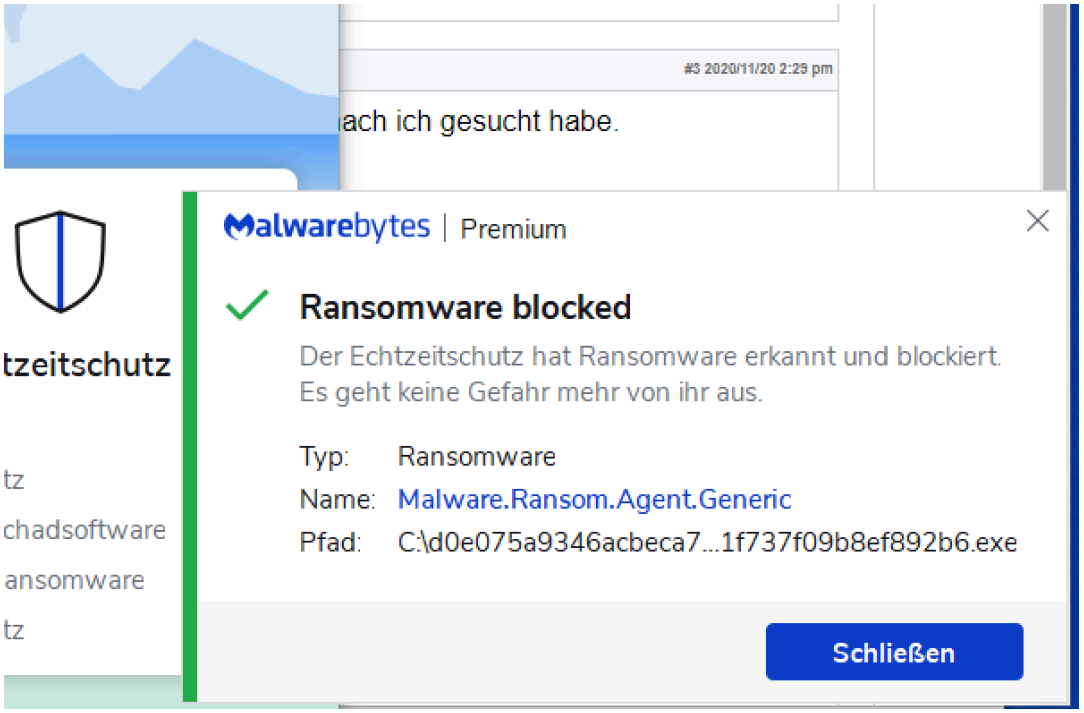
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

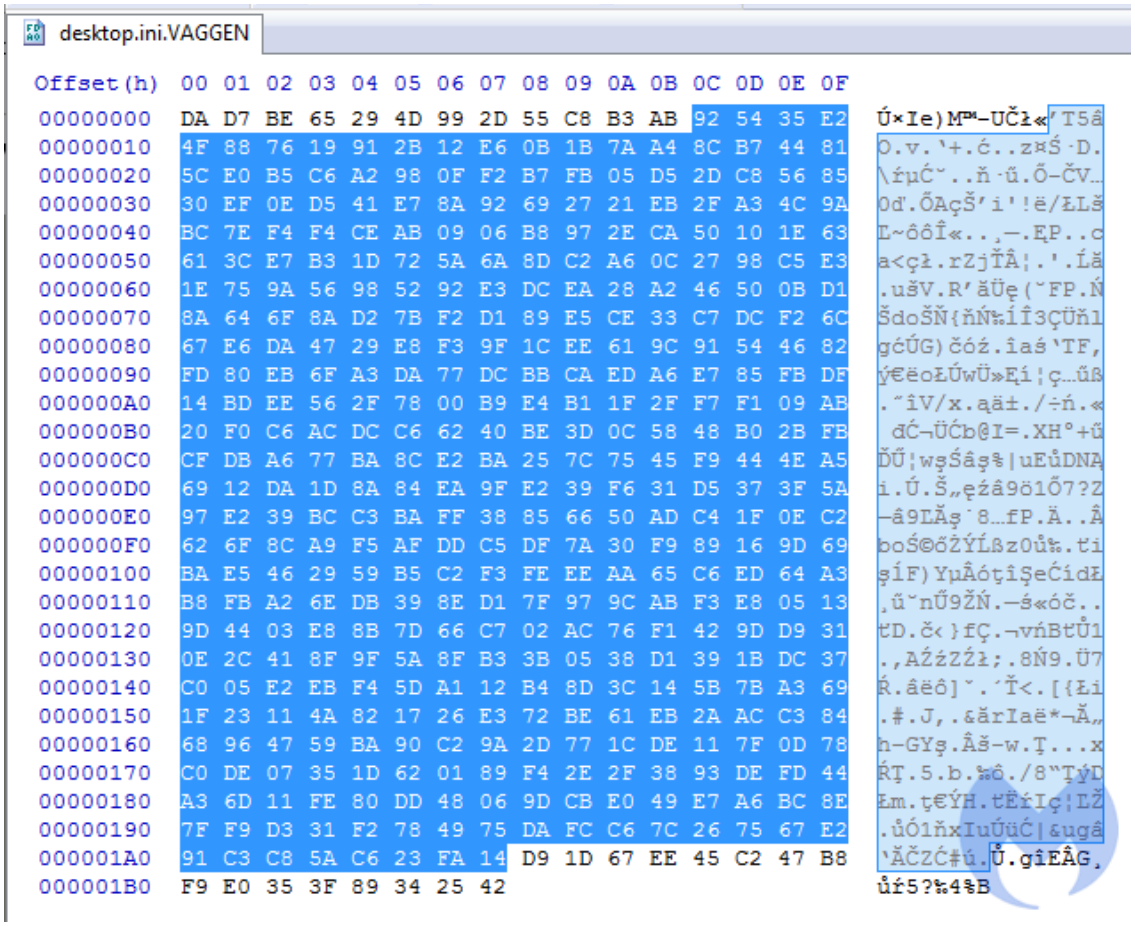
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

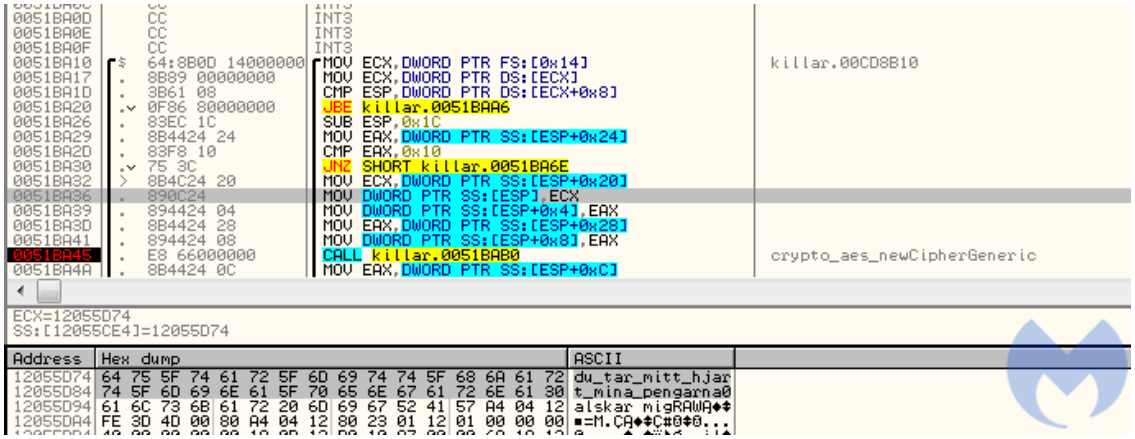


The content of the output file (with .VAGGEN extension) contains:

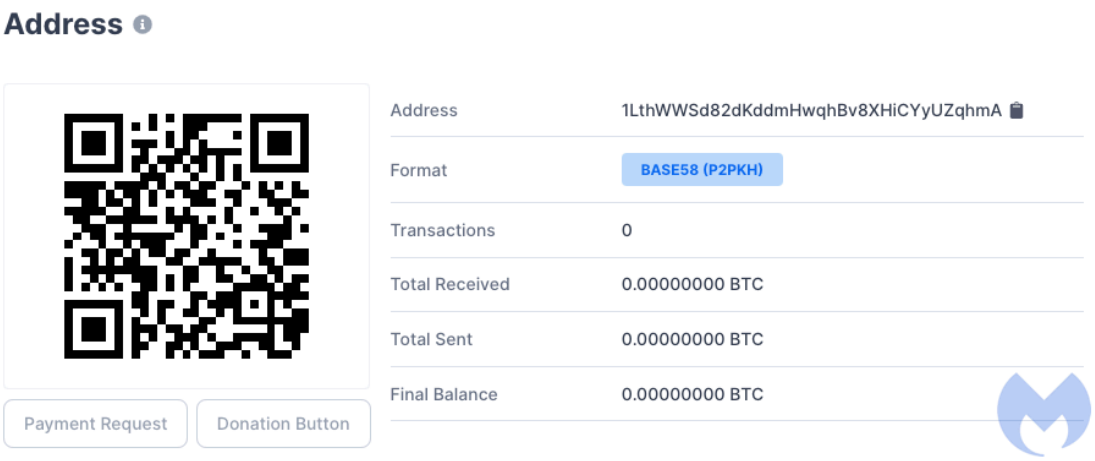
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

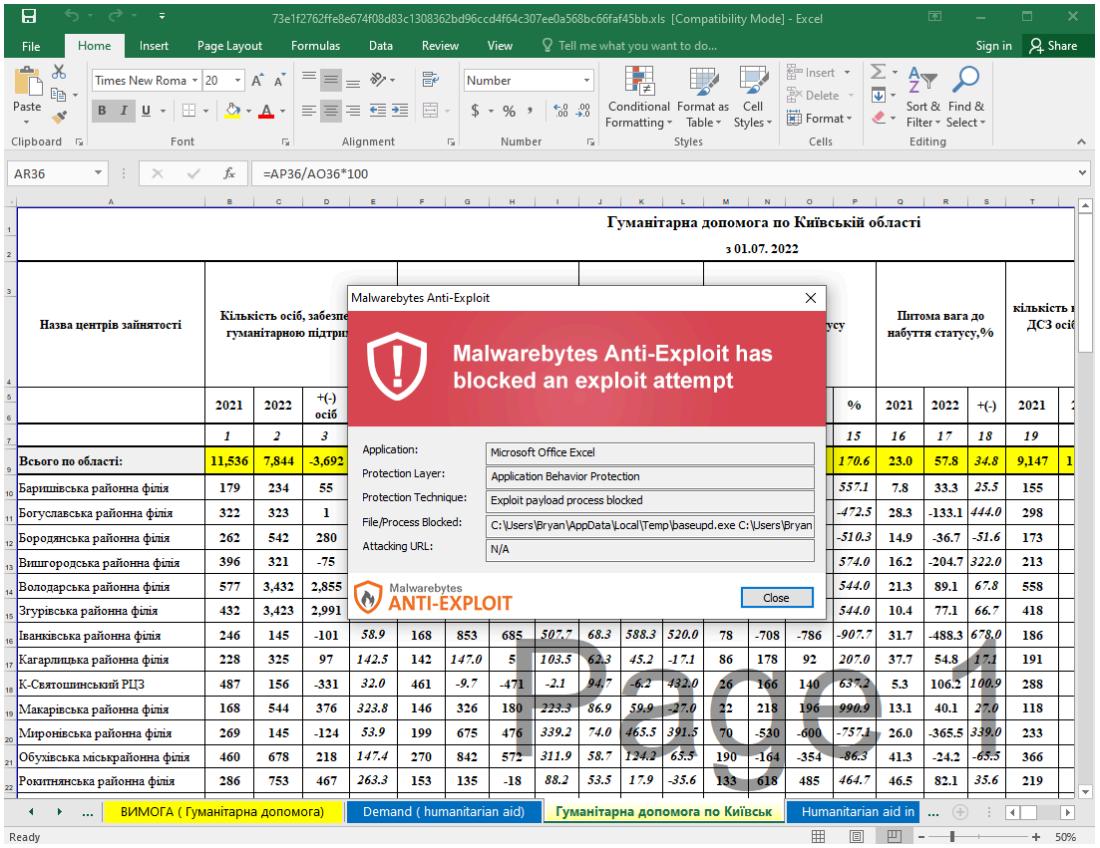
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

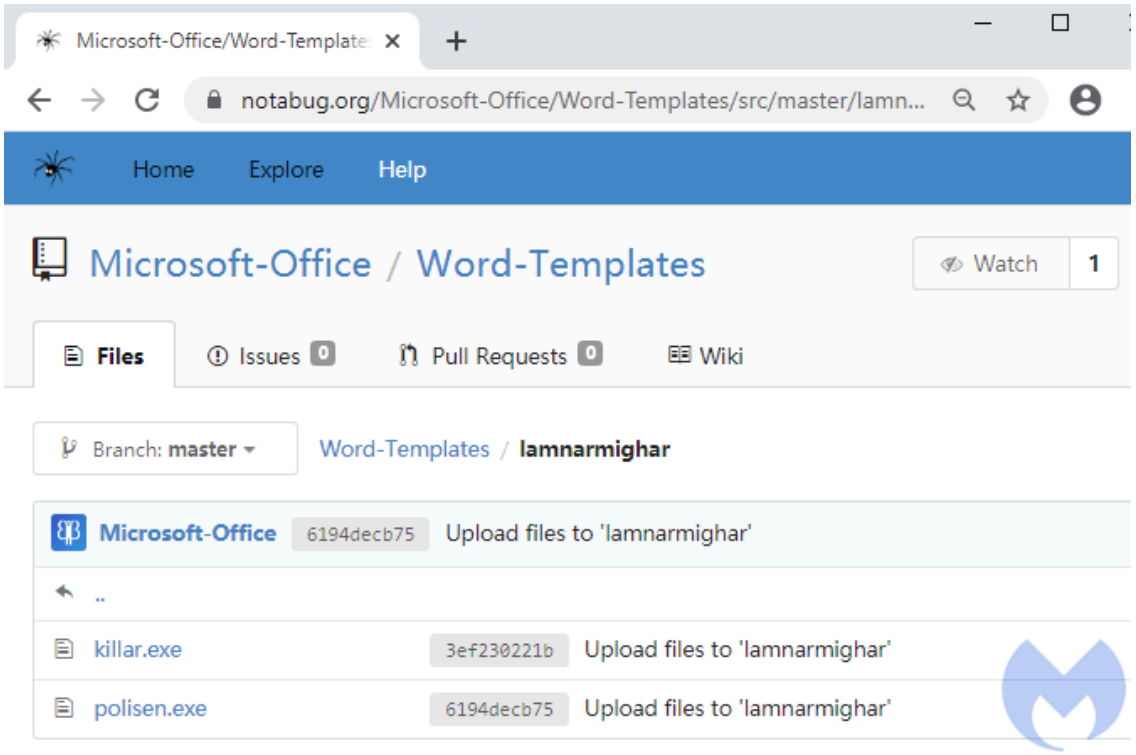
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



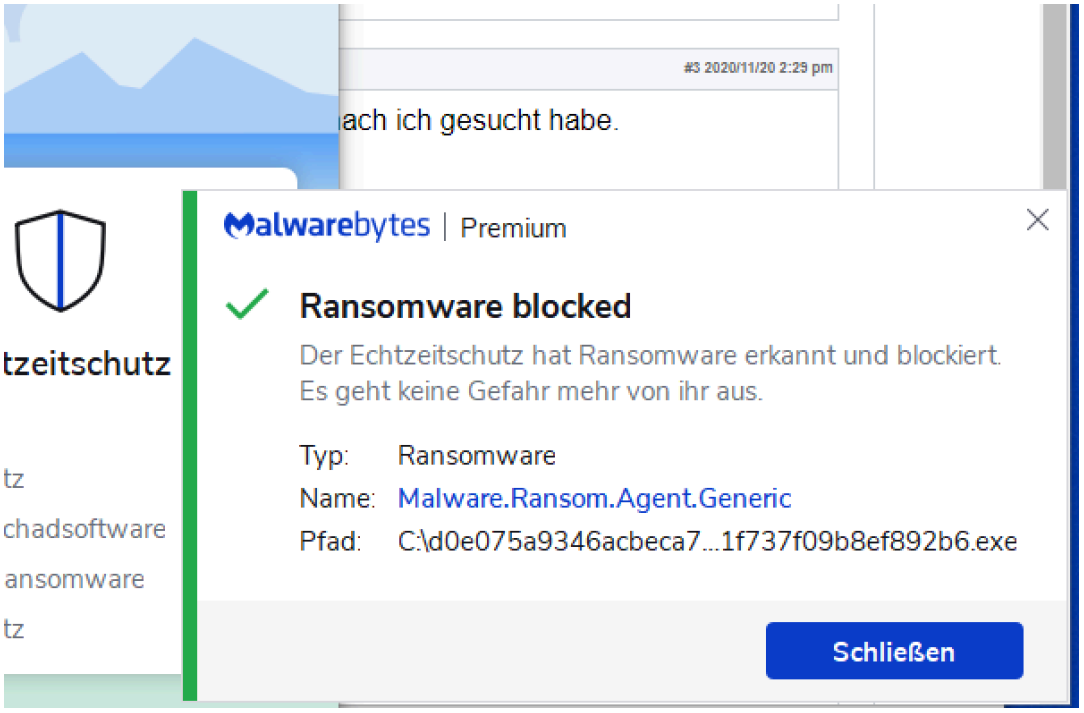
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```



Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

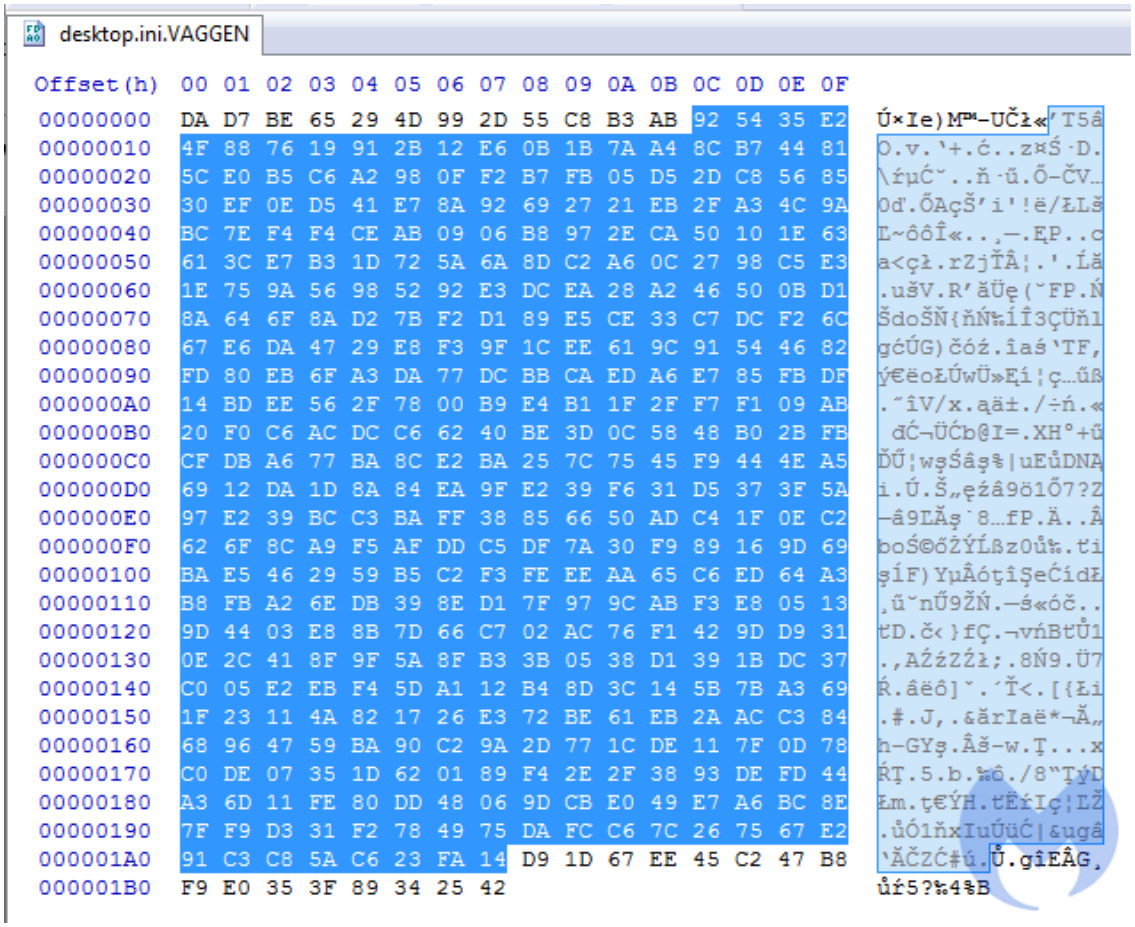


The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

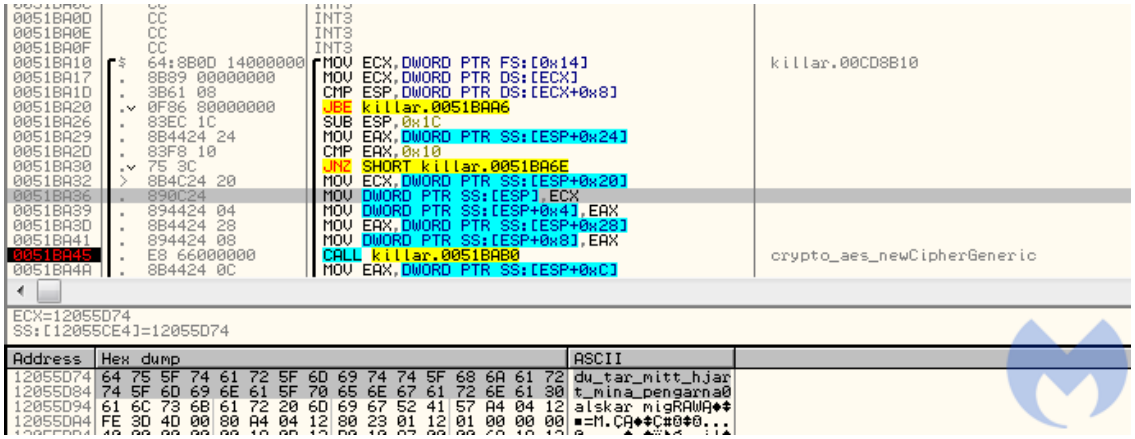


The content of the output file (with .VAGGEN extension) contains:

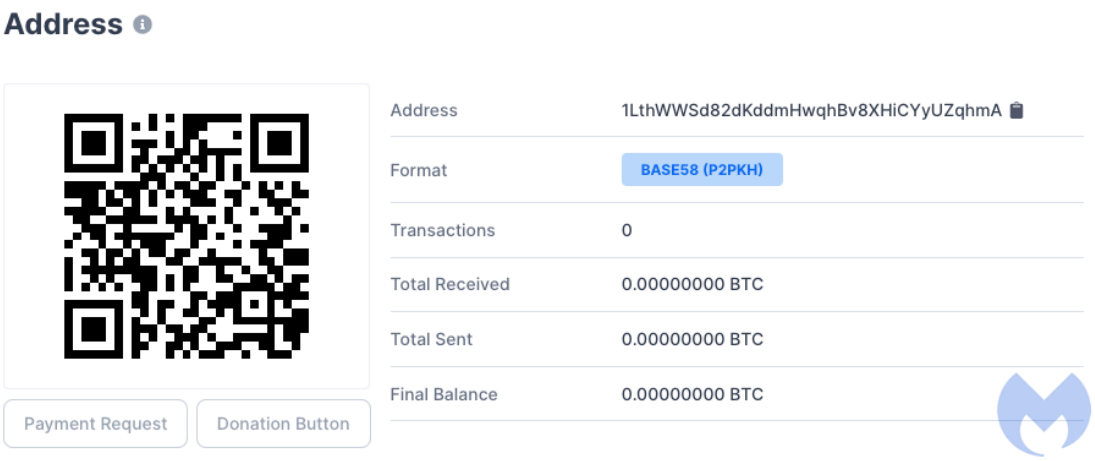
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

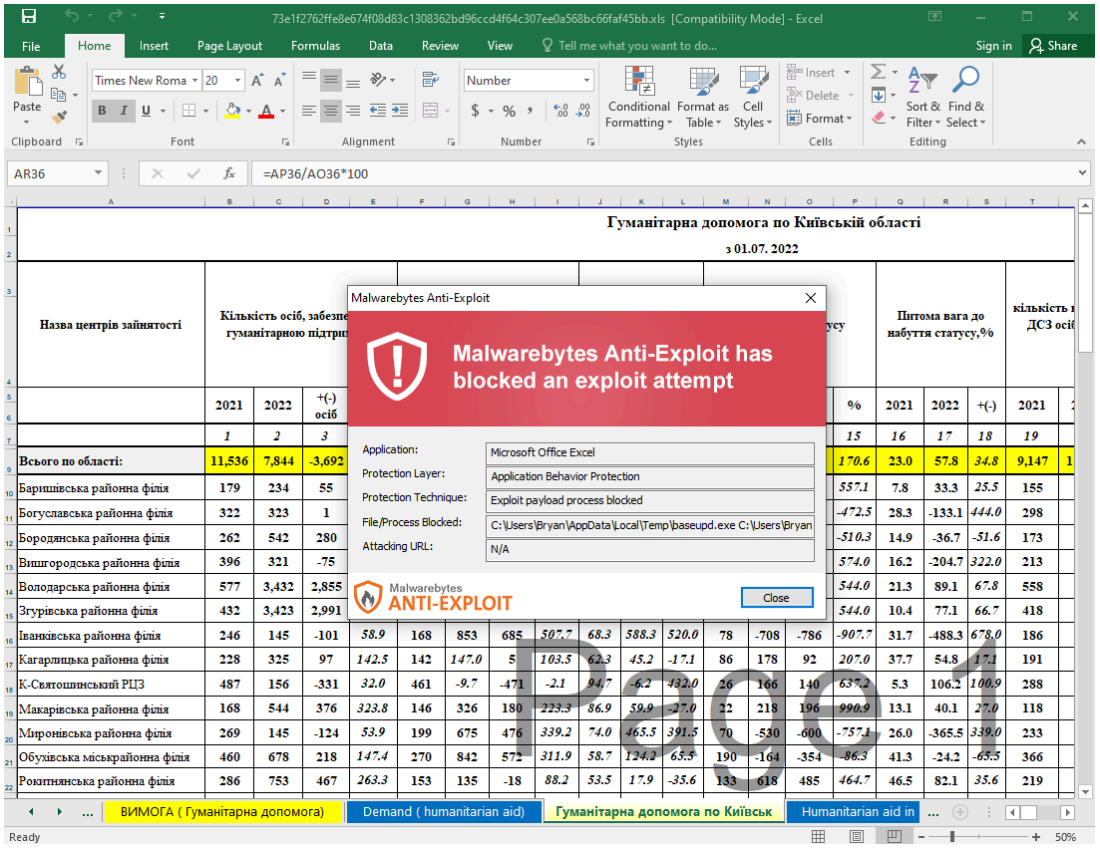
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

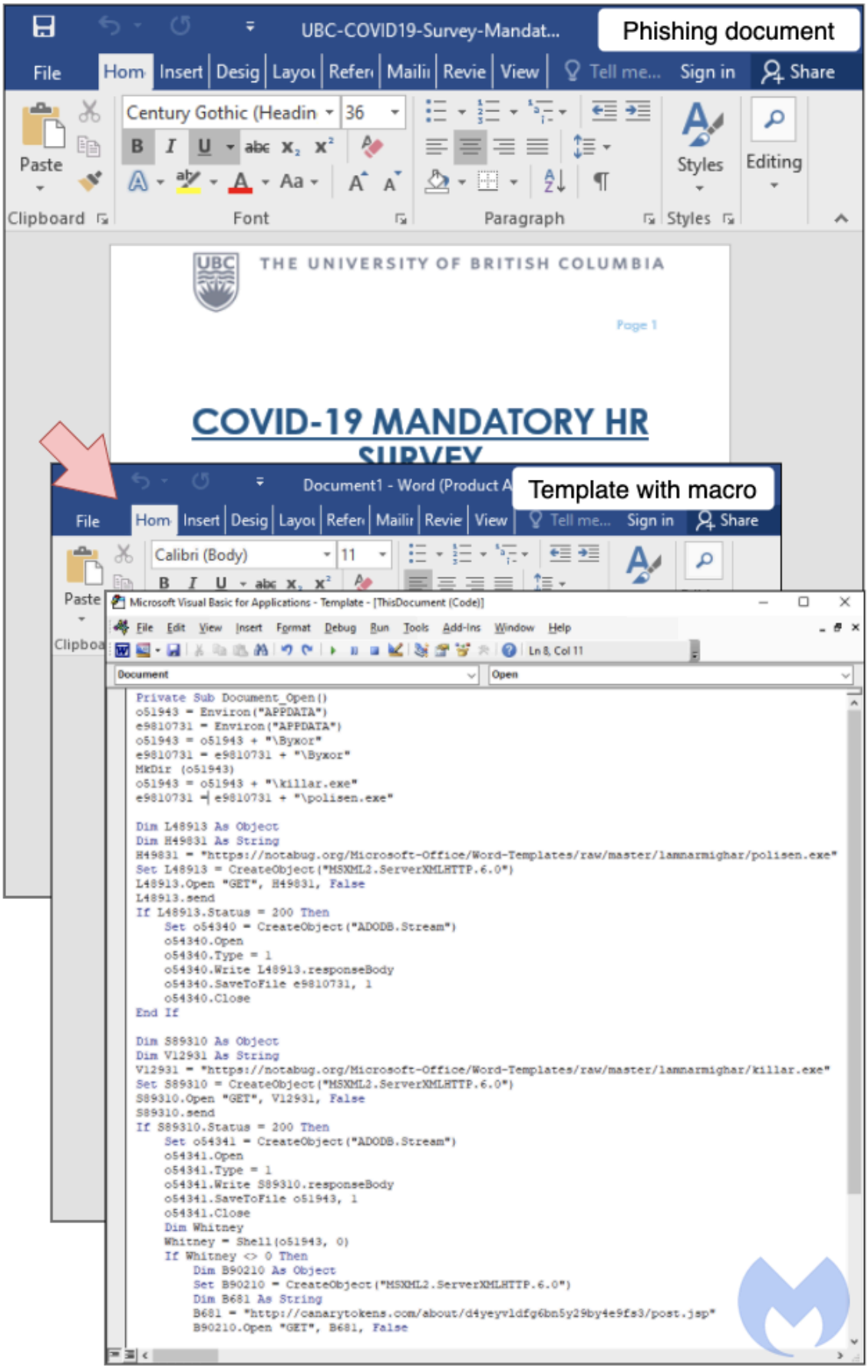
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

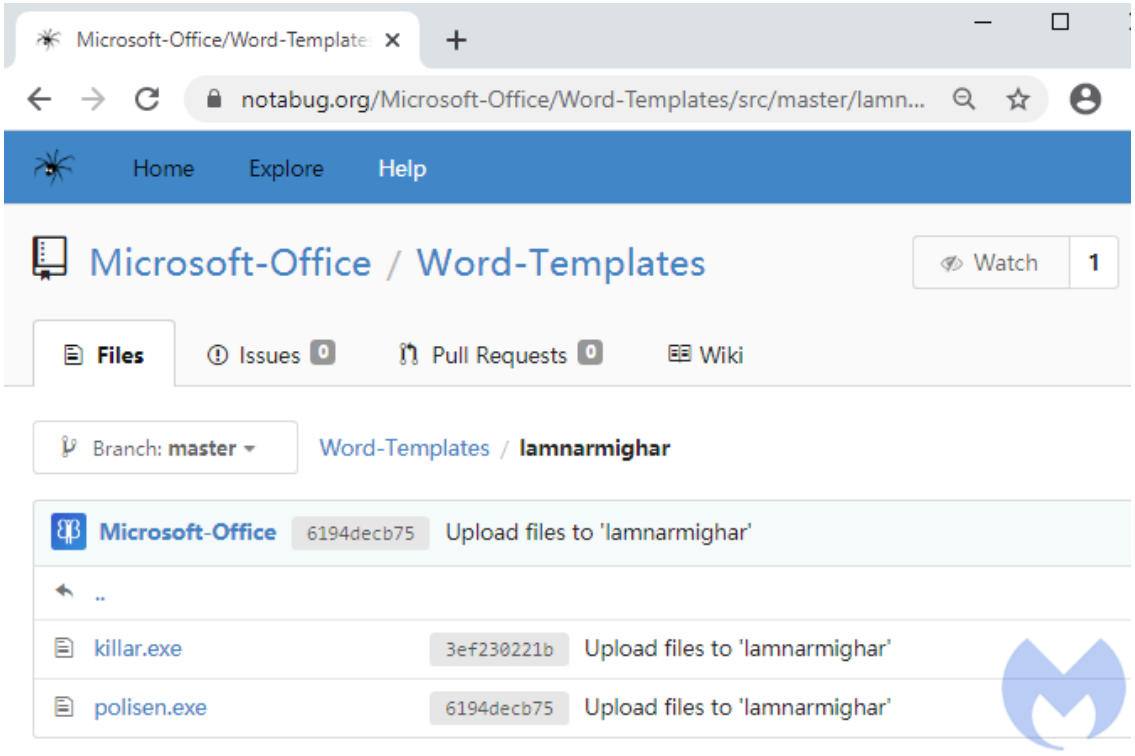
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



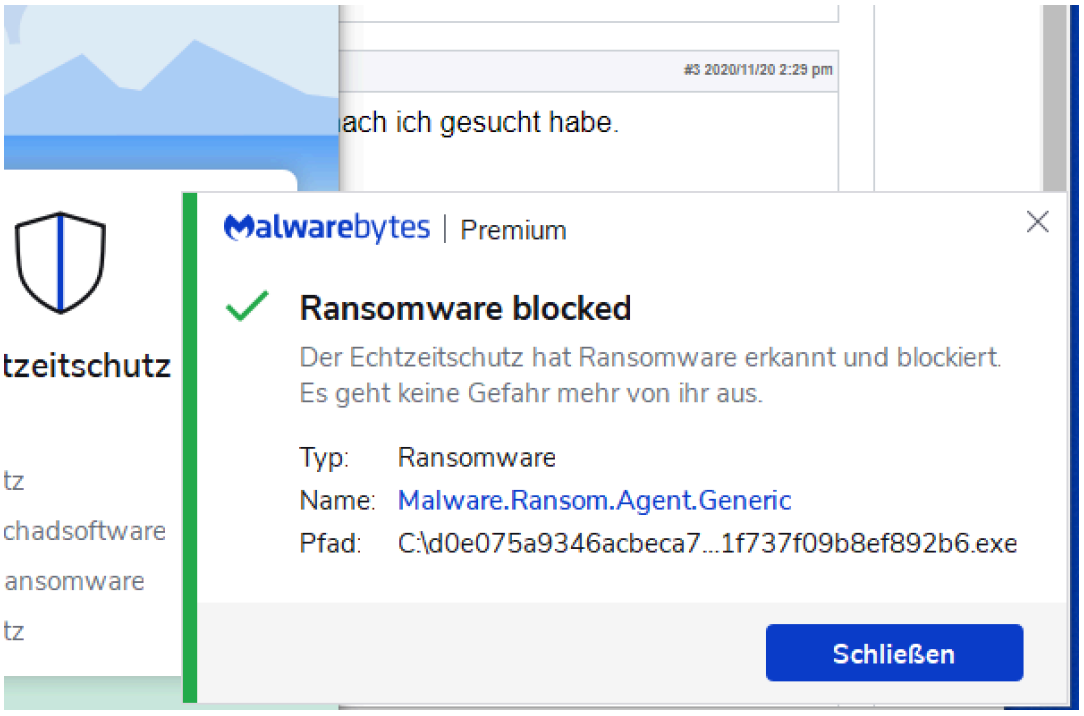
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

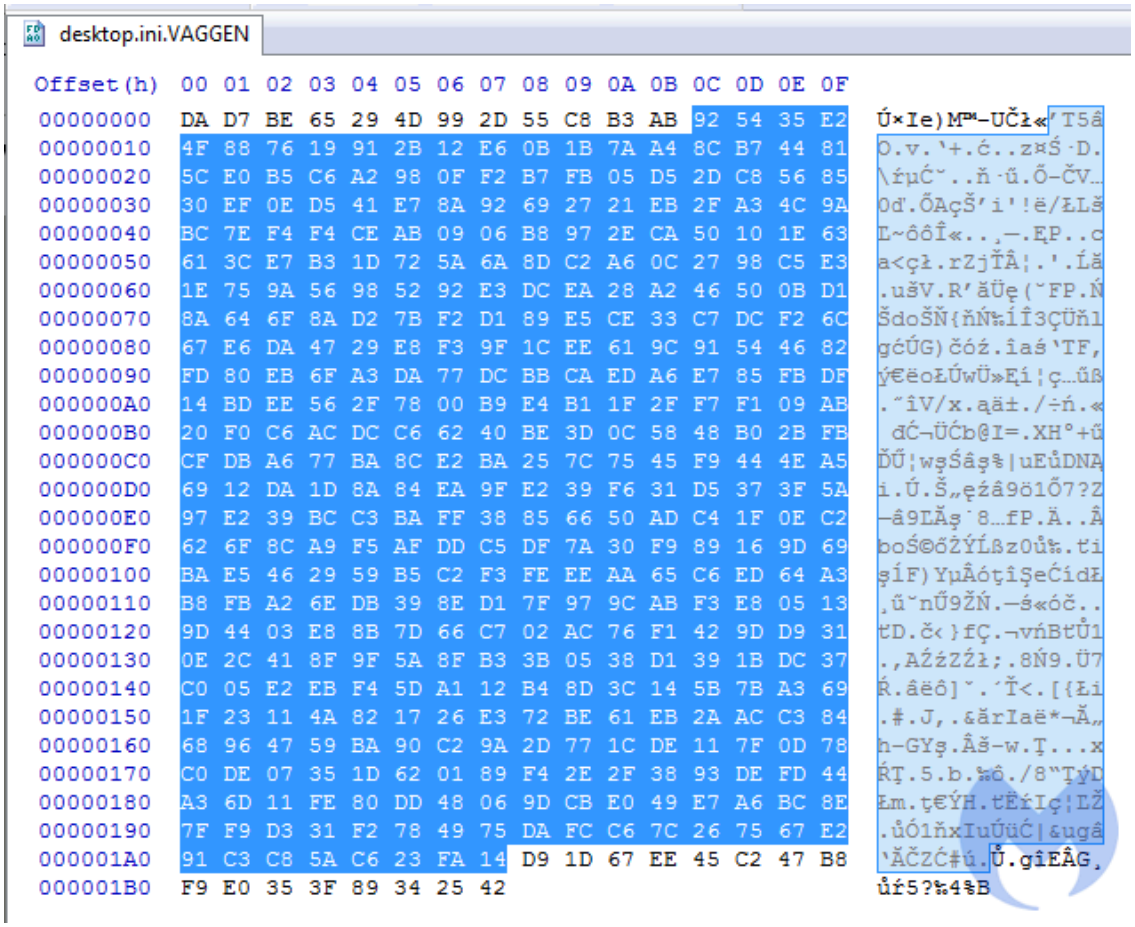
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

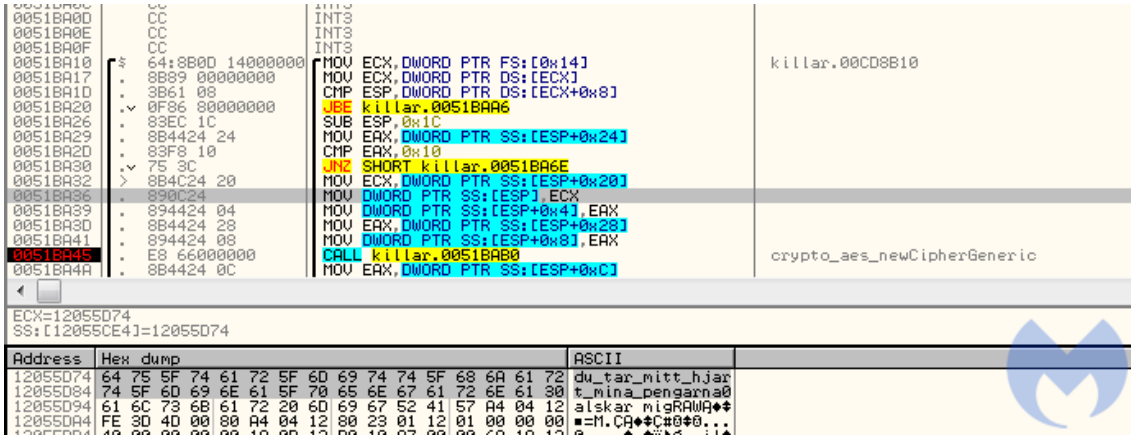


The content of the output file (with .VAGGEN extension) contains:

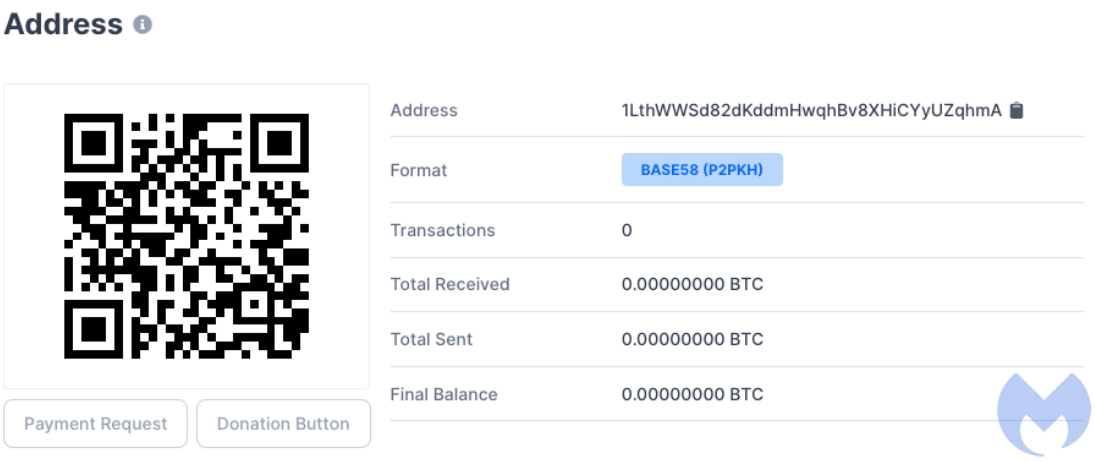
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

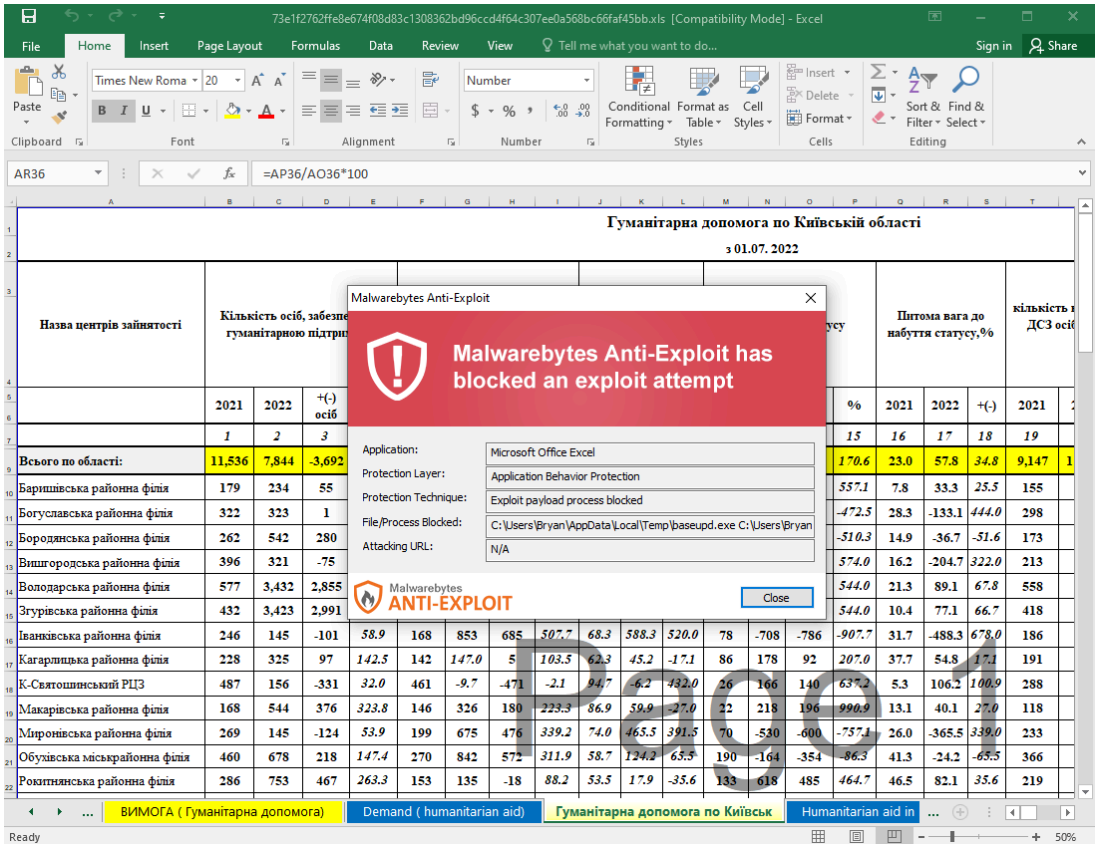
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

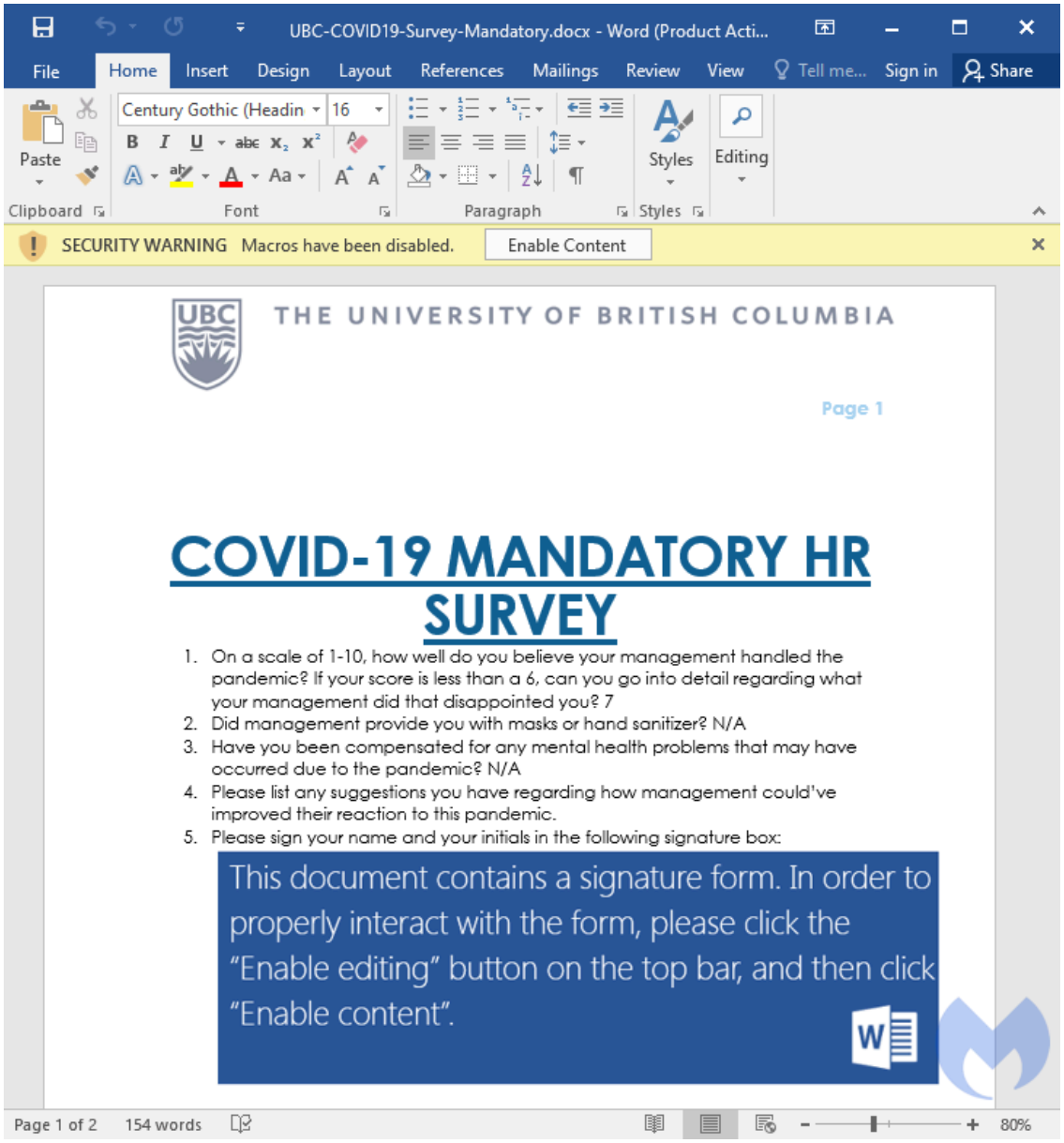
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

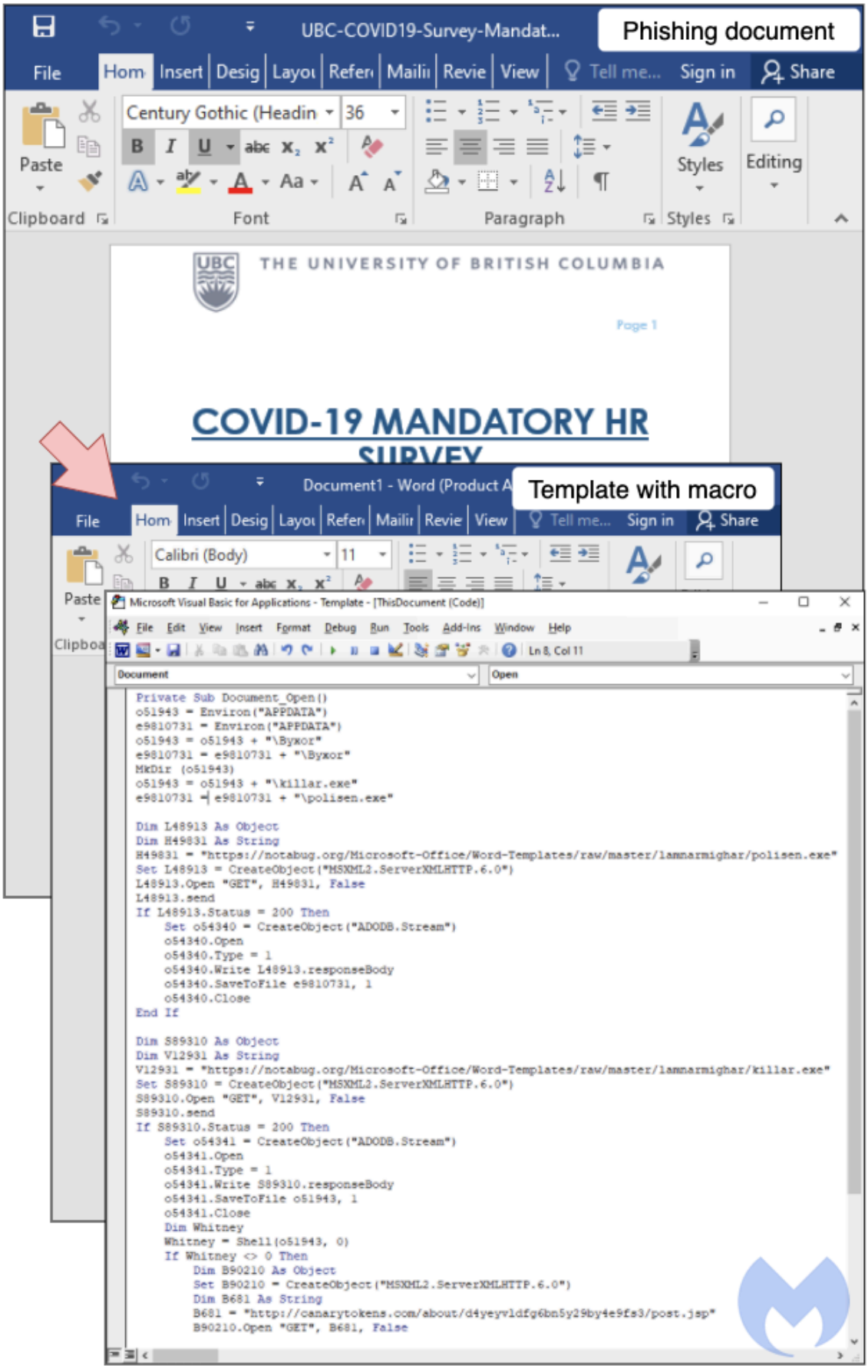
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

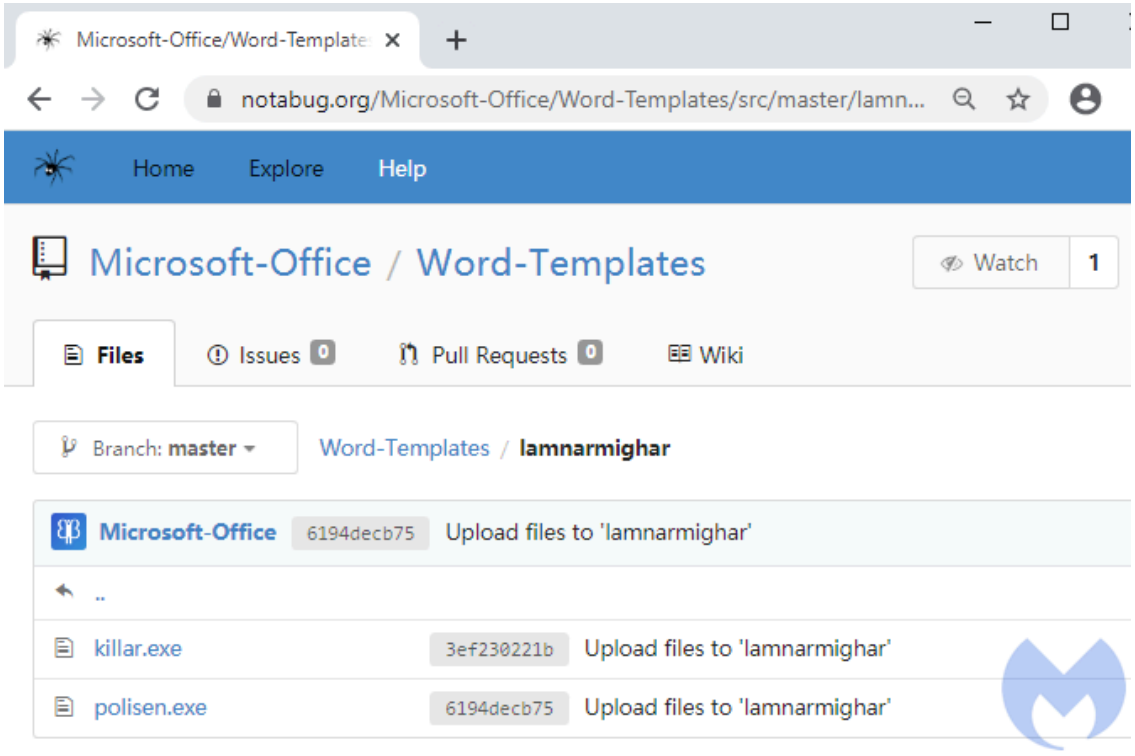
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



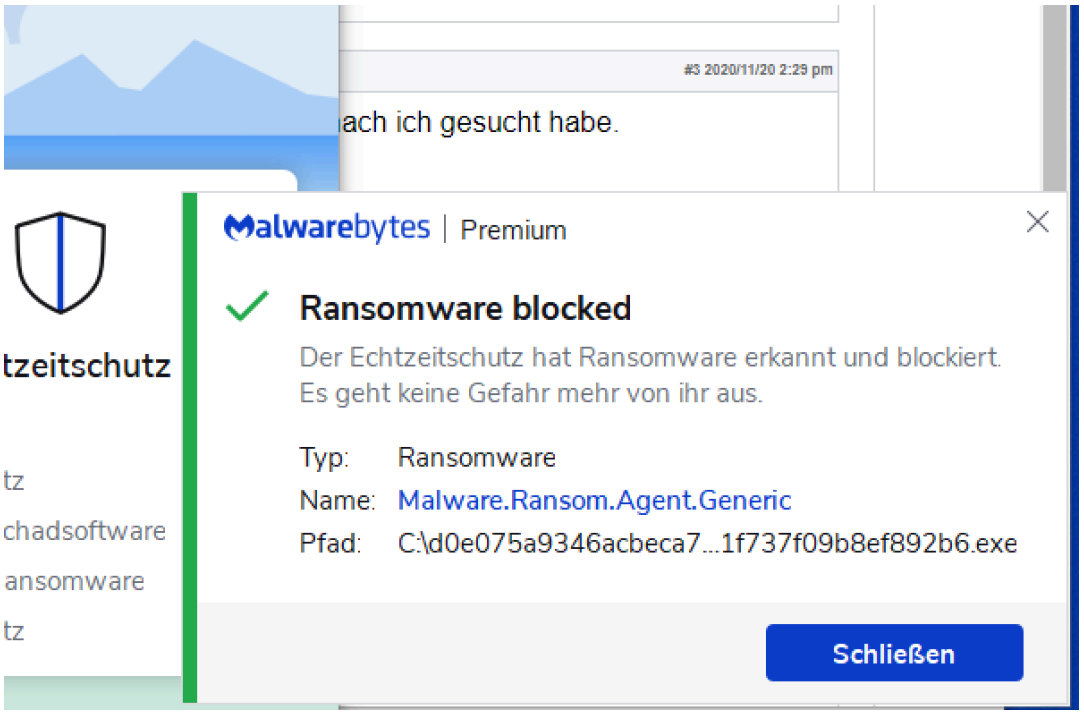
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

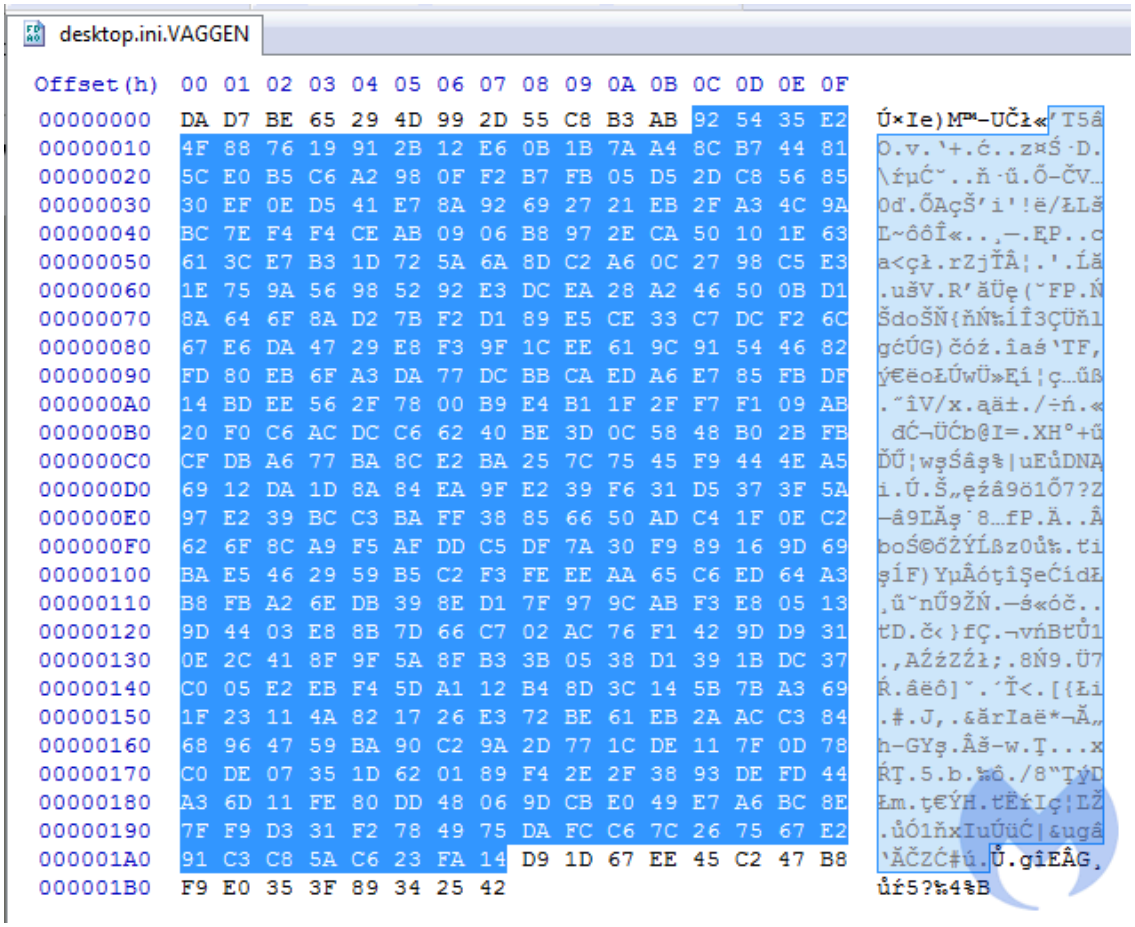
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

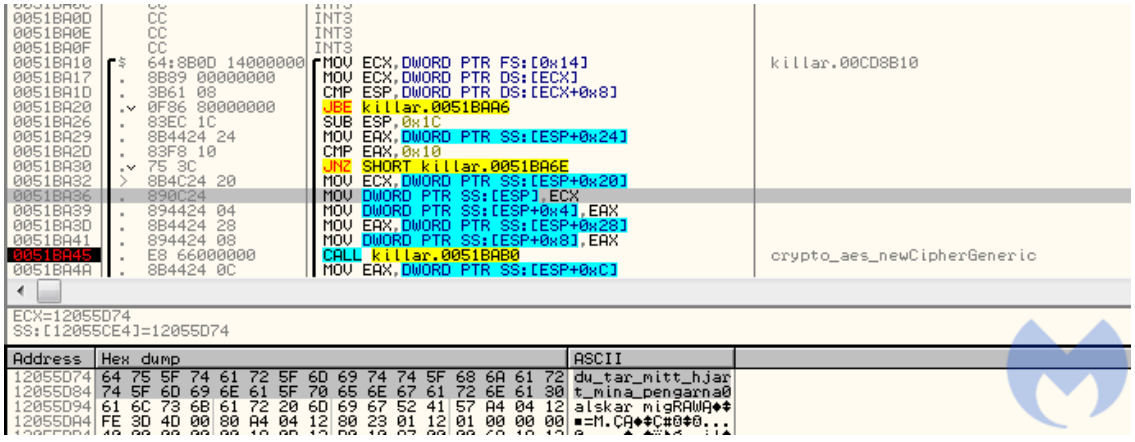


The content of the output file (with .VAGGEN extension) contains:

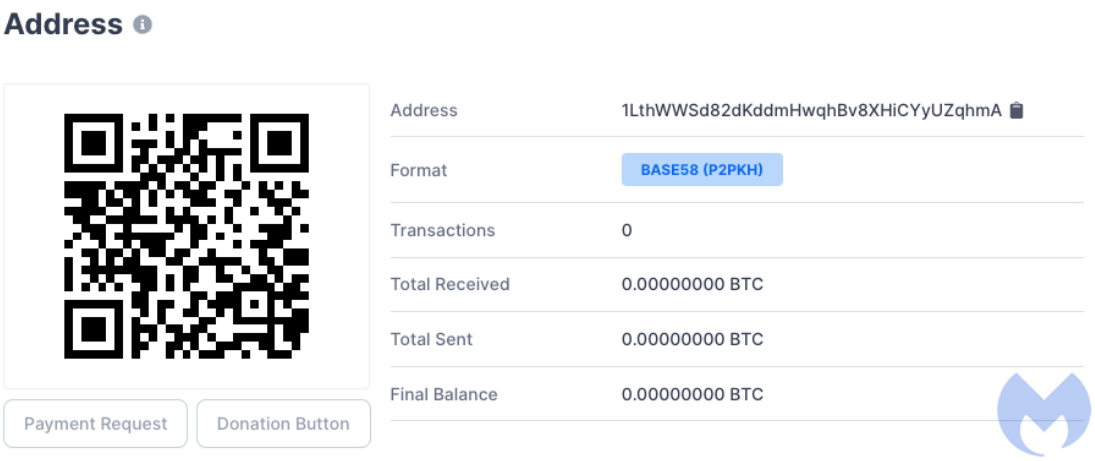
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

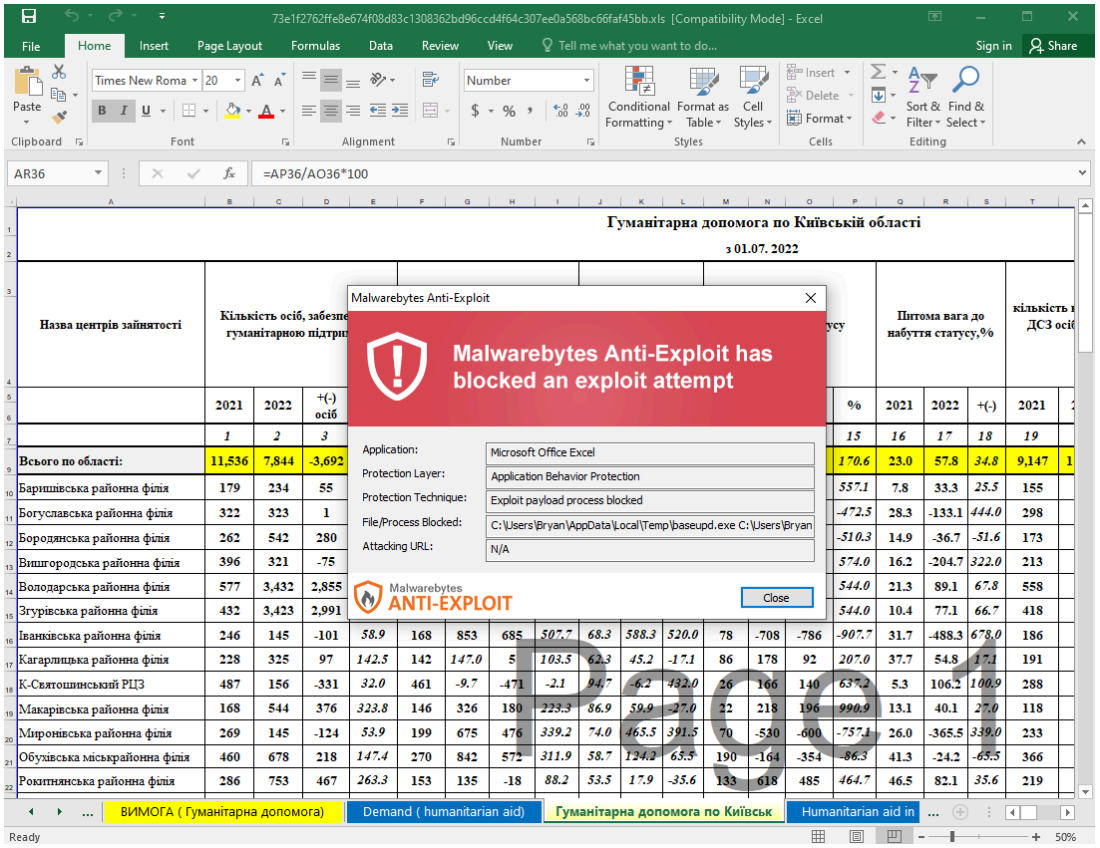
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

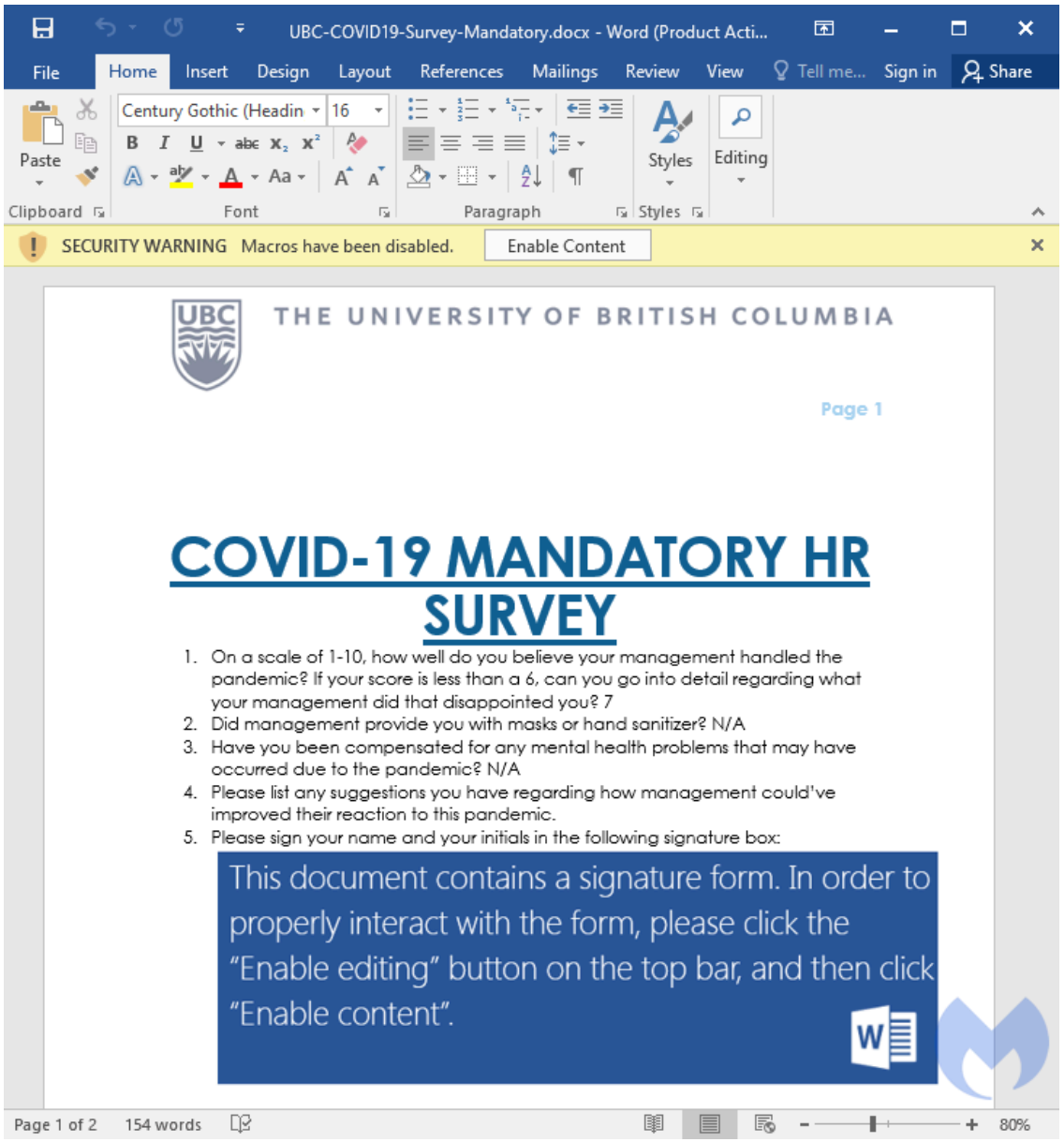
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

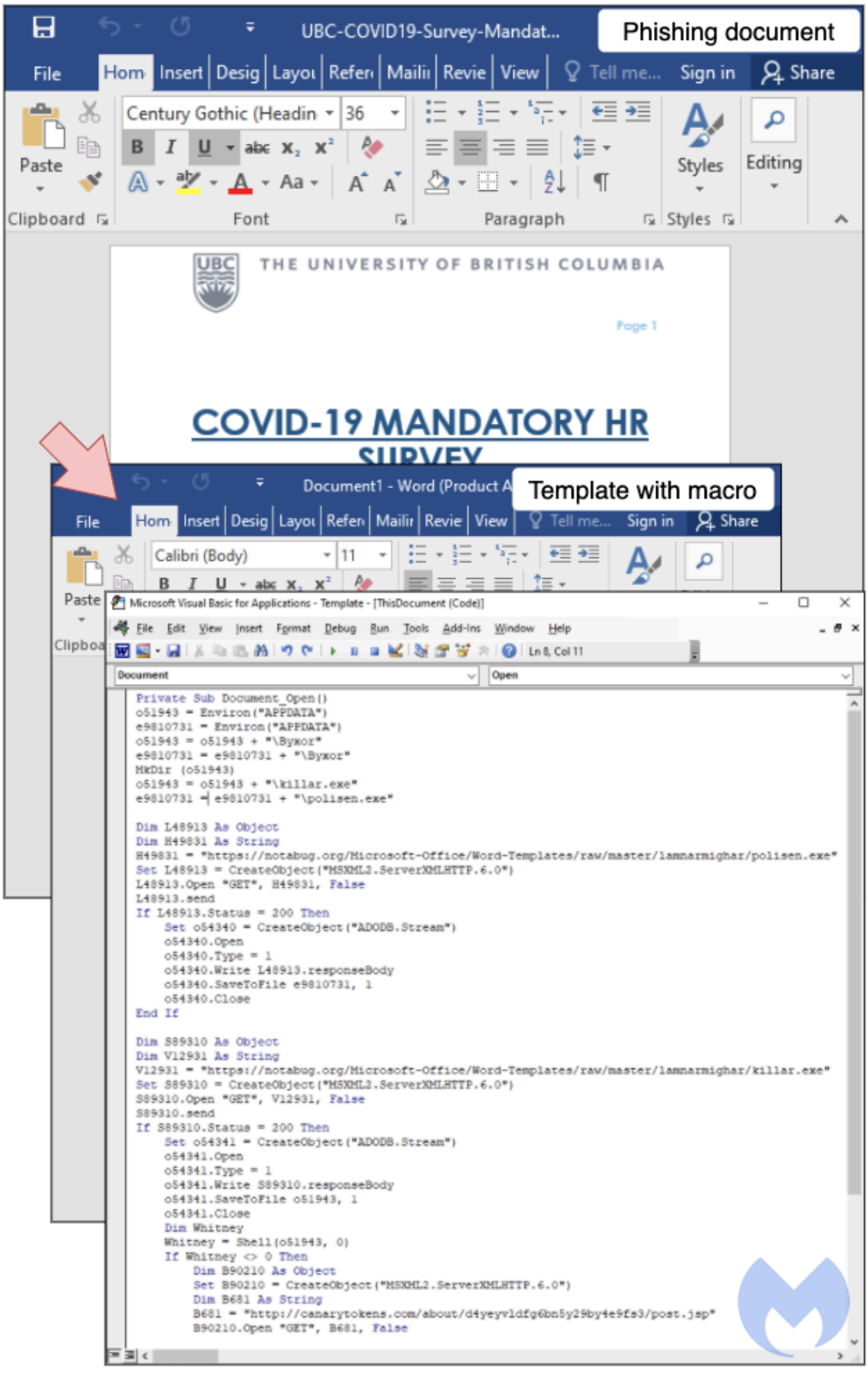
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

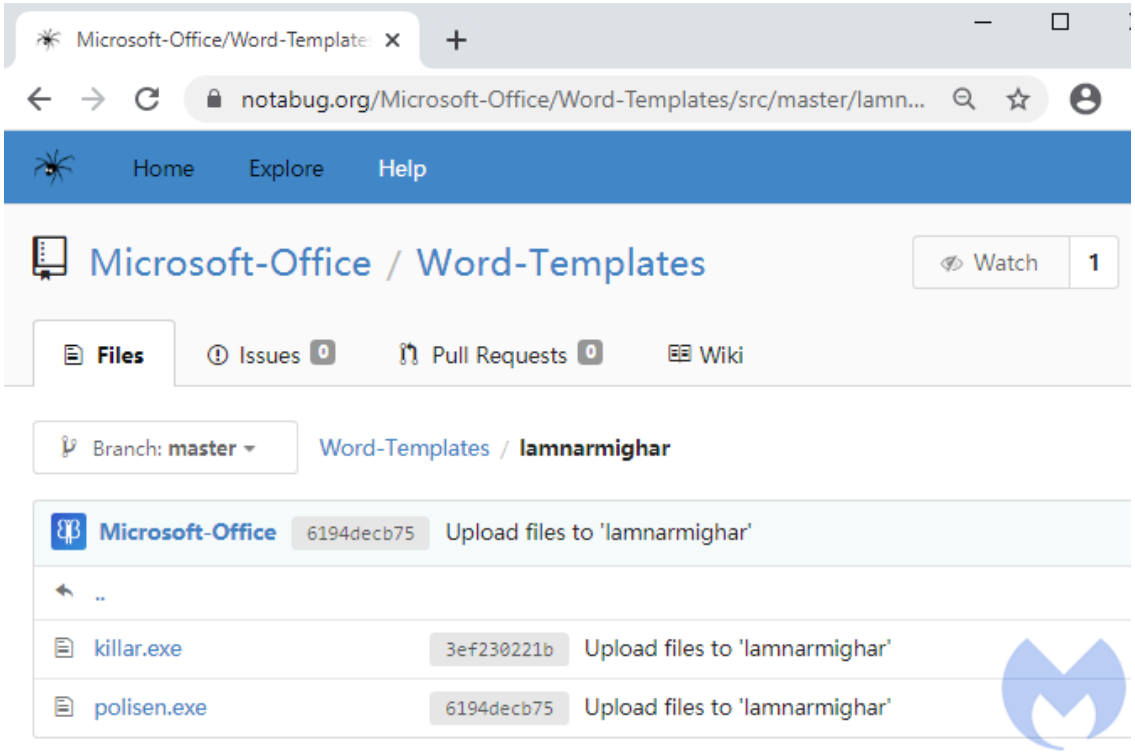
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



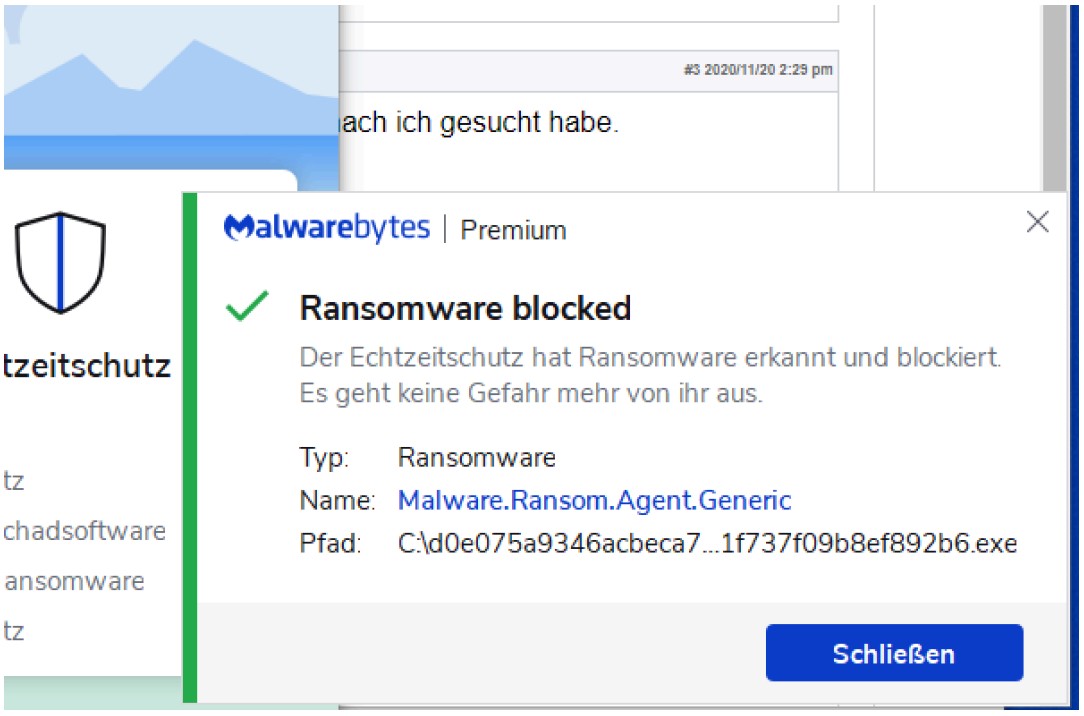
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

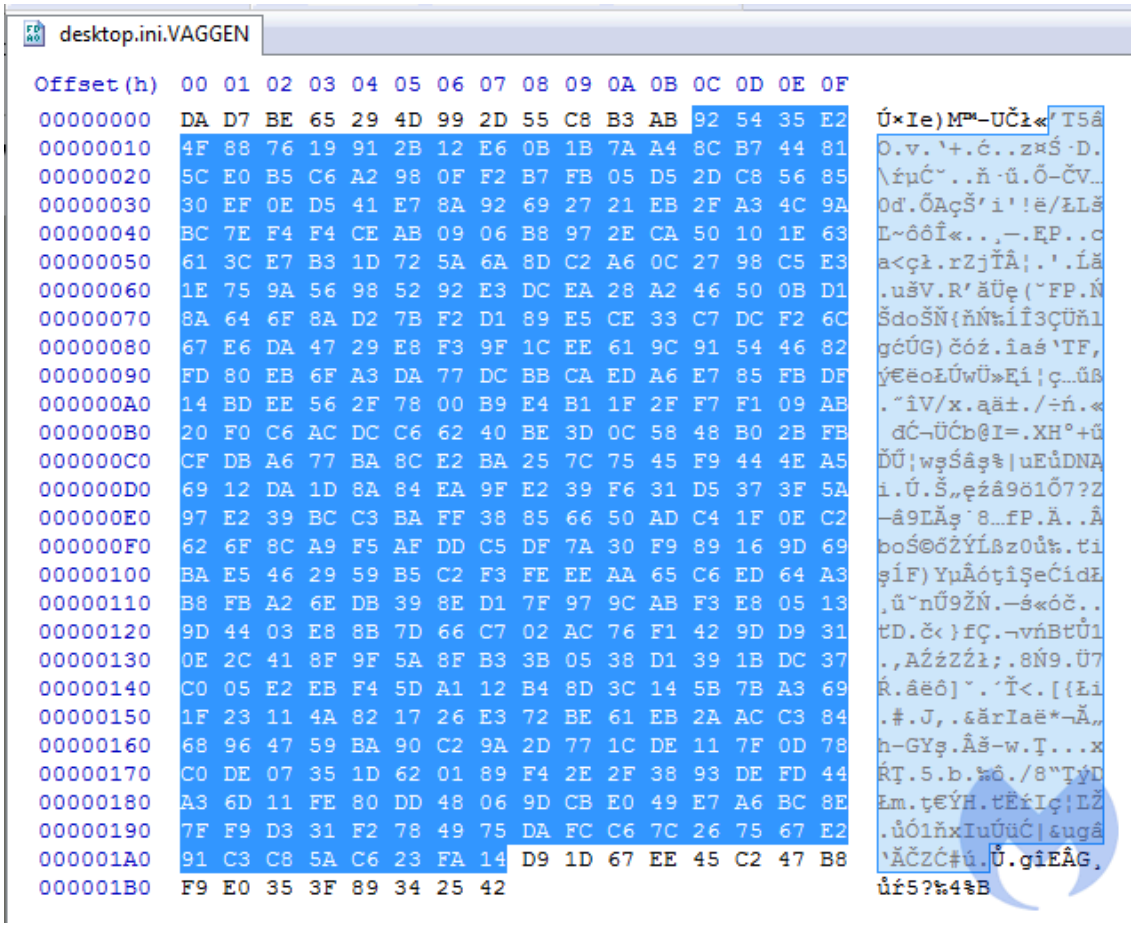
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

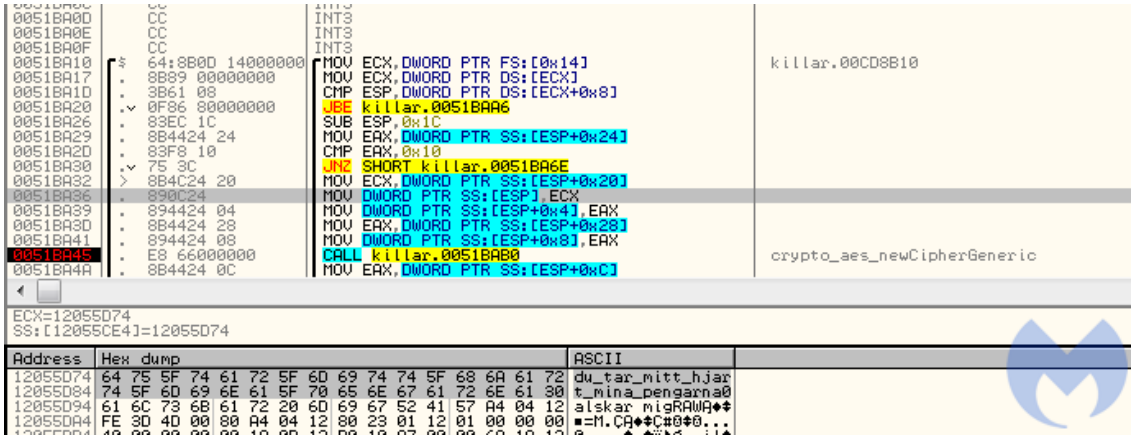


The content of the output file (with .VAGGEN extension) contains:

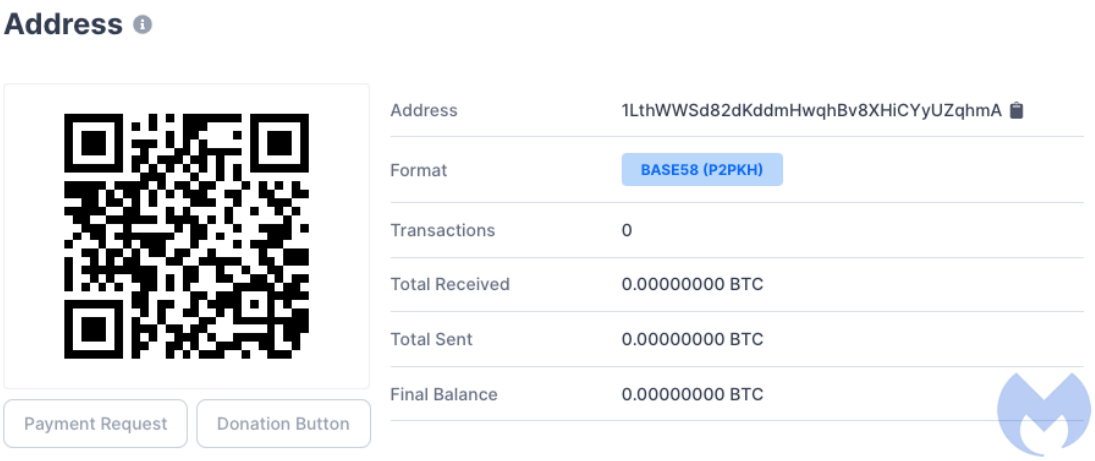
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

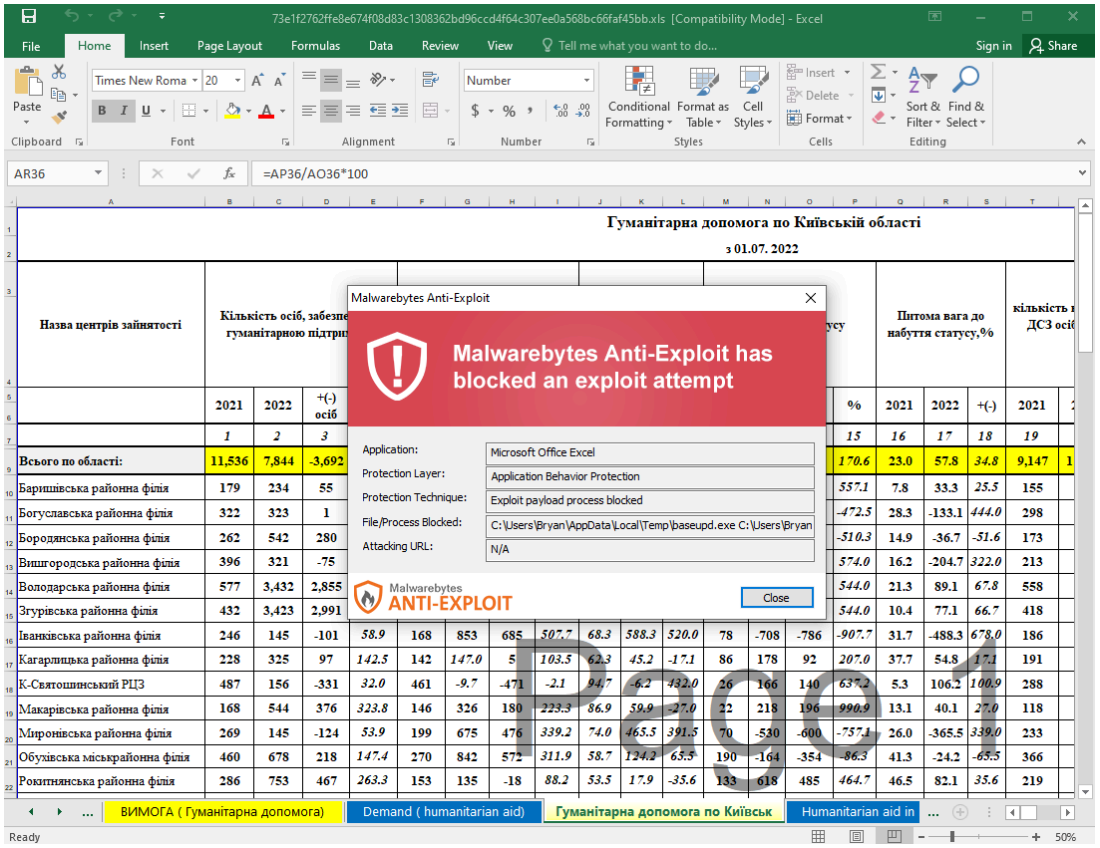
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

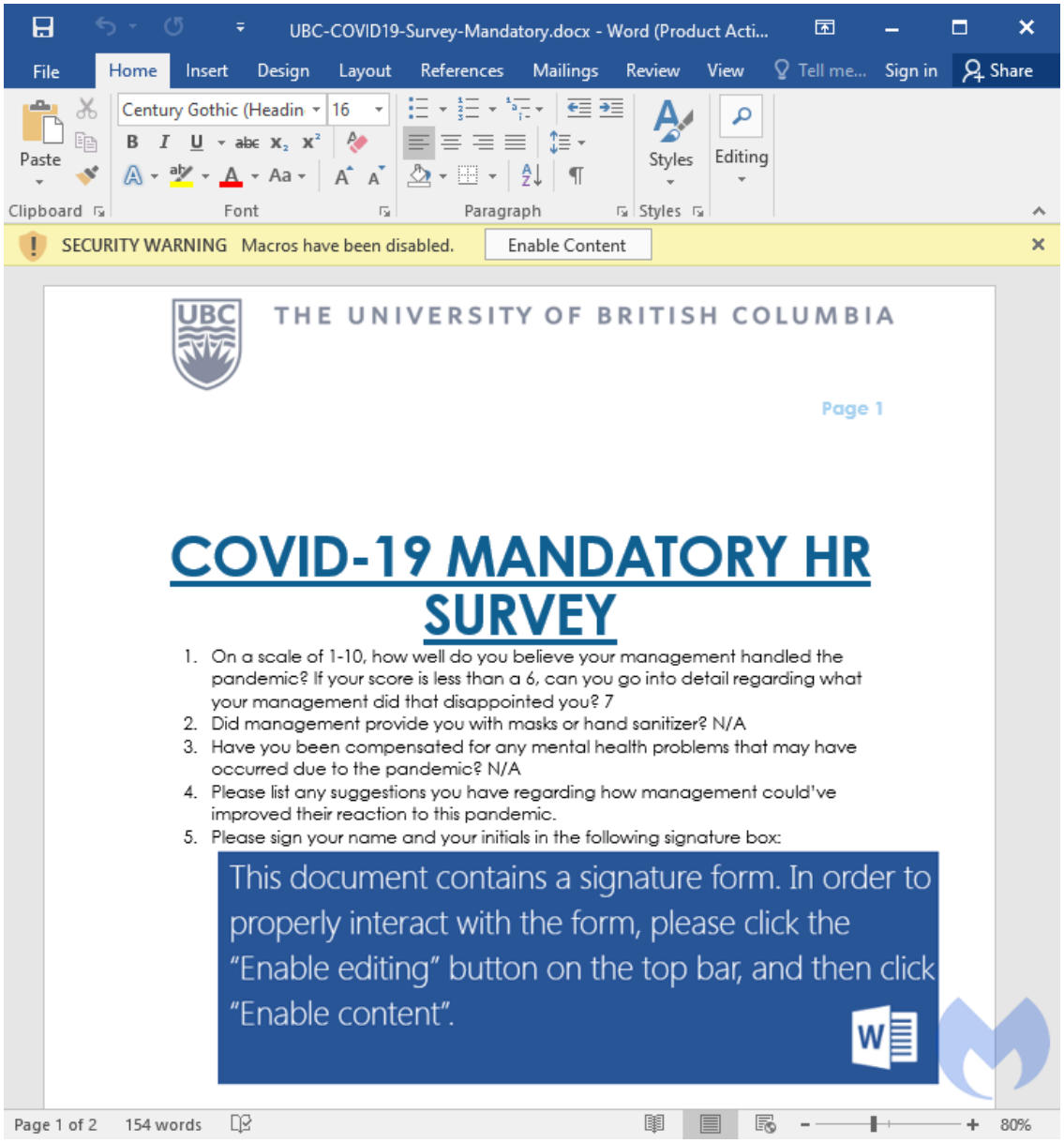
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

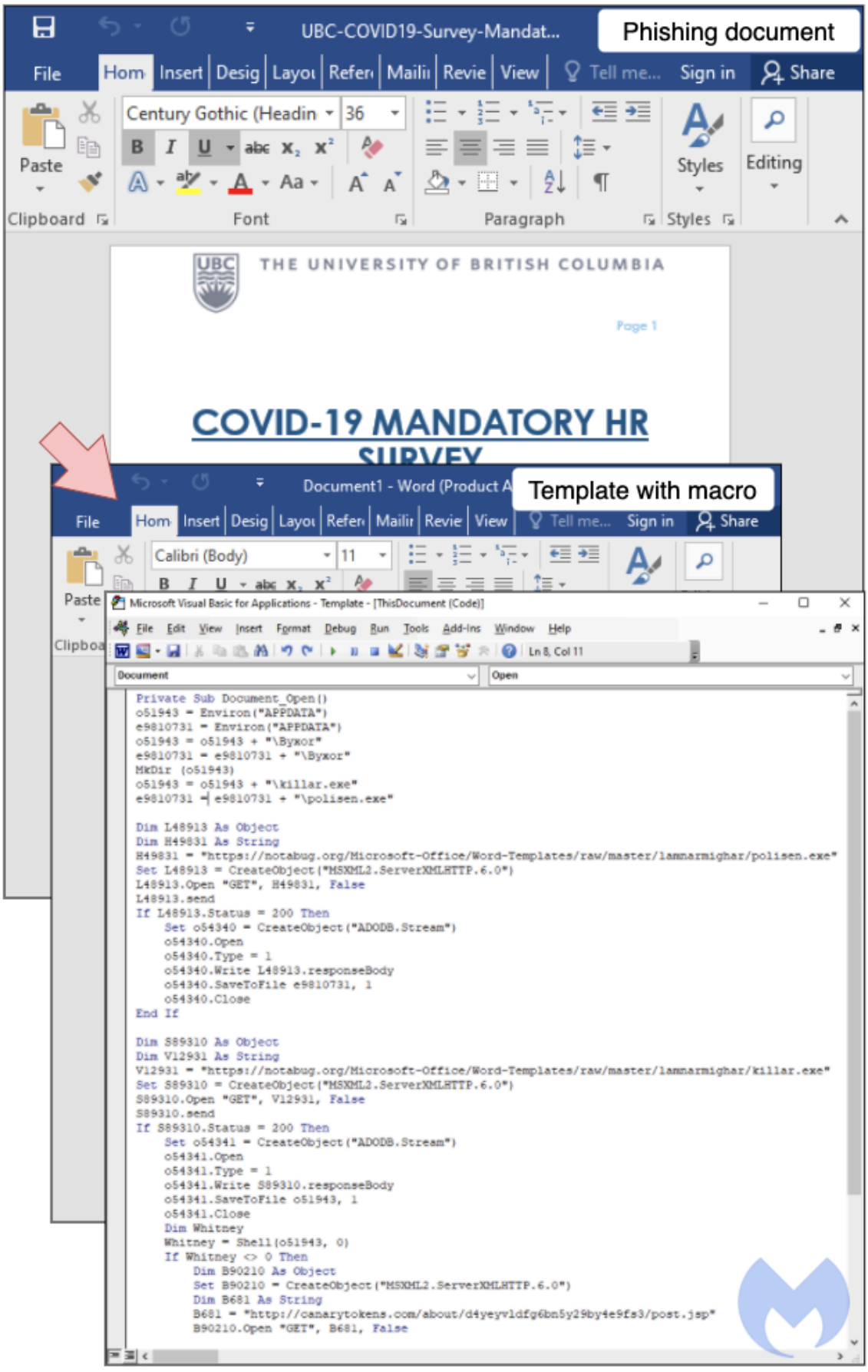
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

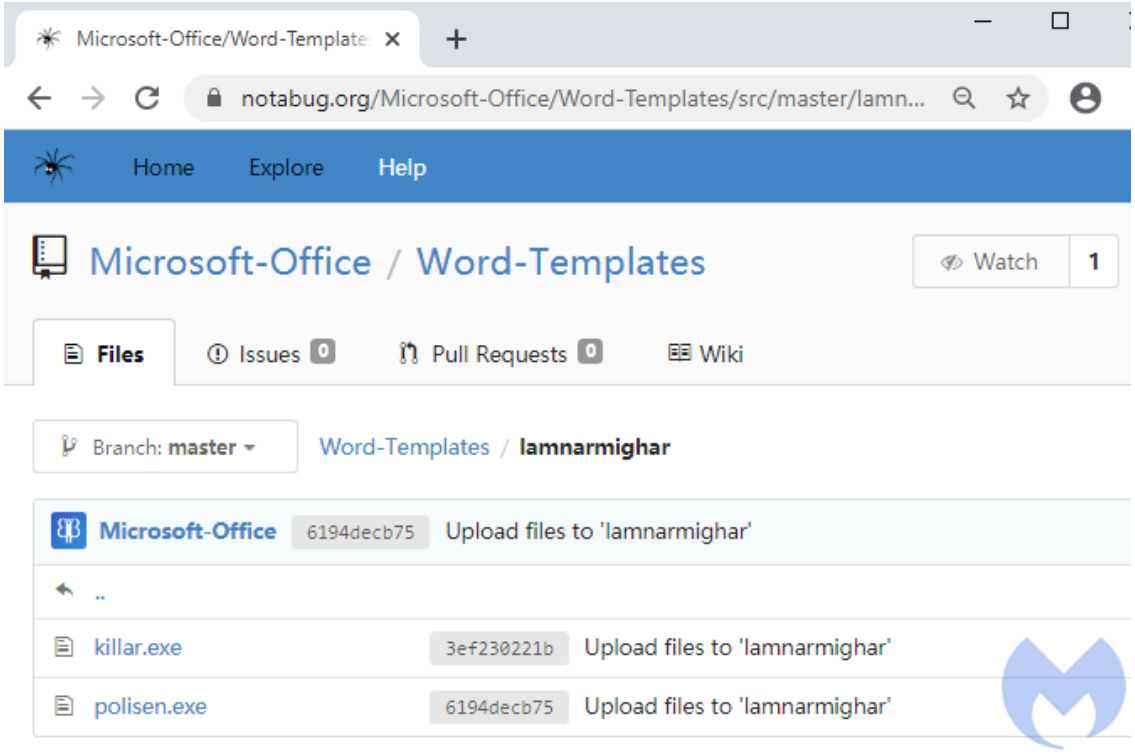
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



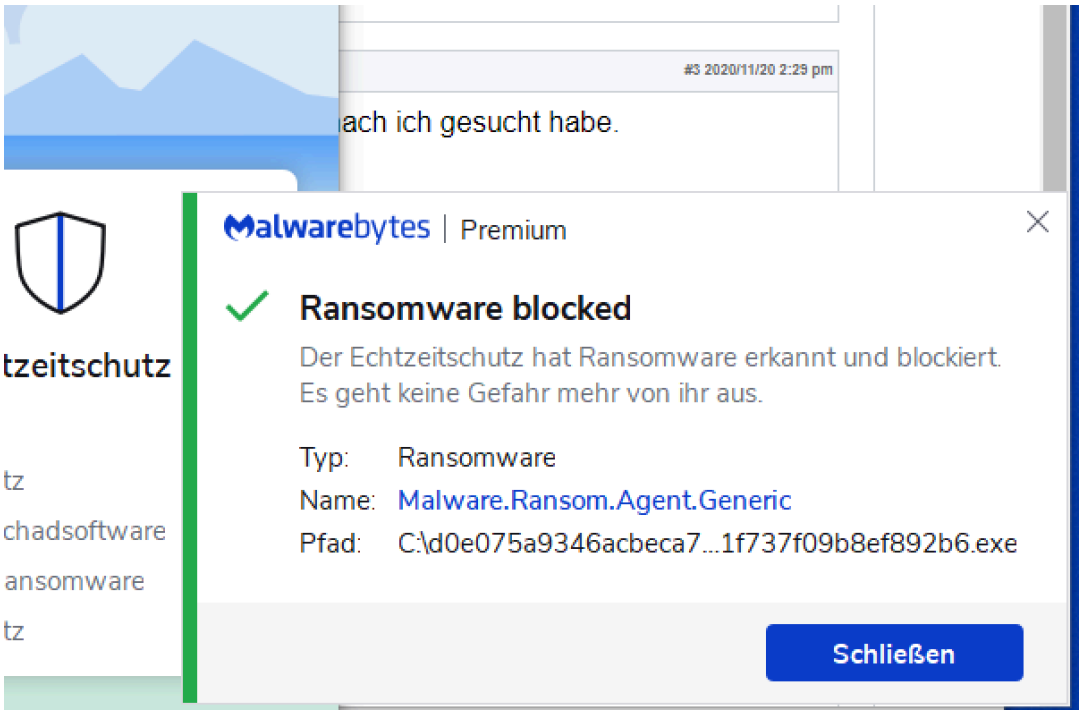
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

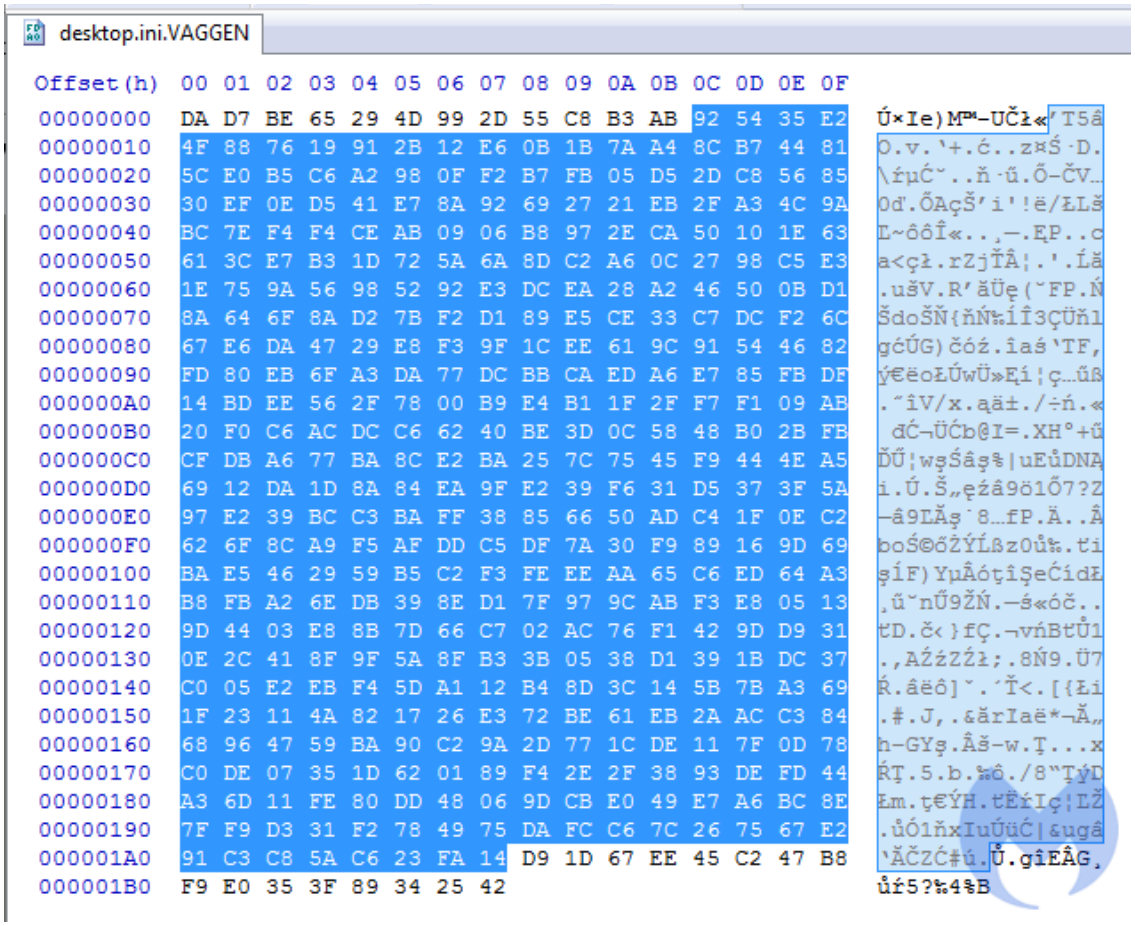
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

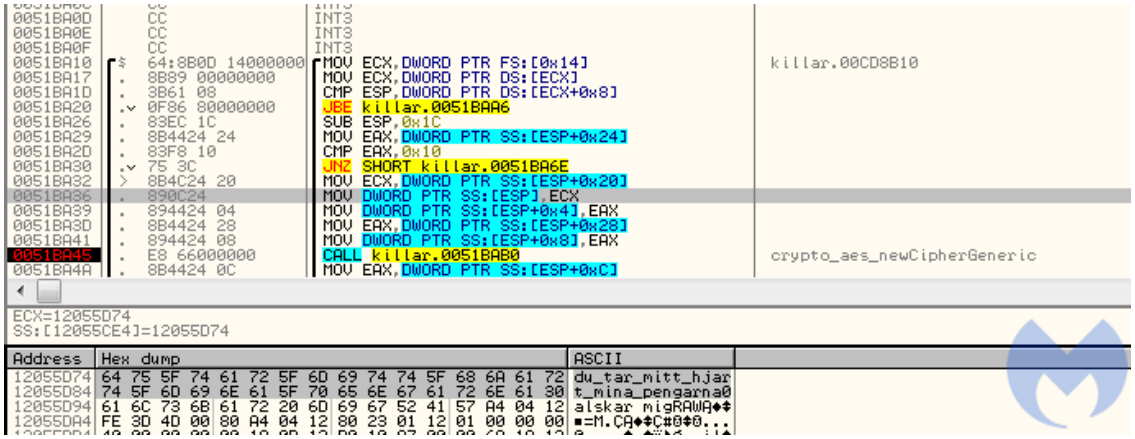


The content of the output file (with .VAGGEN extension) contains:

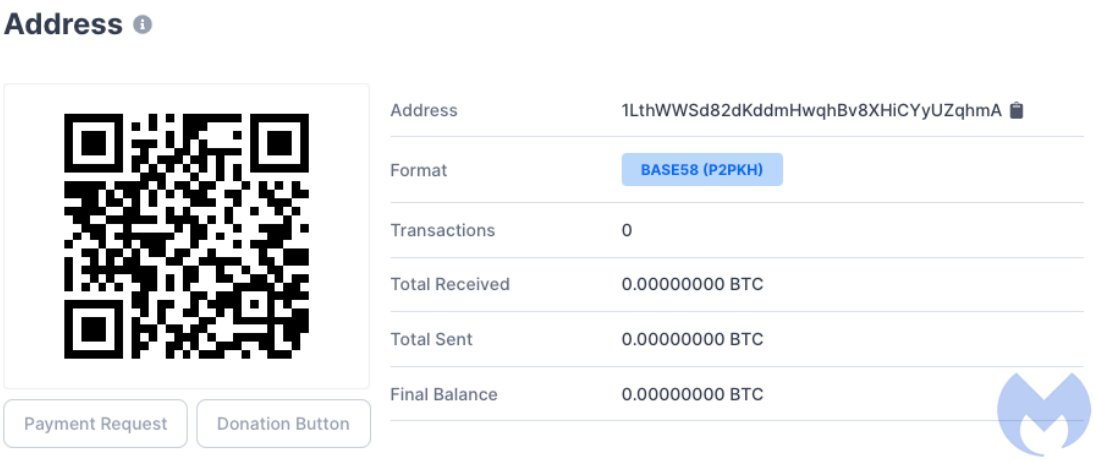
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

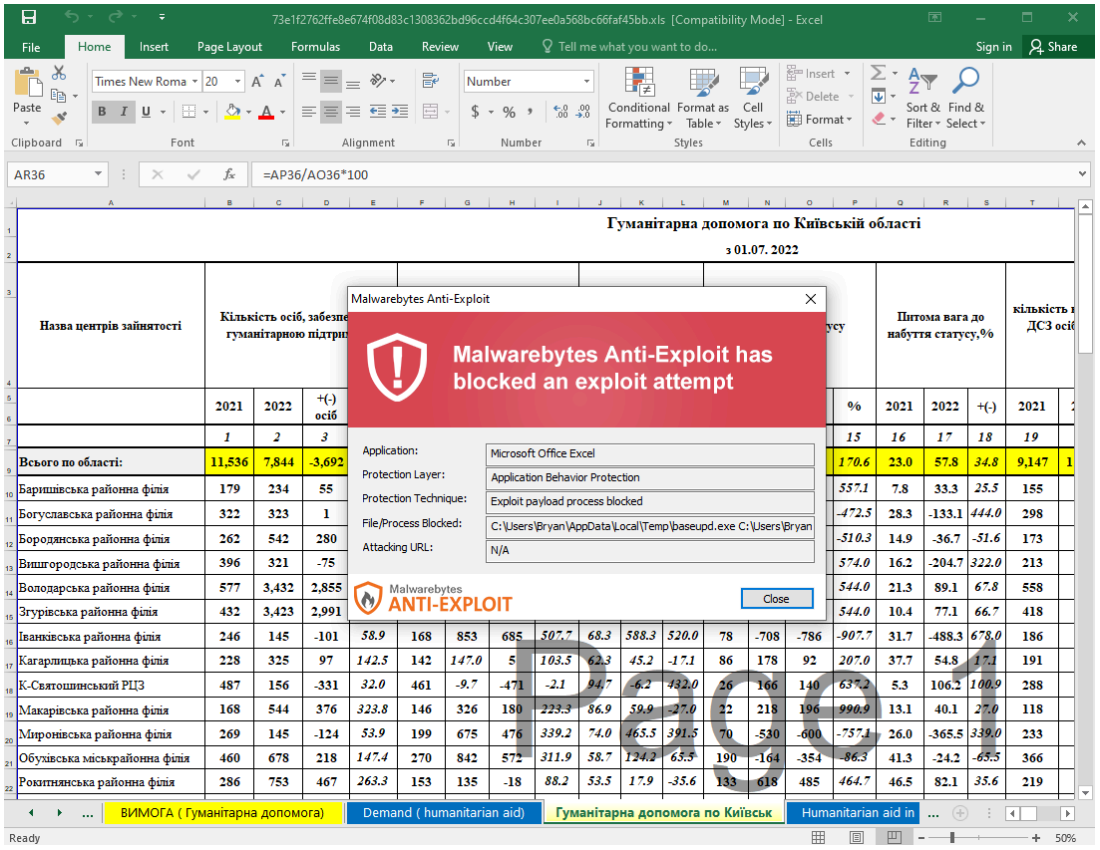
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

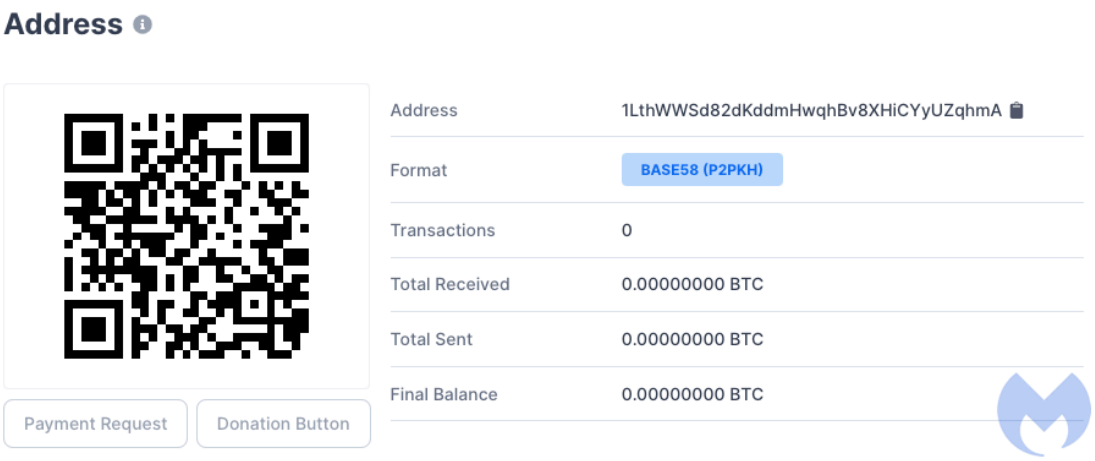
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



Unusually low ransom amount

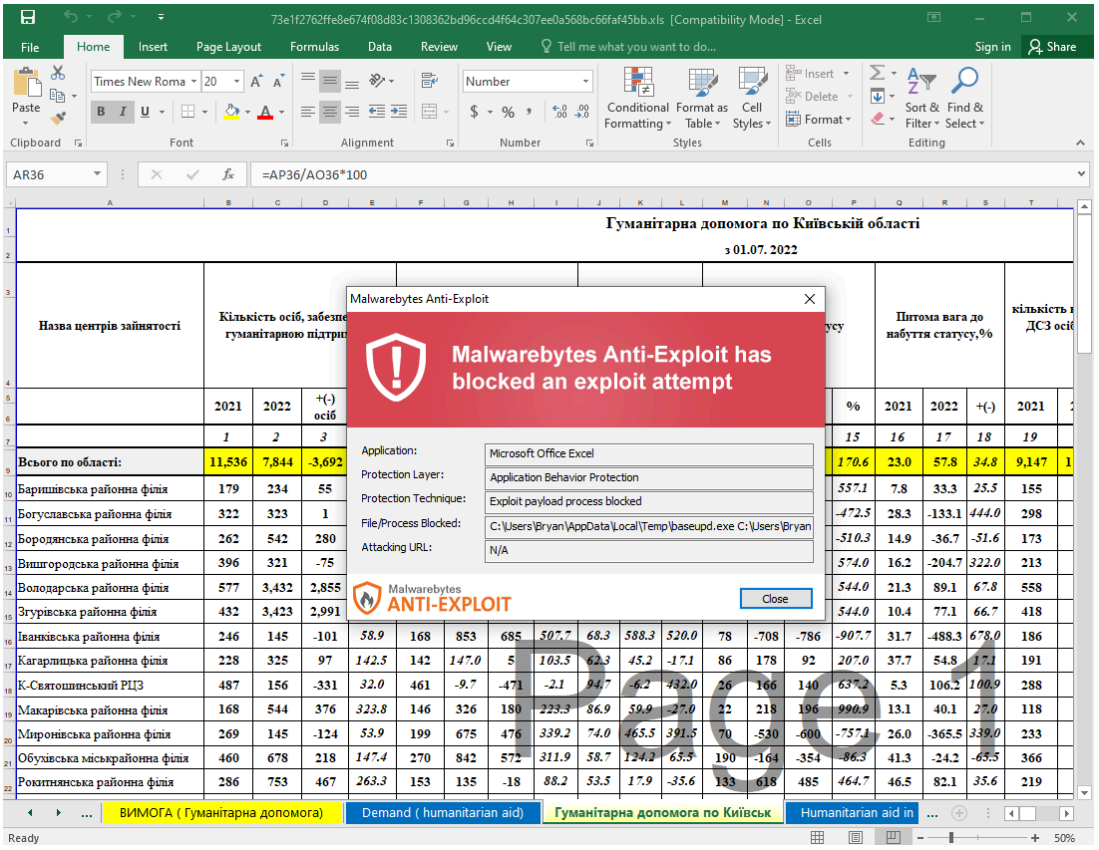
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

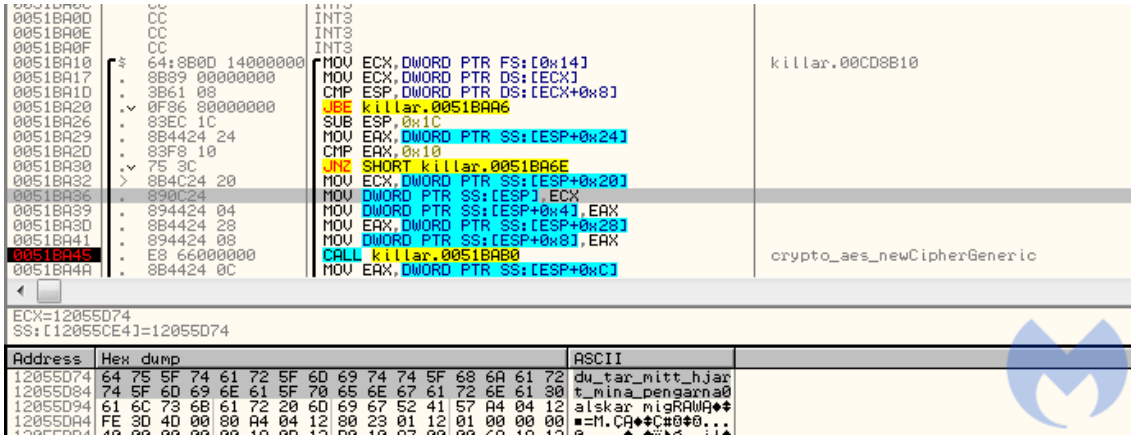
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

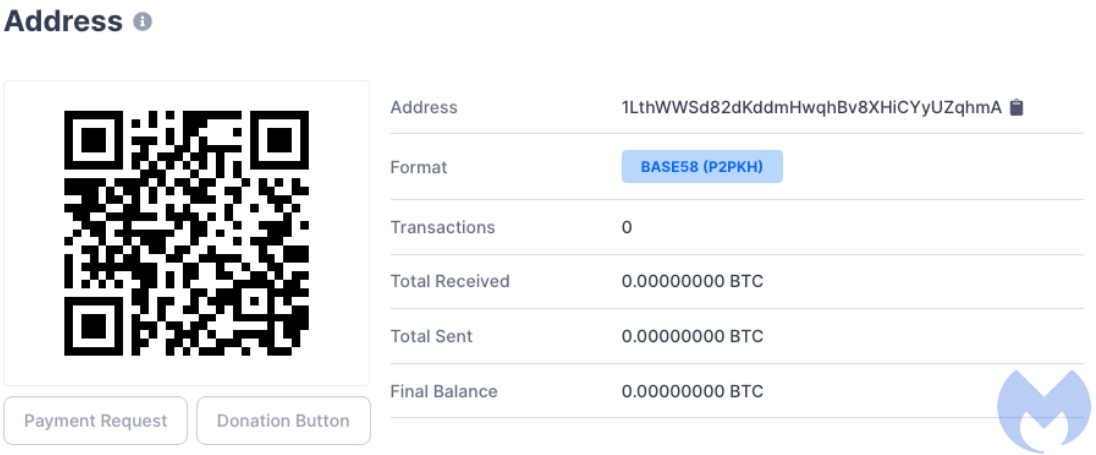
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

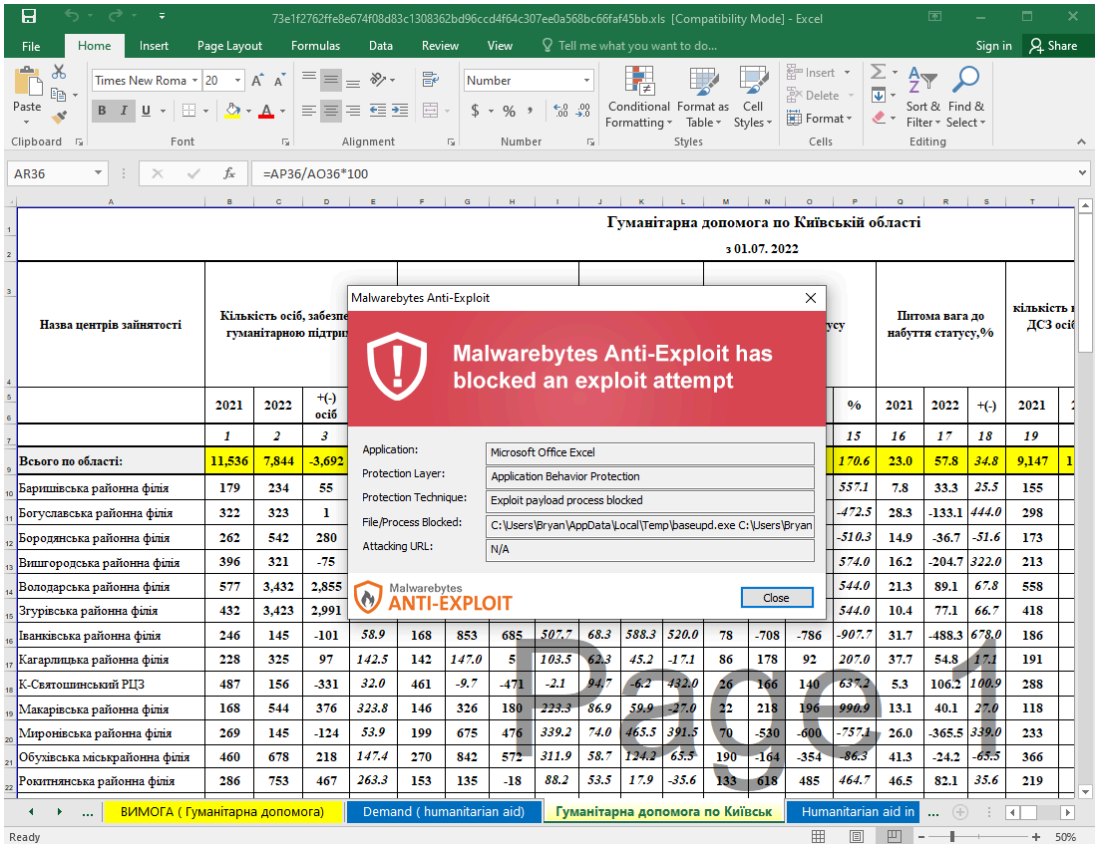
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

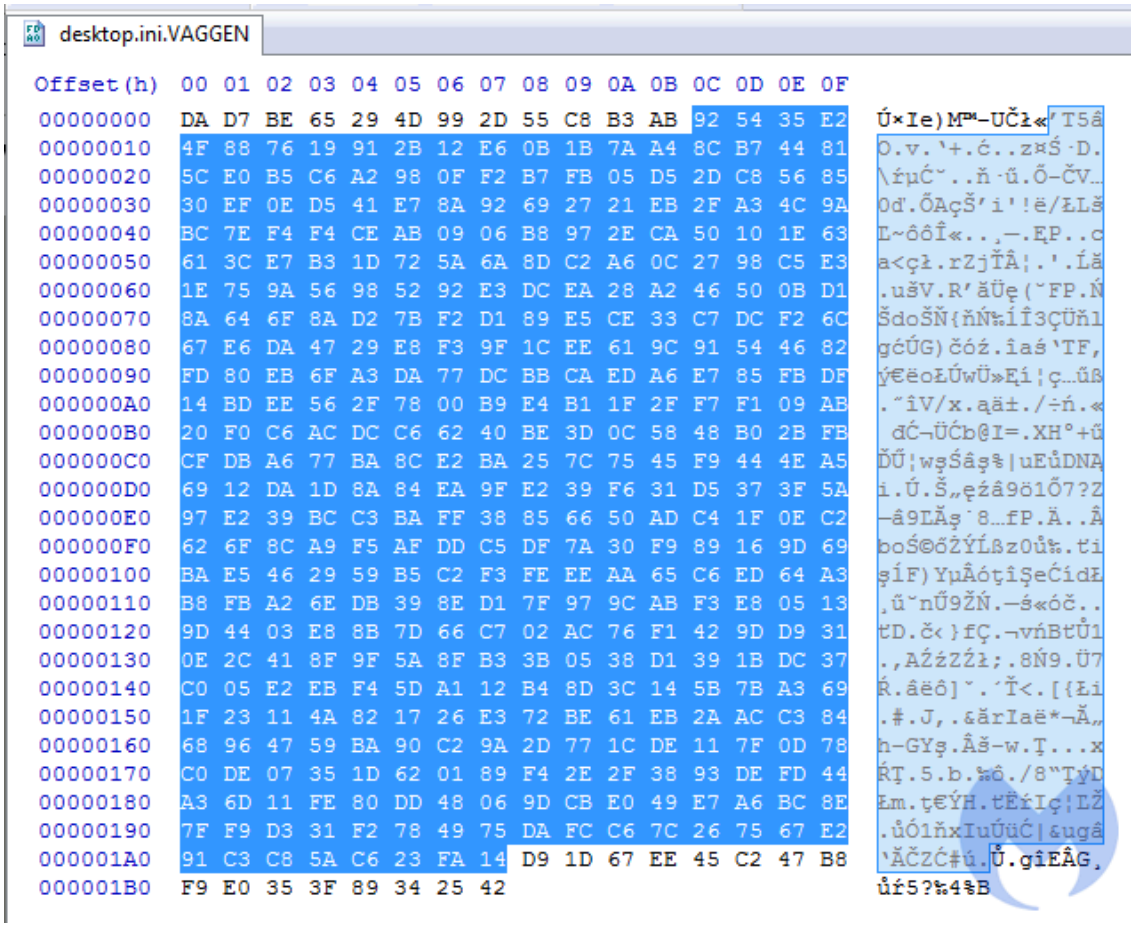
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

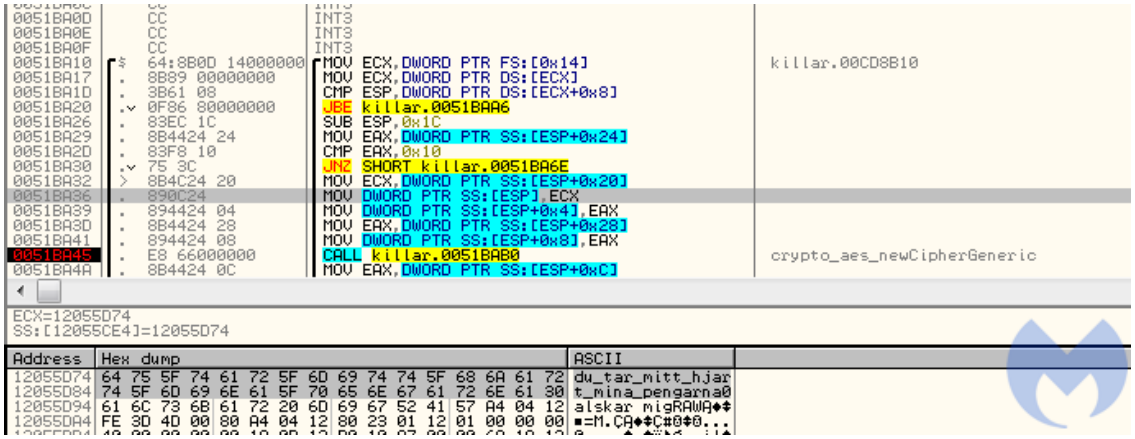
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

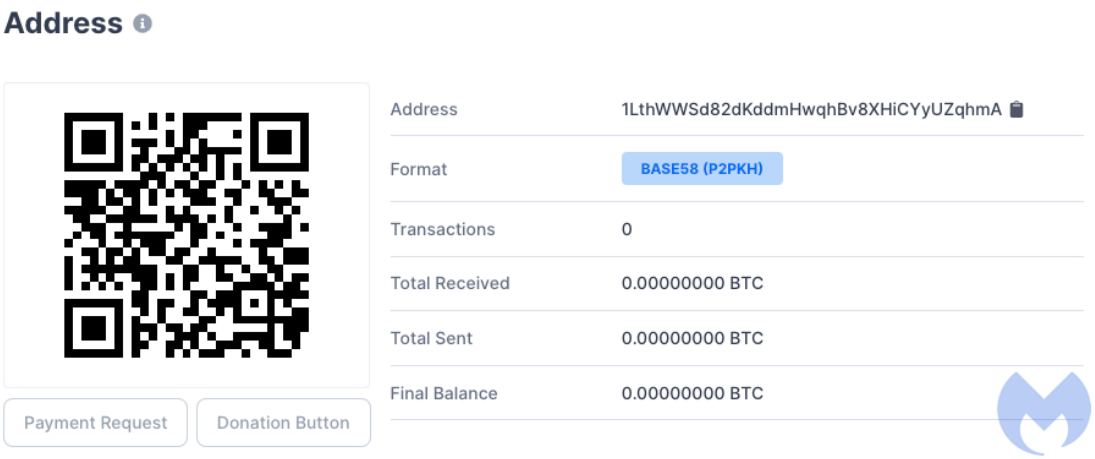
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

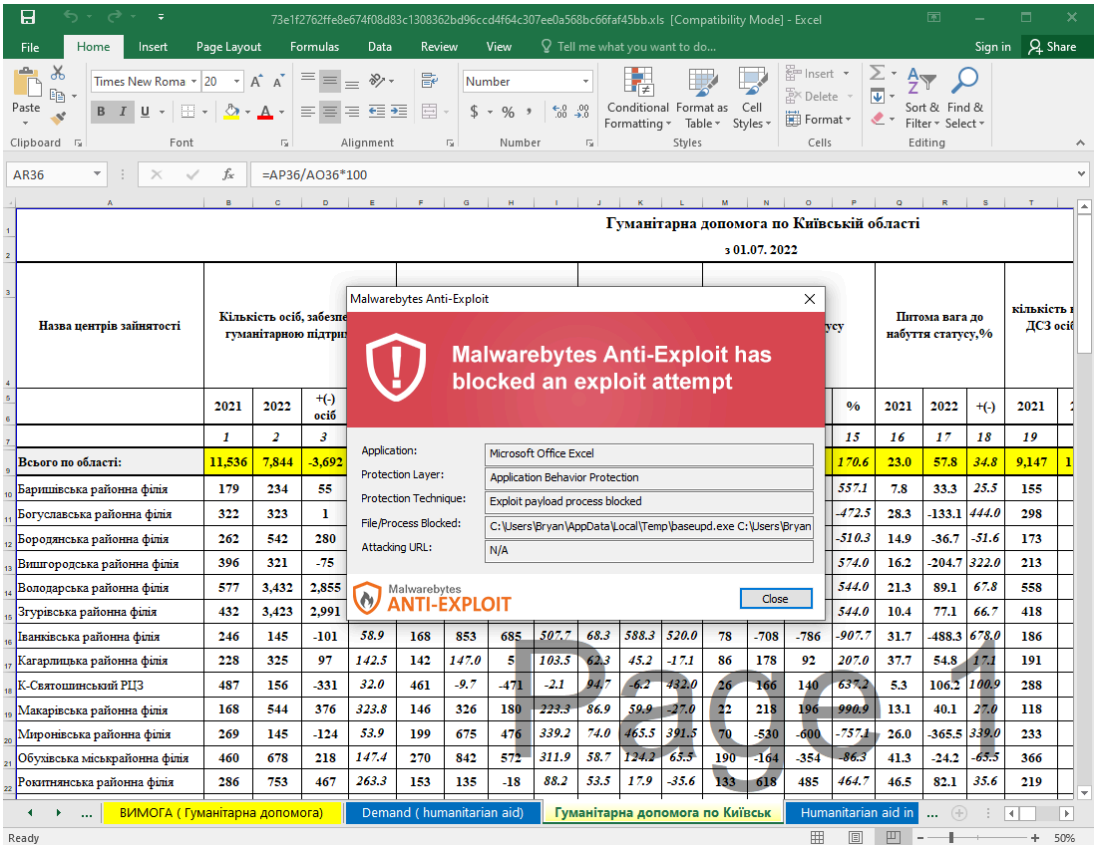
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

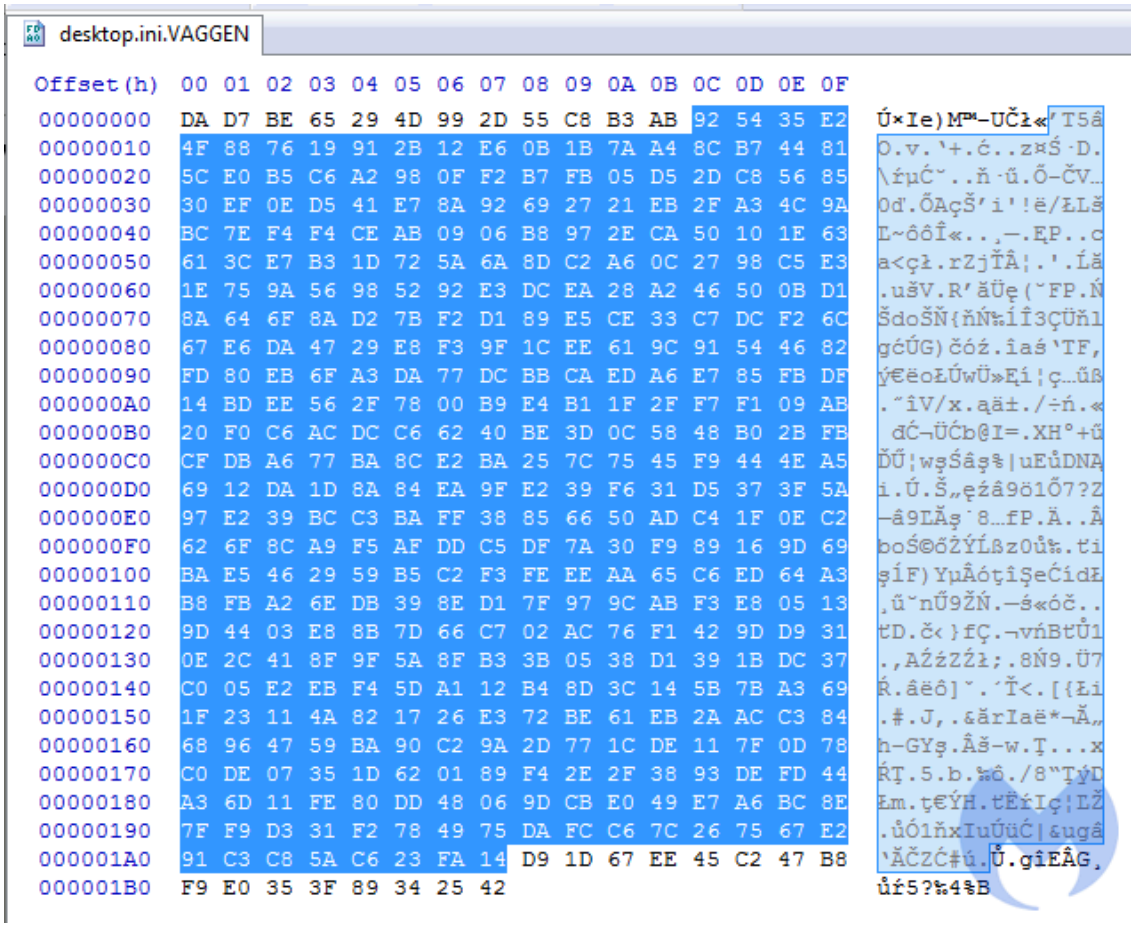
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

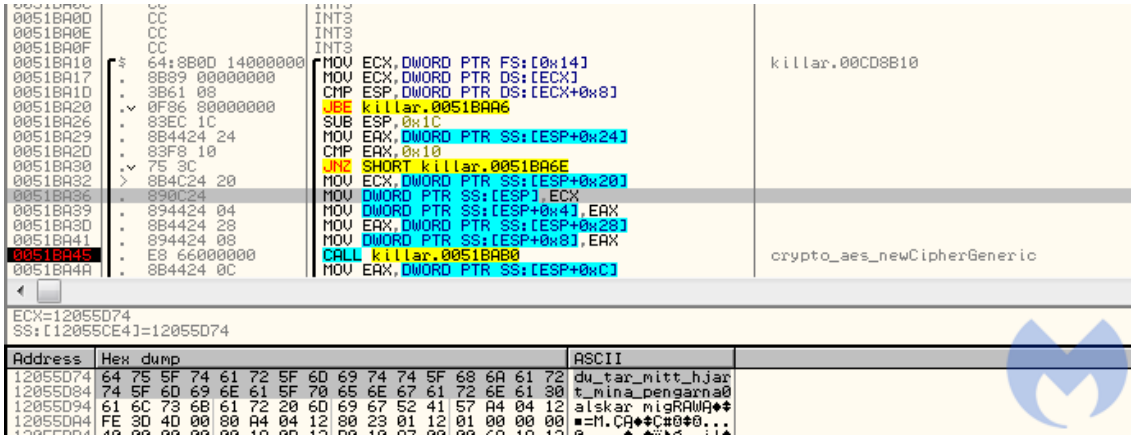


The content of the output file (with .VAGGEN extension) contains:

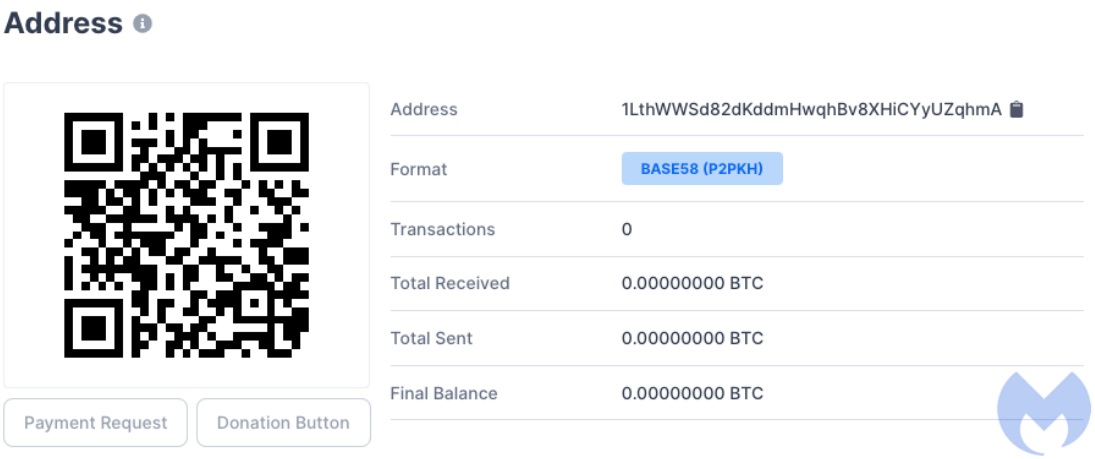
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

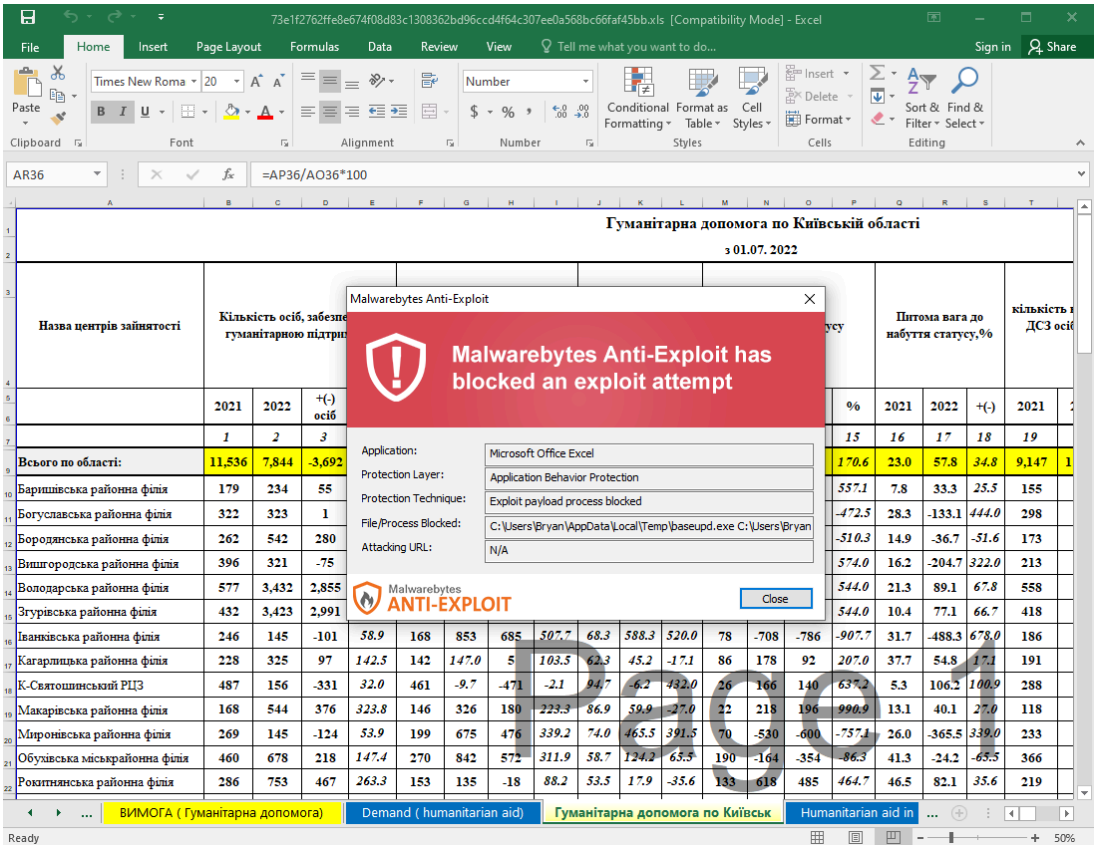
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
46  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
47    runtime_morestack_noctxt();
48  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
49  ((void (*)(void))loc_45CDA0)();
50  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
51  v38 = v4;
52  v30 = v5;
53  v31 = v6;
54  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
55  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
56  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
57  v32 = v10;
58  v11 = *(void (__cdecl **)(int))(v10 + 16);
59  v39 = v12;
60  v11(v12);
61  v29 = v13;
62  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
63  v40 = v15;
64  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
65  v17 = v27;
66  if ( v27 )
67  {
68    if ( v27 )
69      v17 = *(_DWORD *)(v27 + 4);
70    v43 = v17;
71    v44 = v28;
72    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
73  }
```

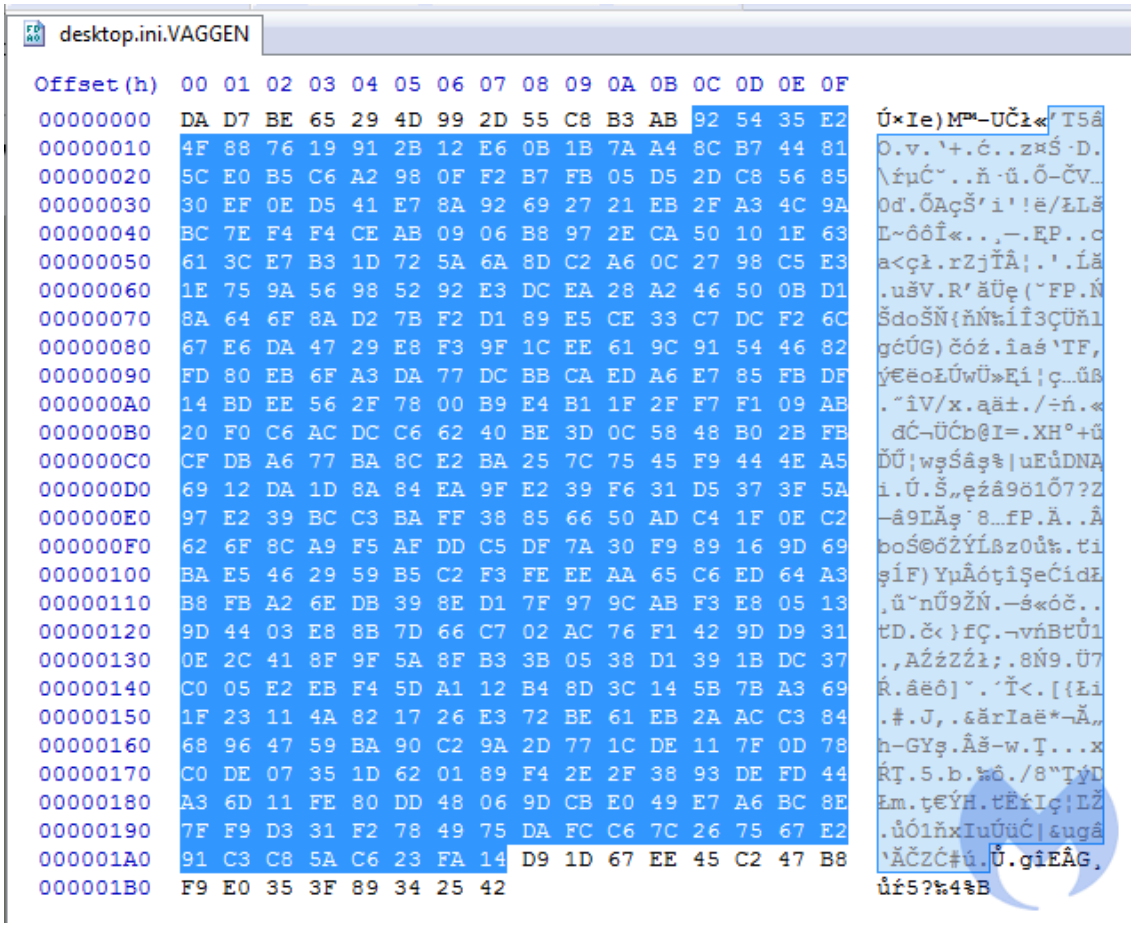


The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

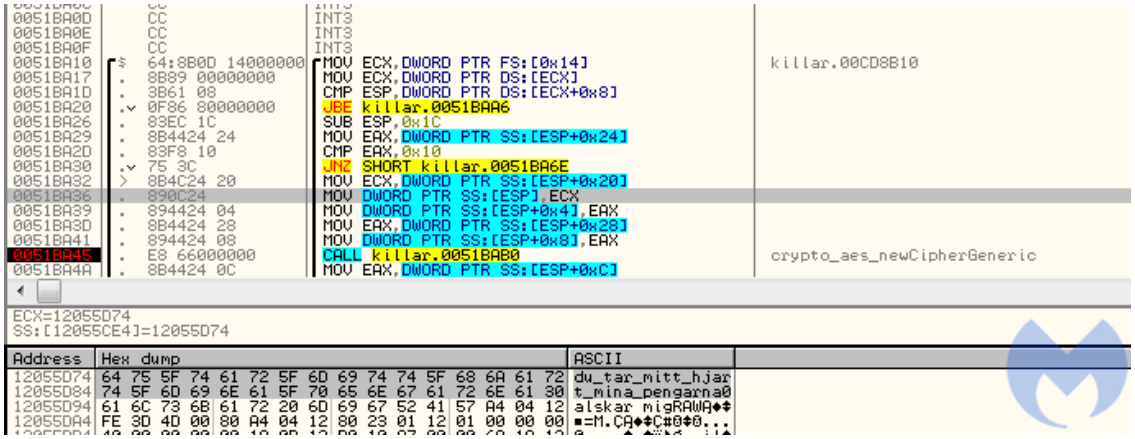


The content of the output file (with .VAGGEN extension) contains:

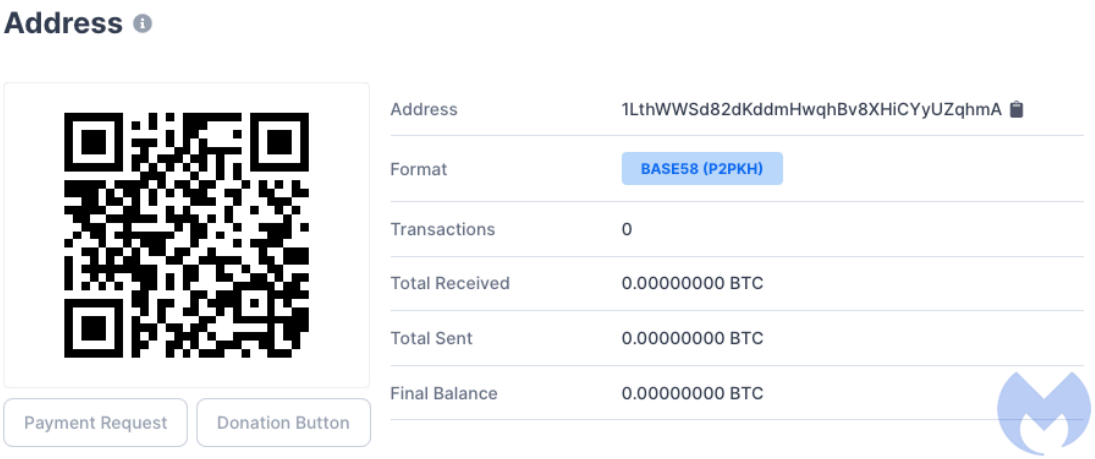
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

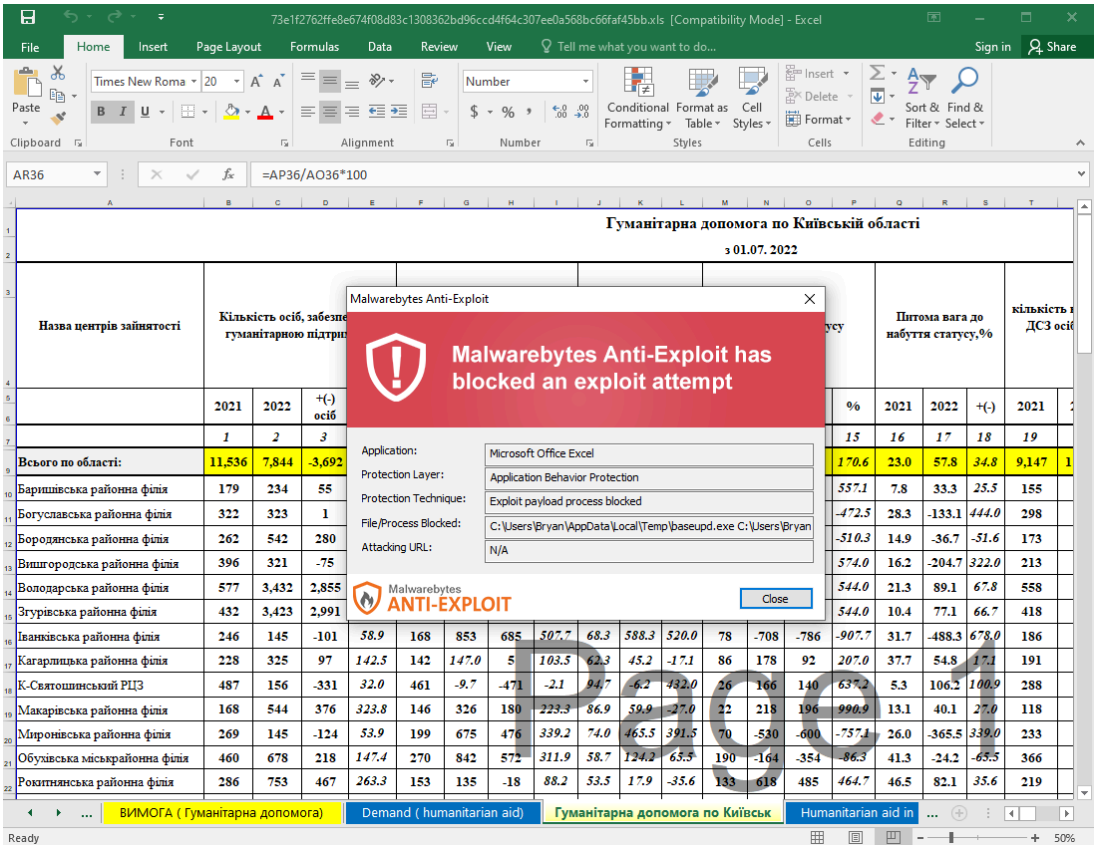
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

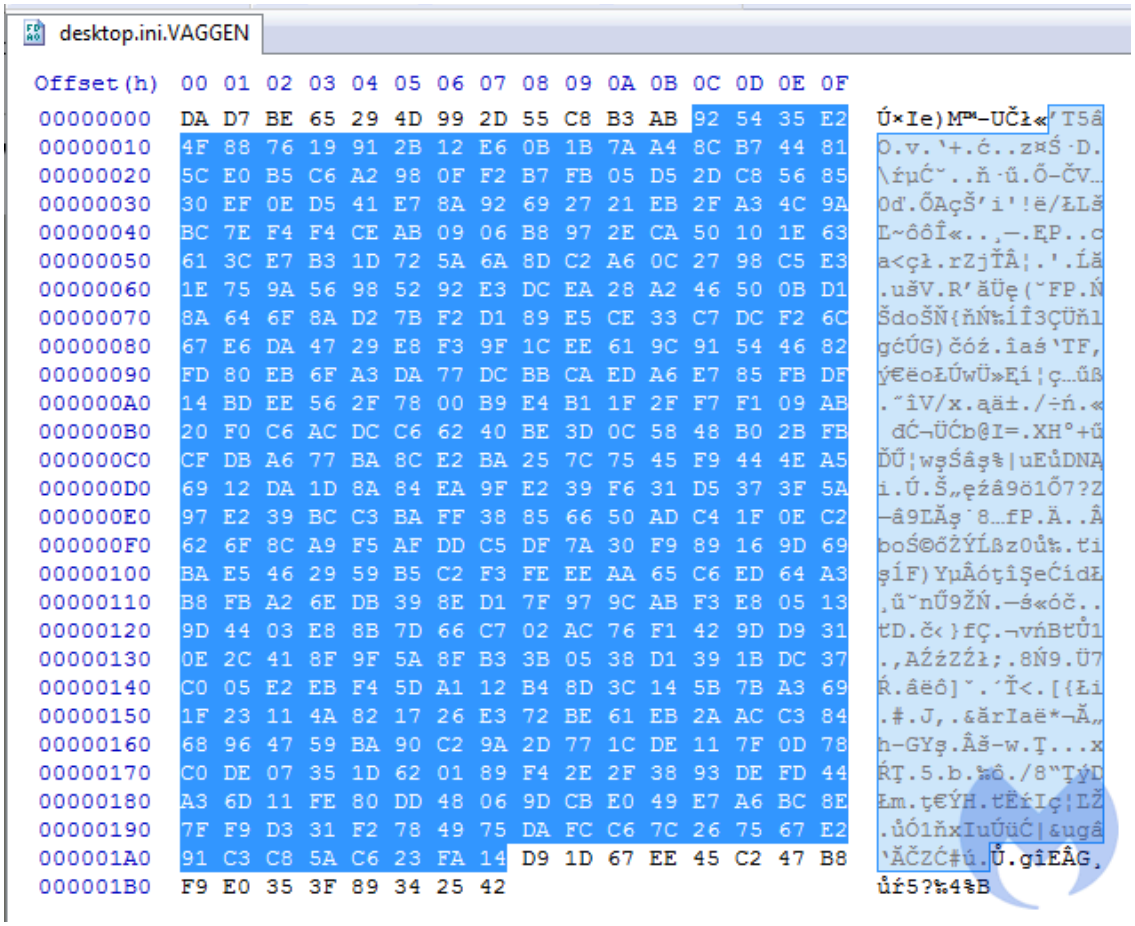
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

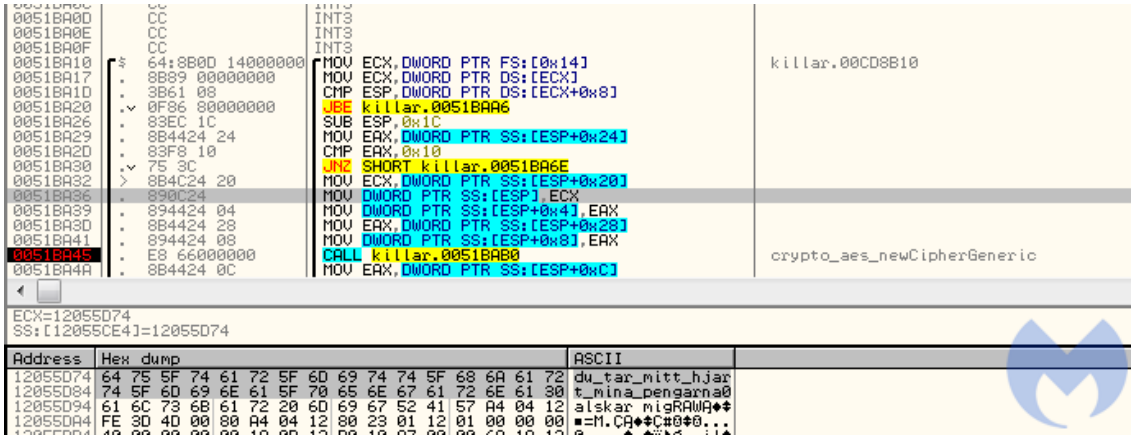


The content of the output file (with .VAGGEN extension) contains:

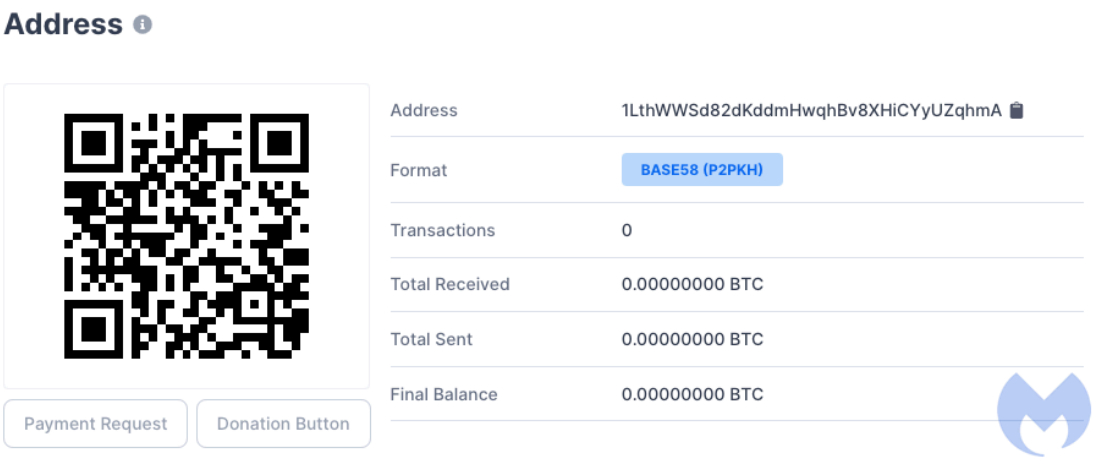
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

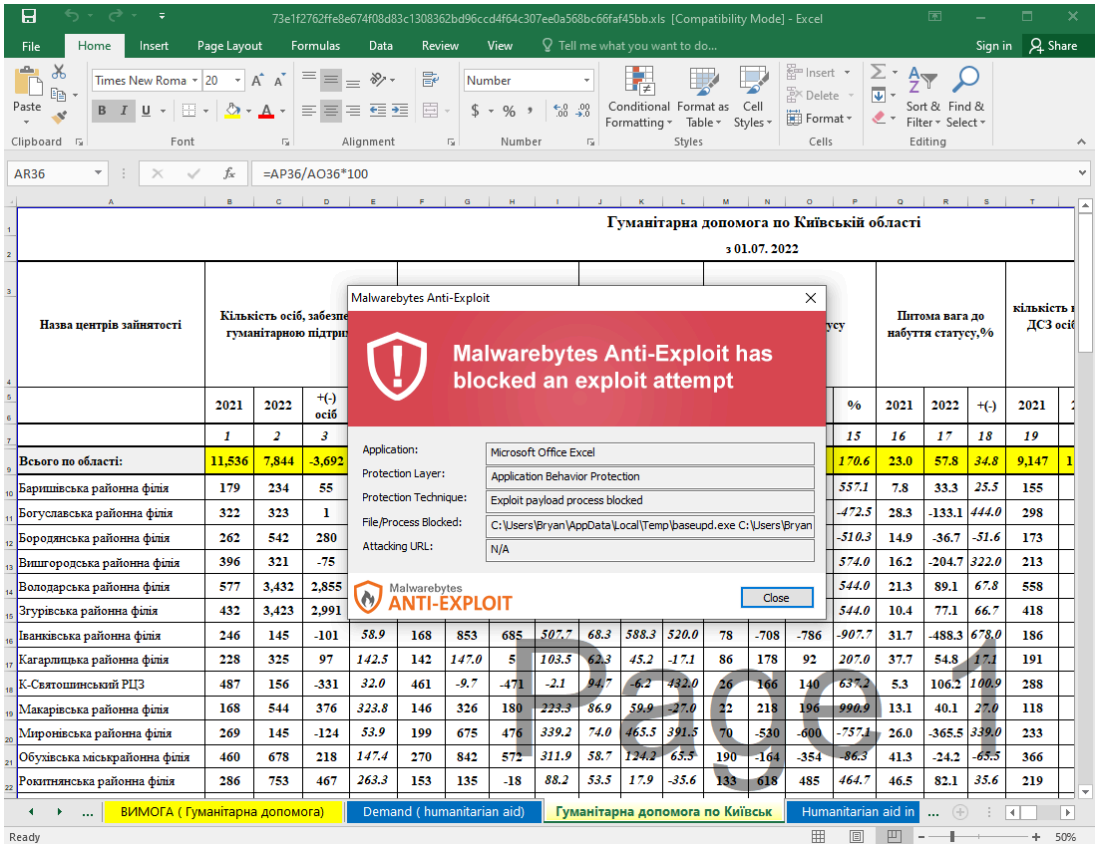
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

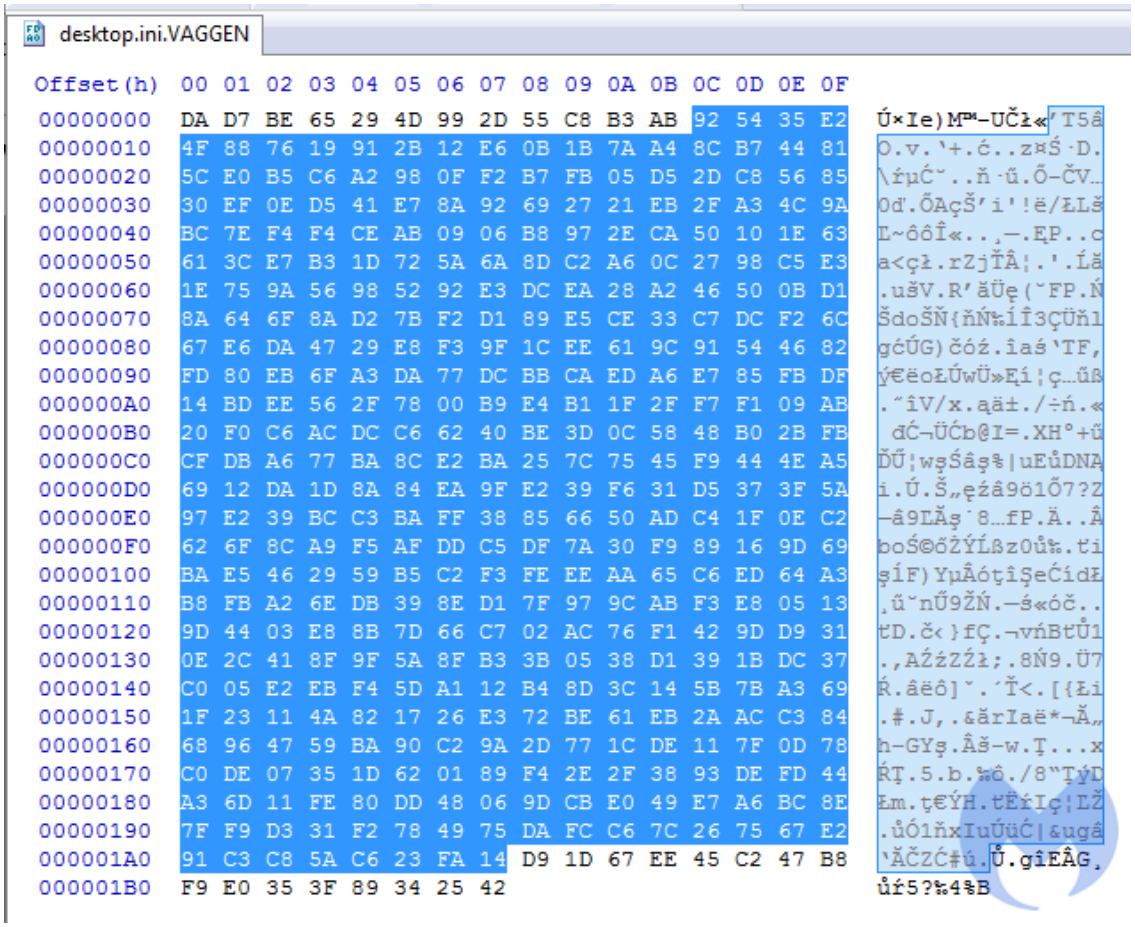
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

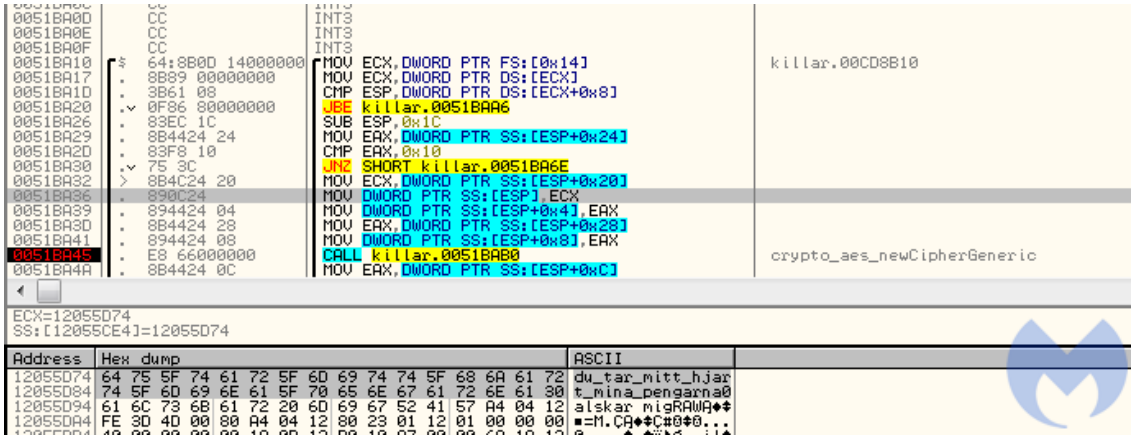


The content of the output file (with .VAGGEN extension) contains:

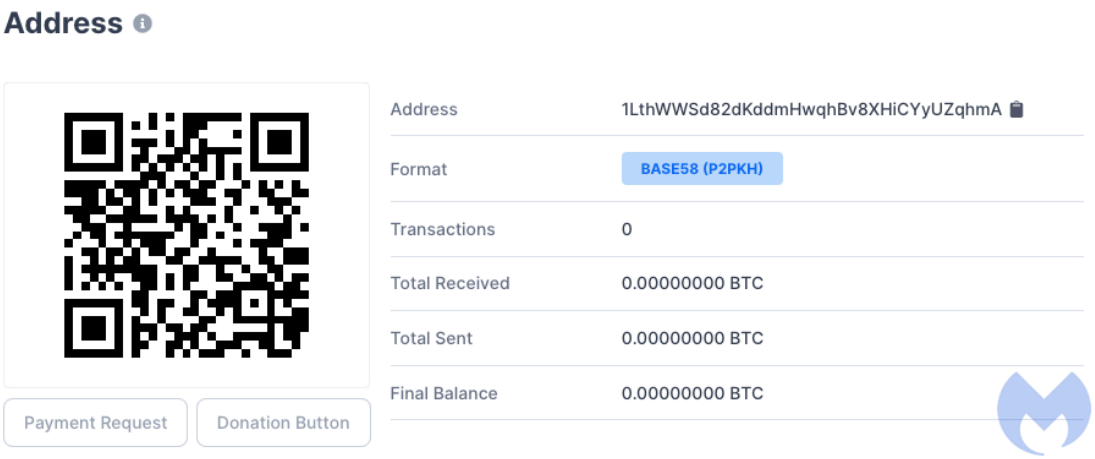
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

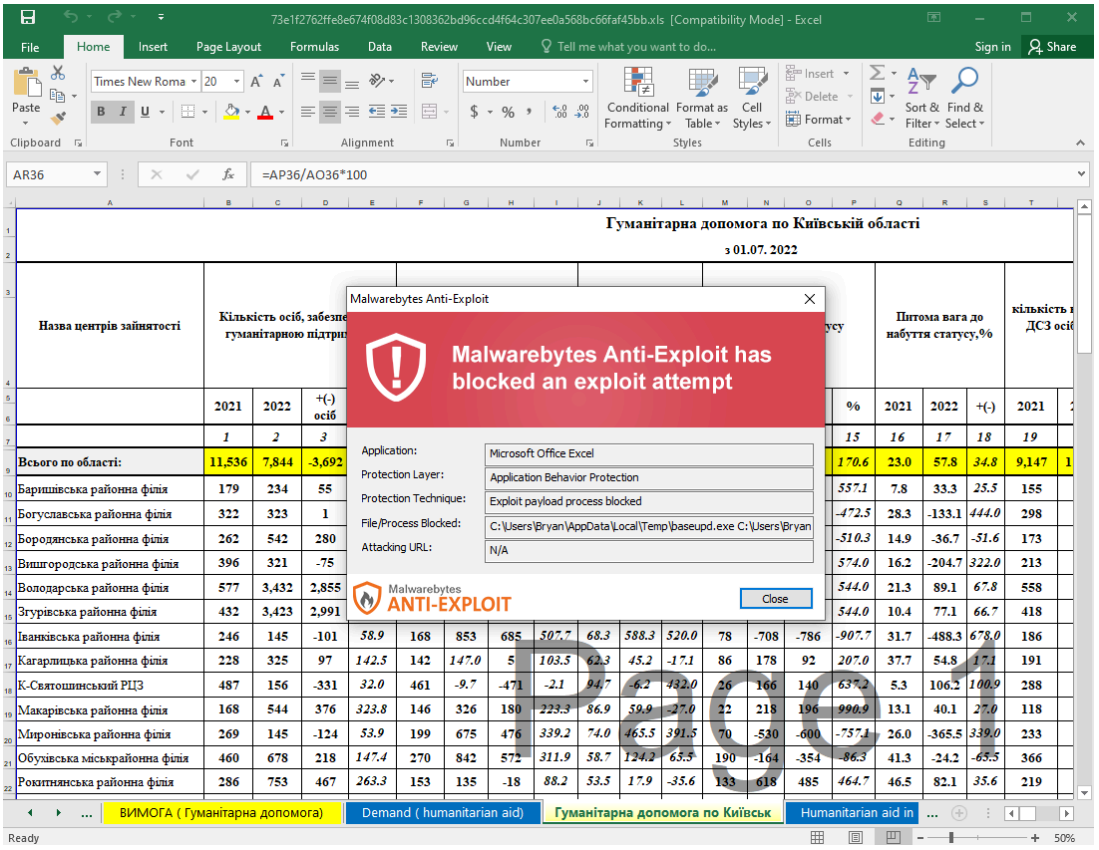
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

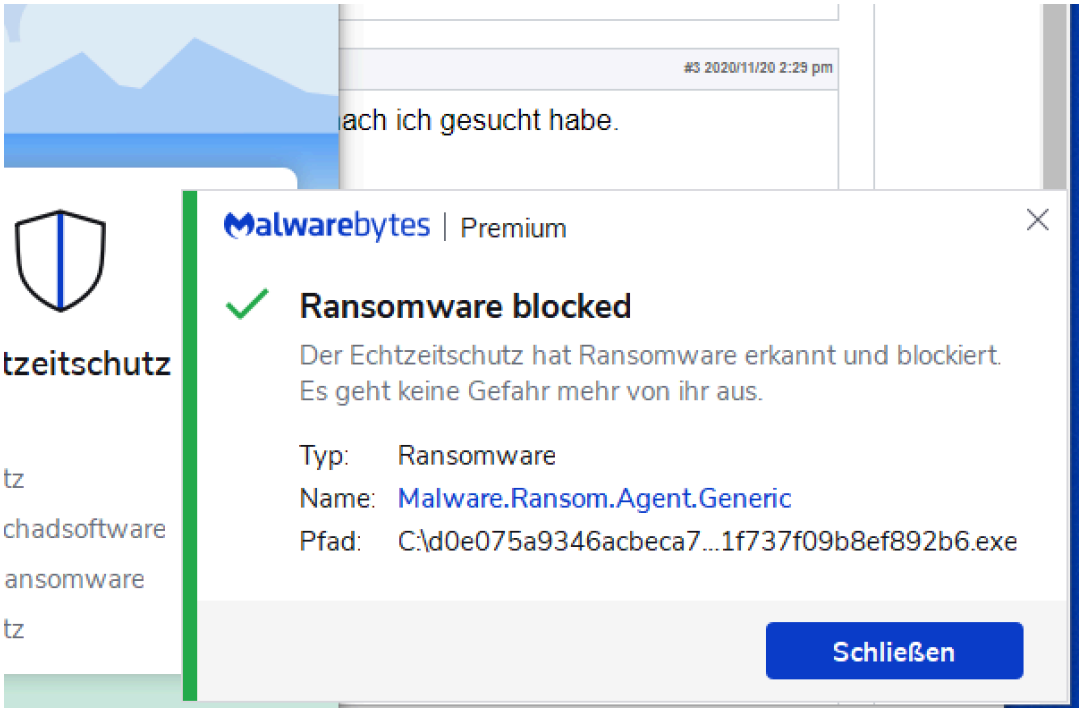
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

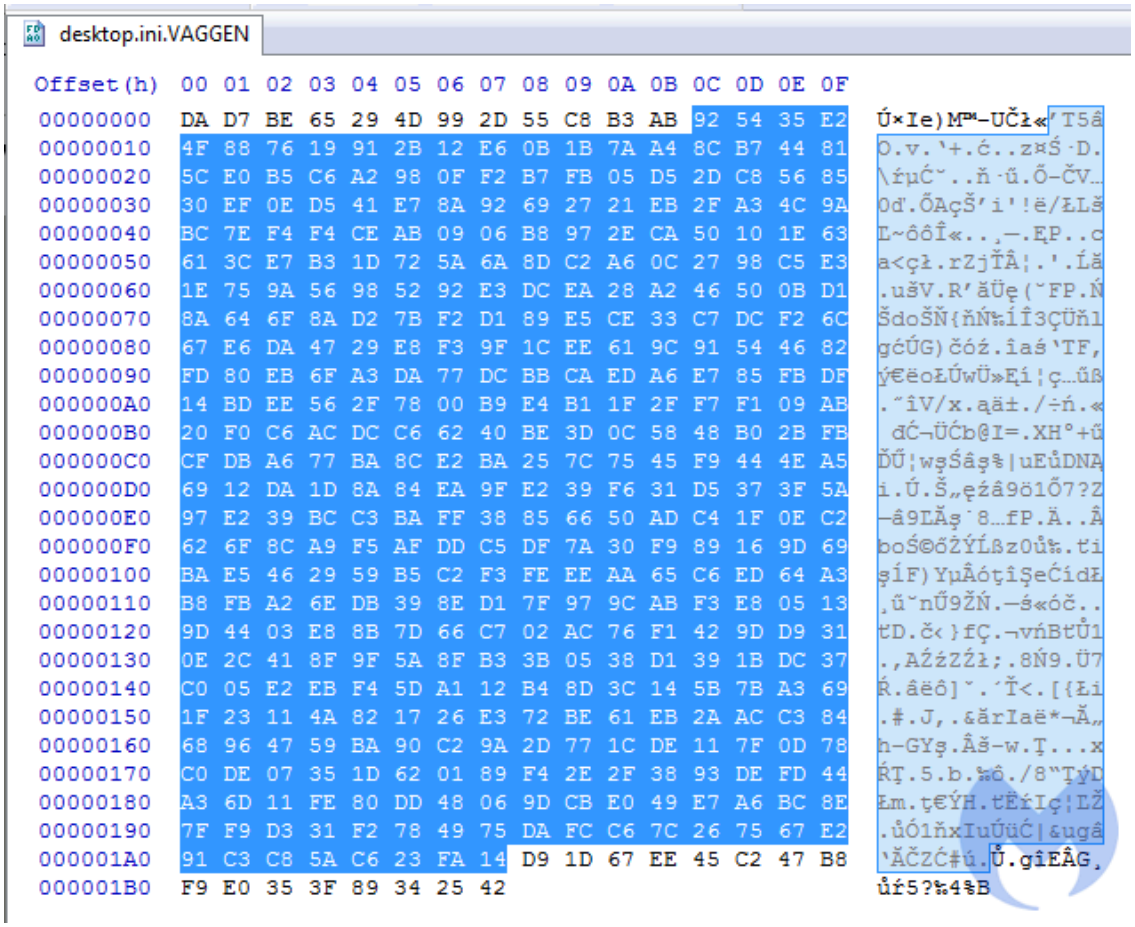
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

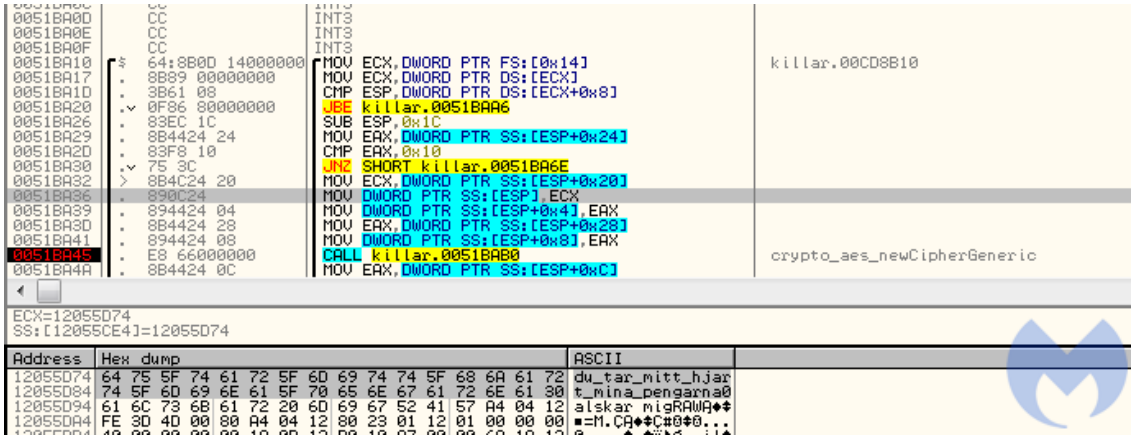


The content of the output file (with .VAGGEN extension) contains:

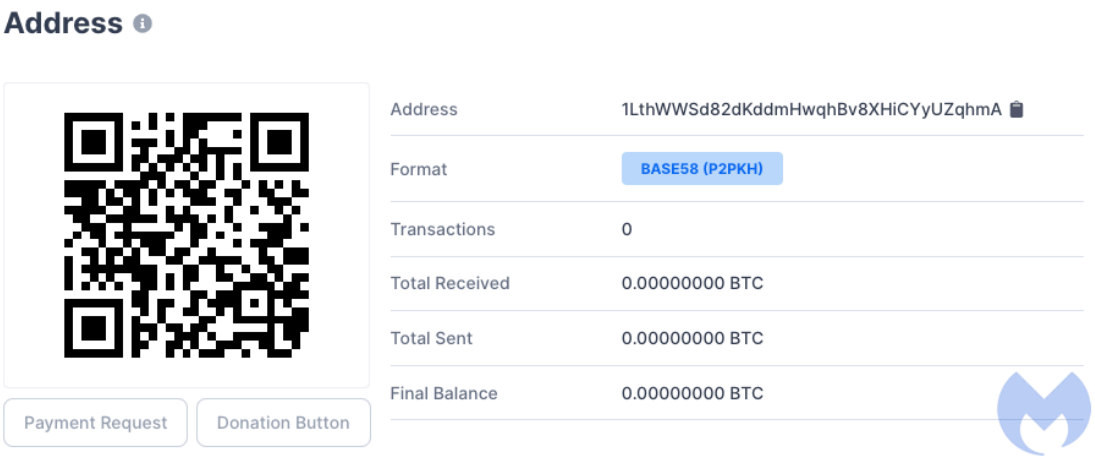
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

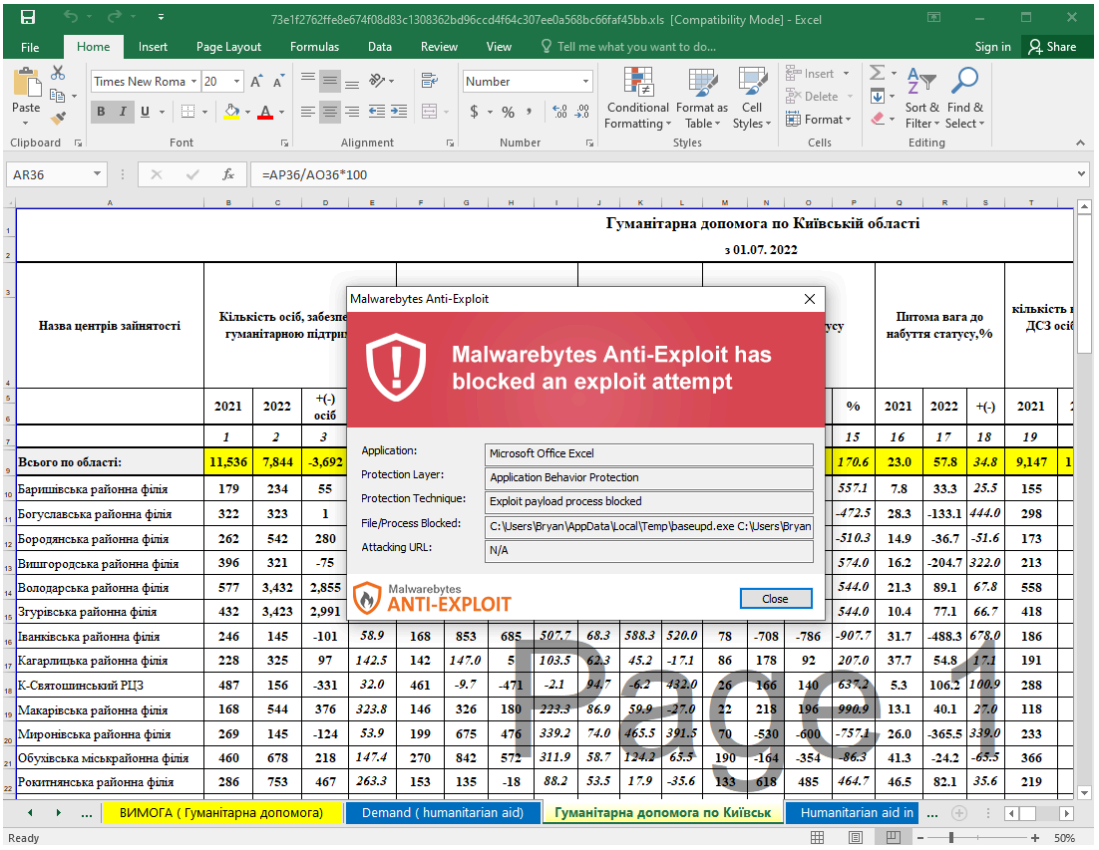
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

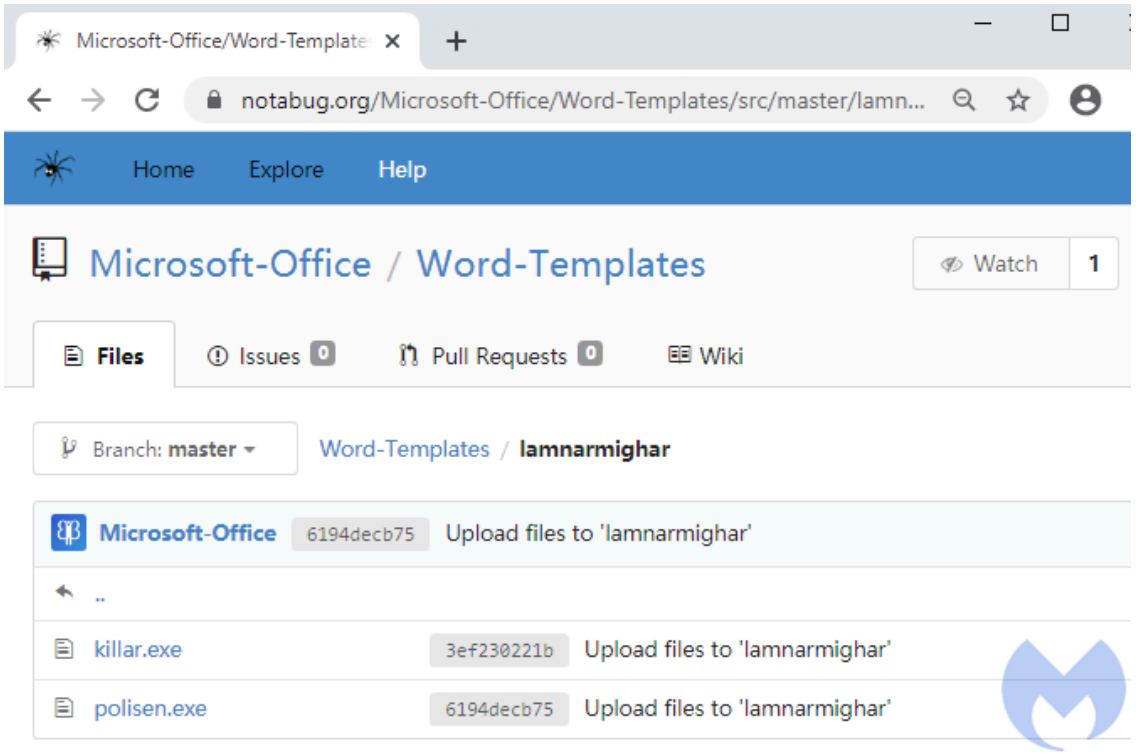
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



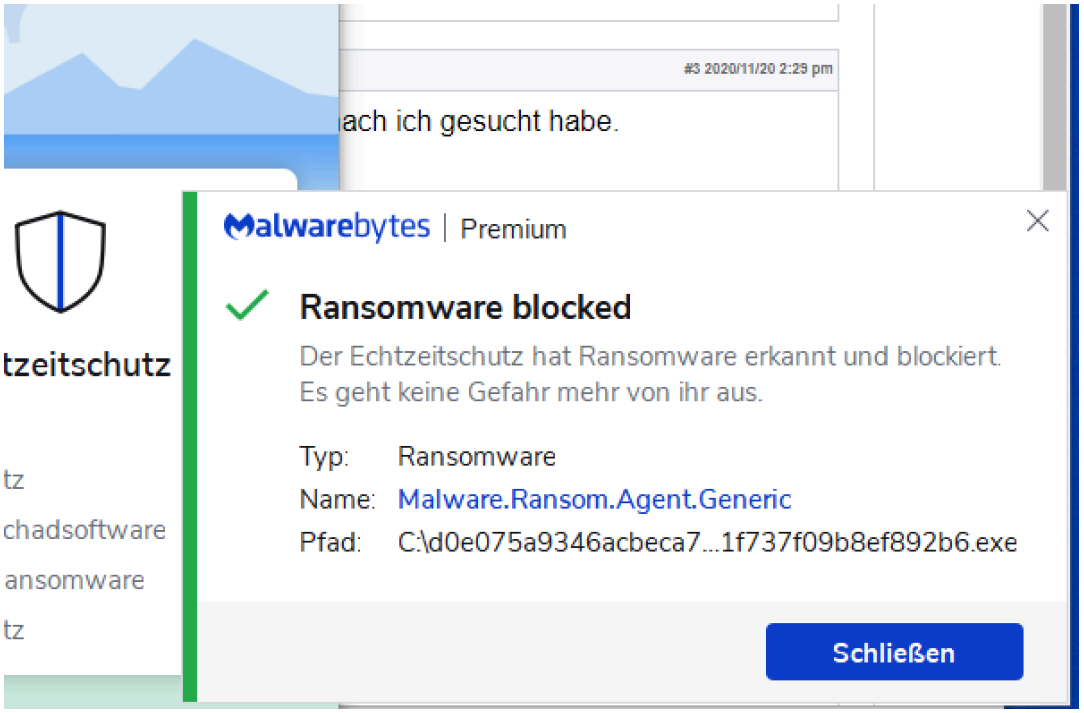
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as ‘main_main’.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

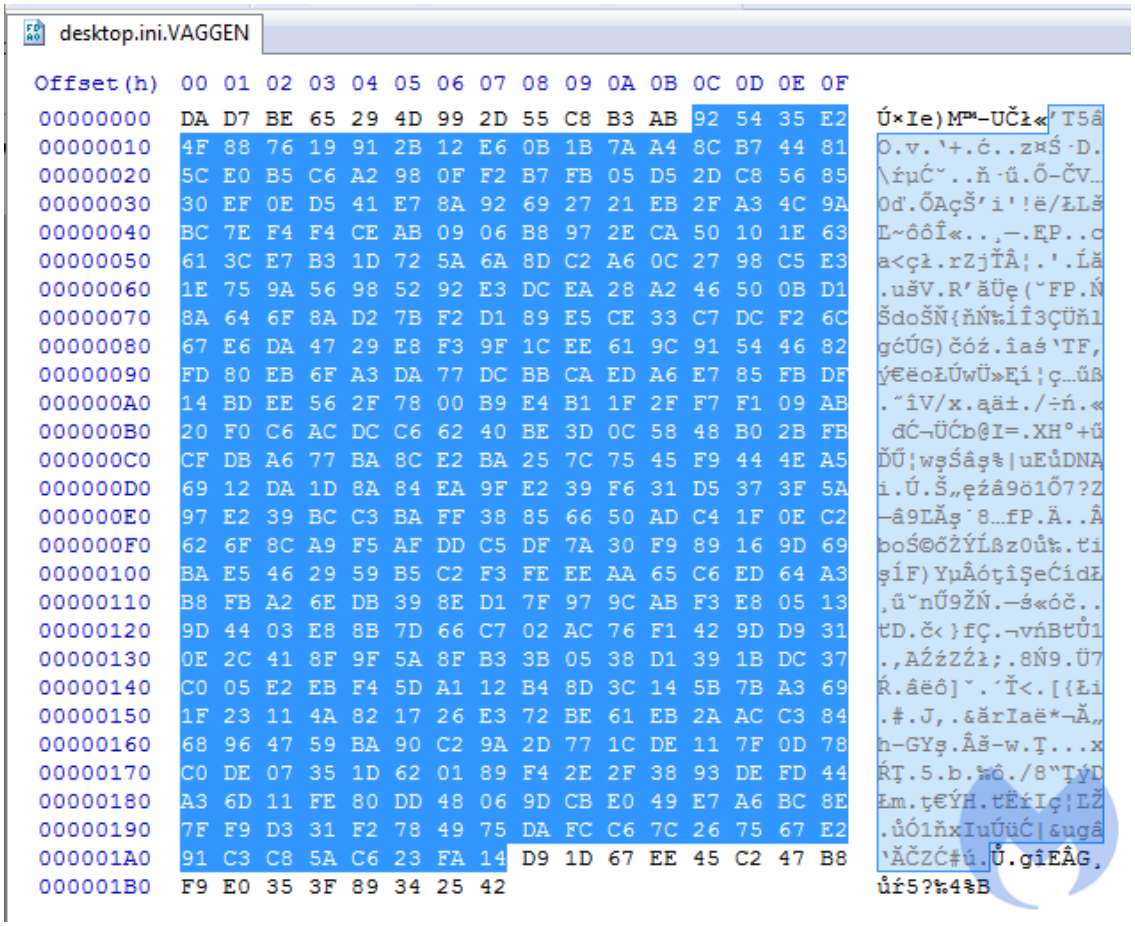
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

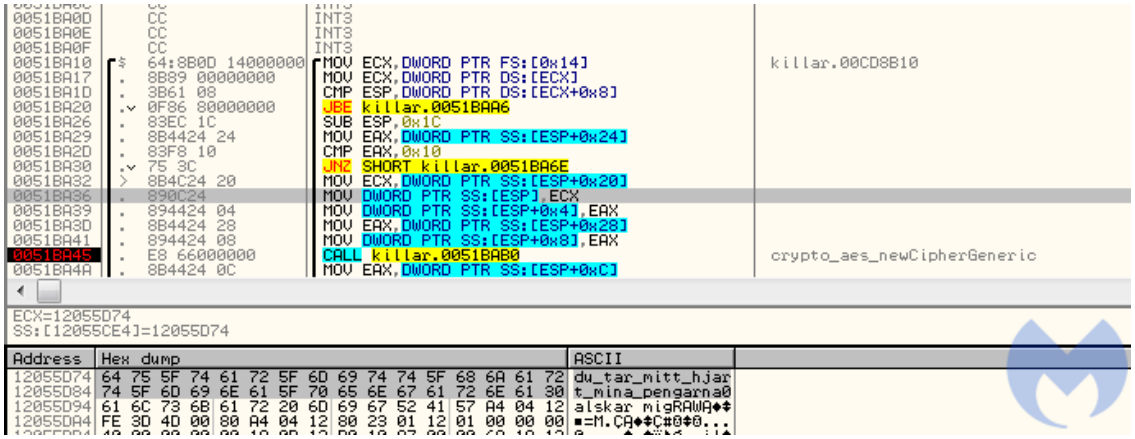


The content of the output file (with .VAGGEN extension) contains:

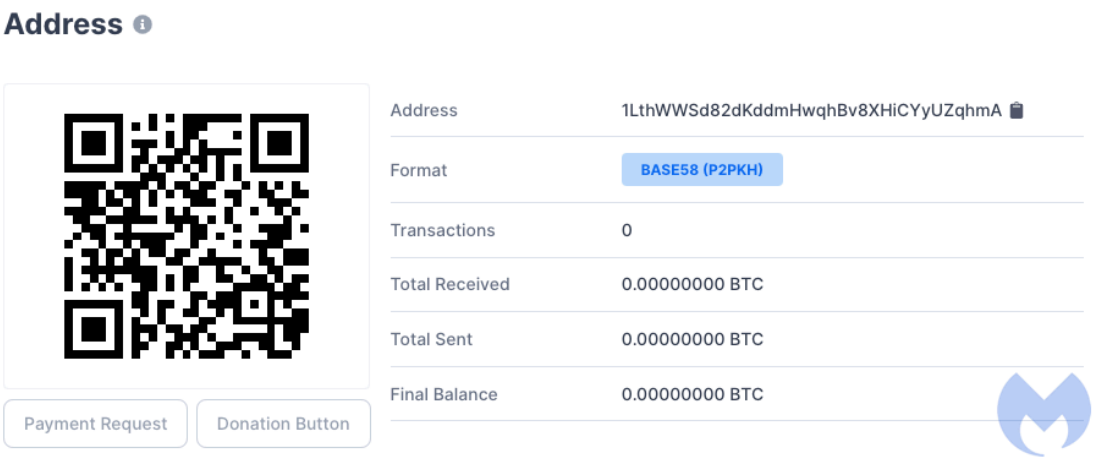
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

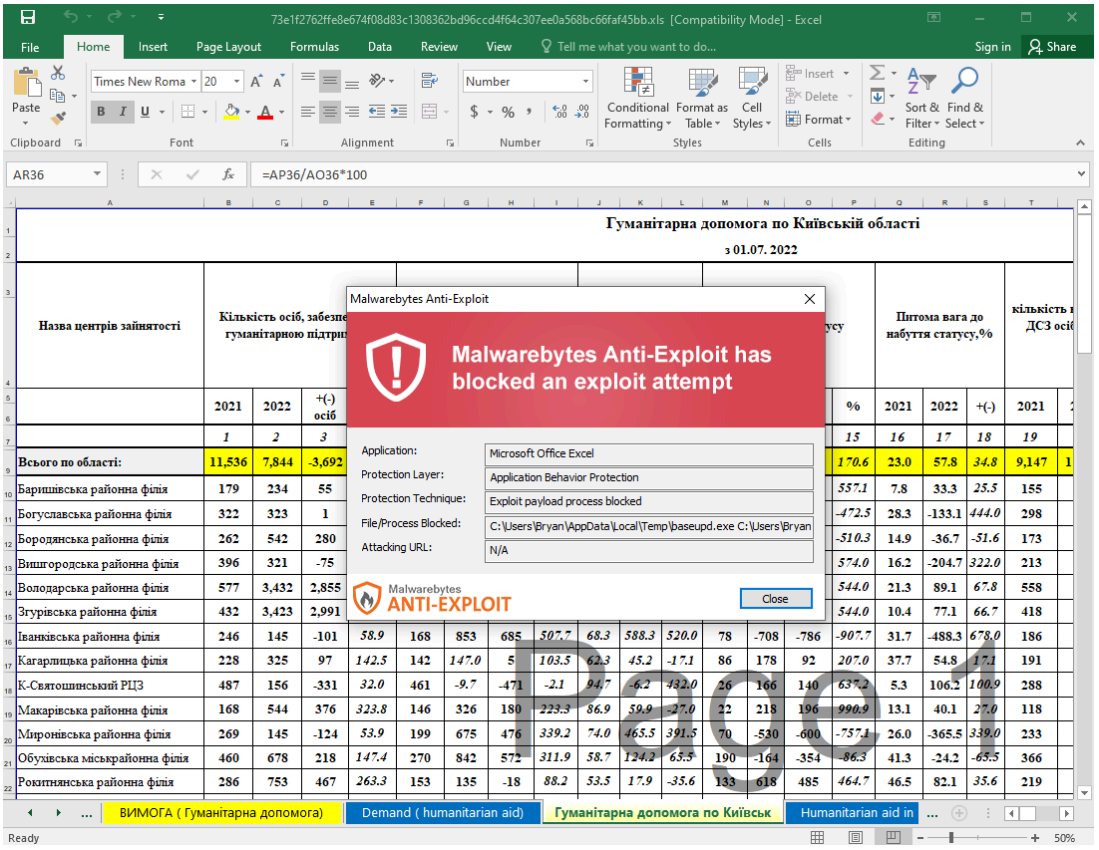
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

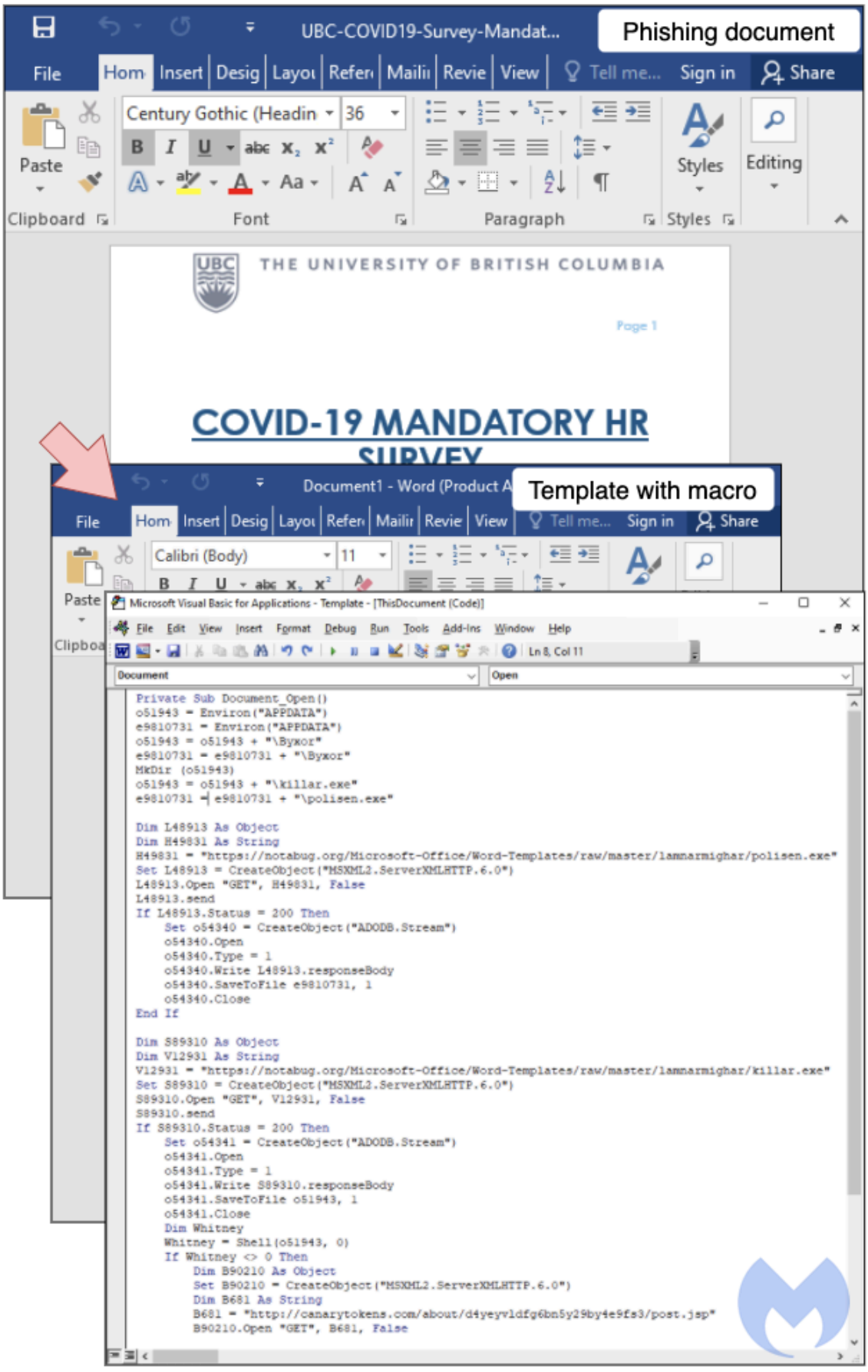
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

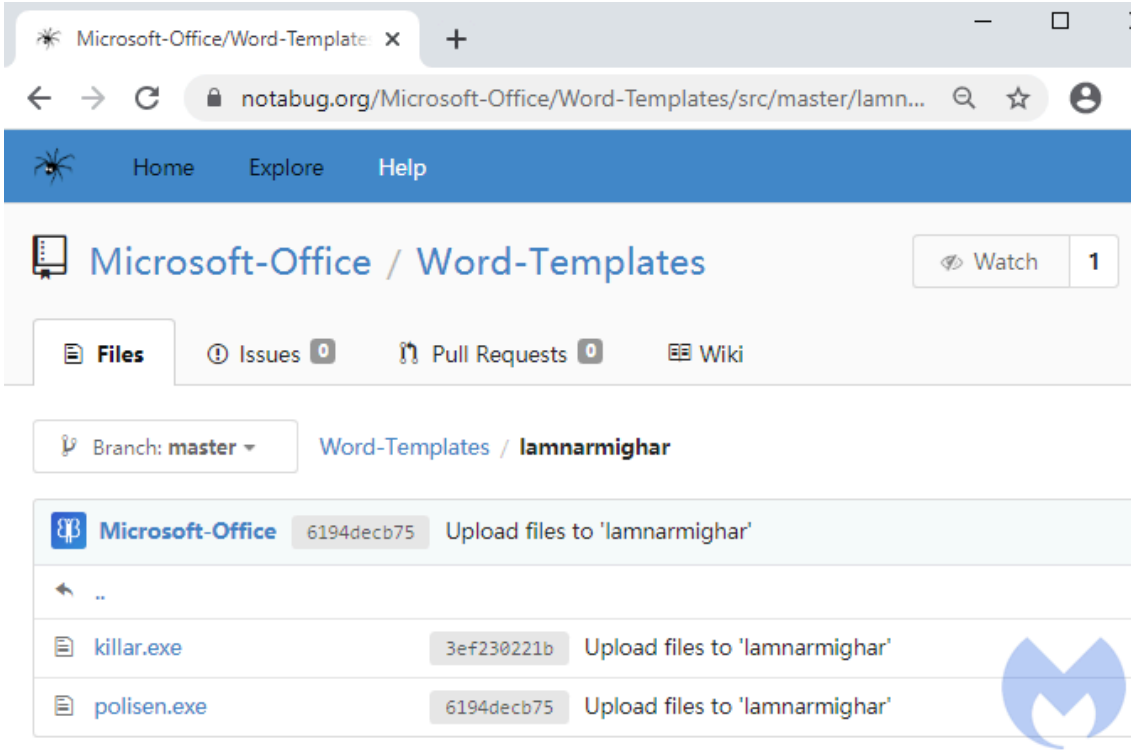
Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



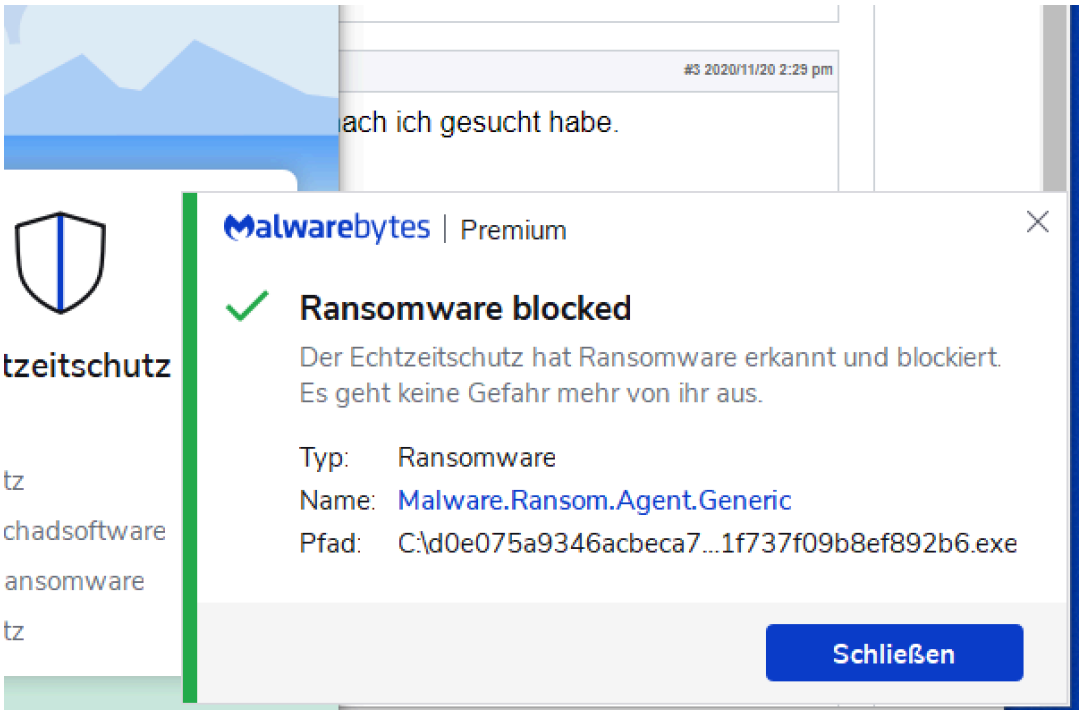
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

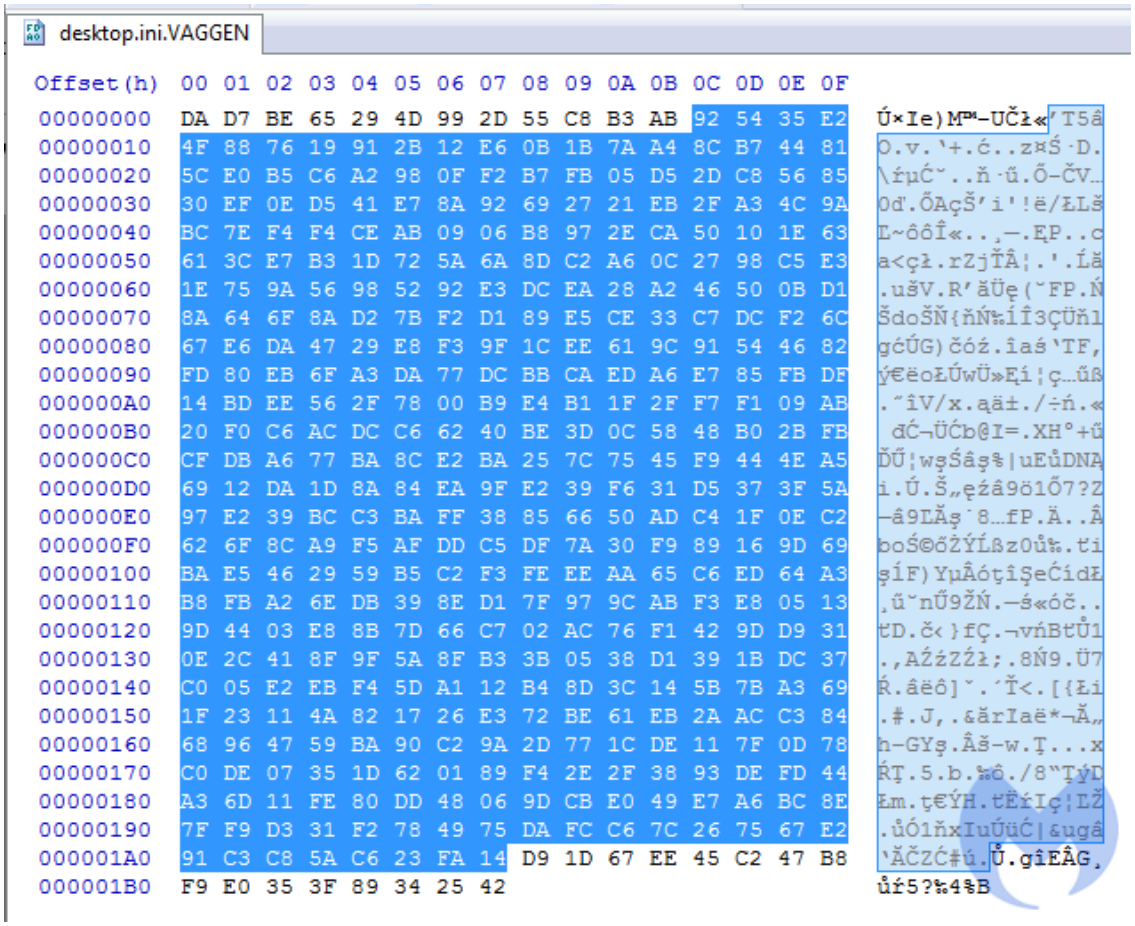
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

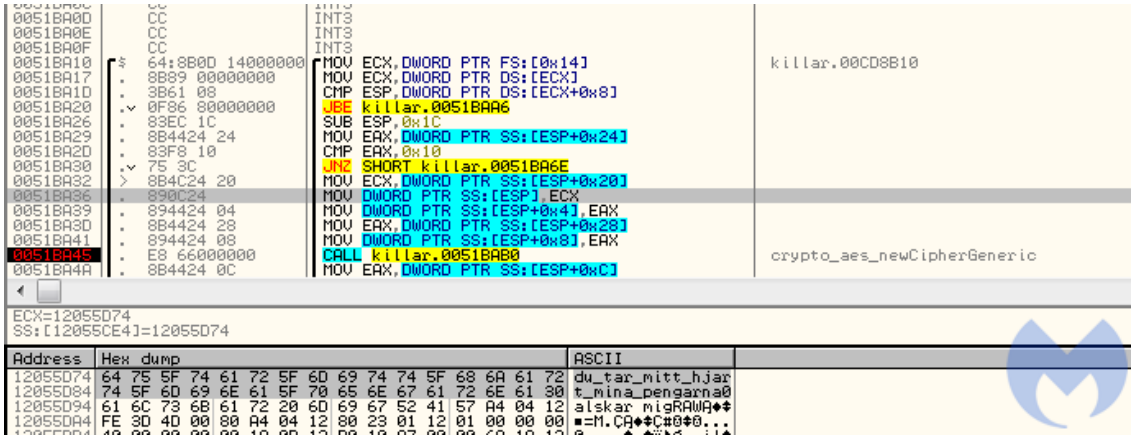


The content of the output file (with .VAGGEN extension) contains:

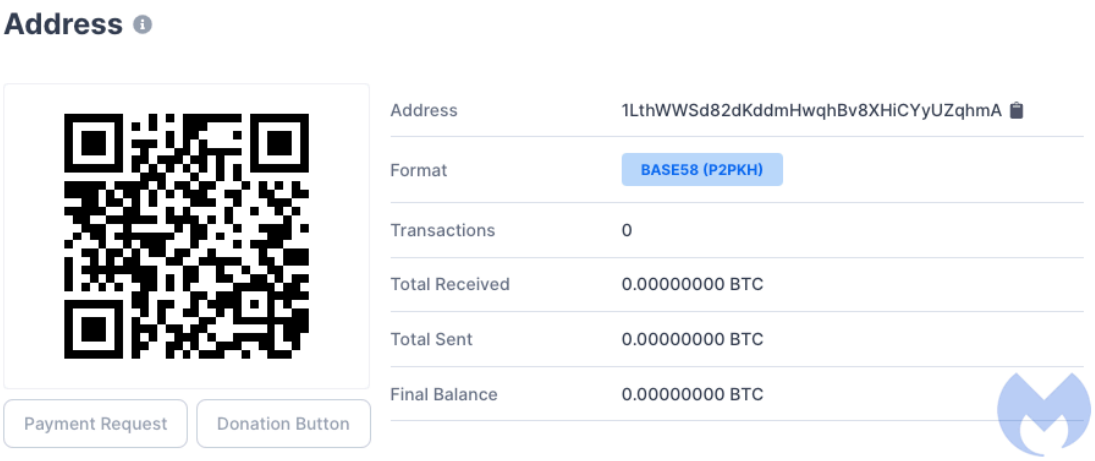
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

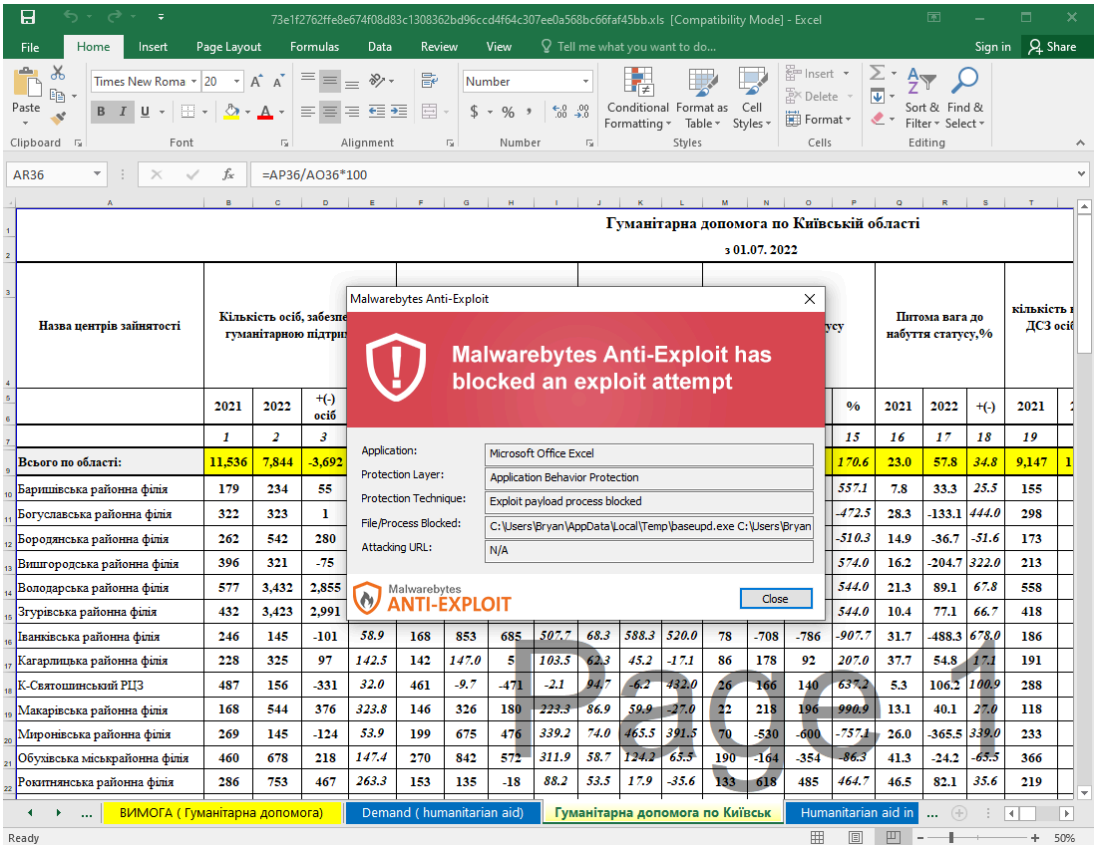
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

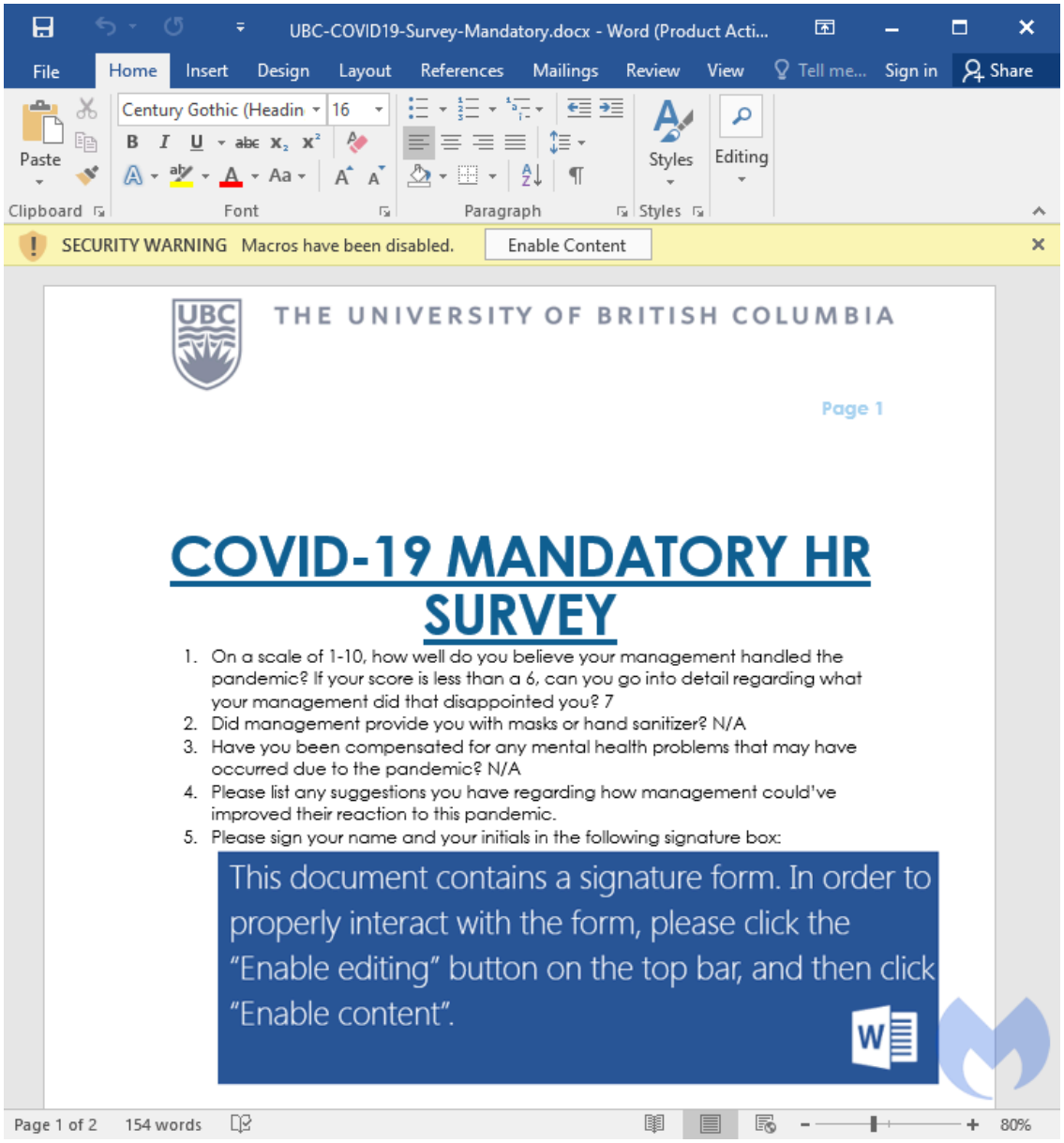
canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

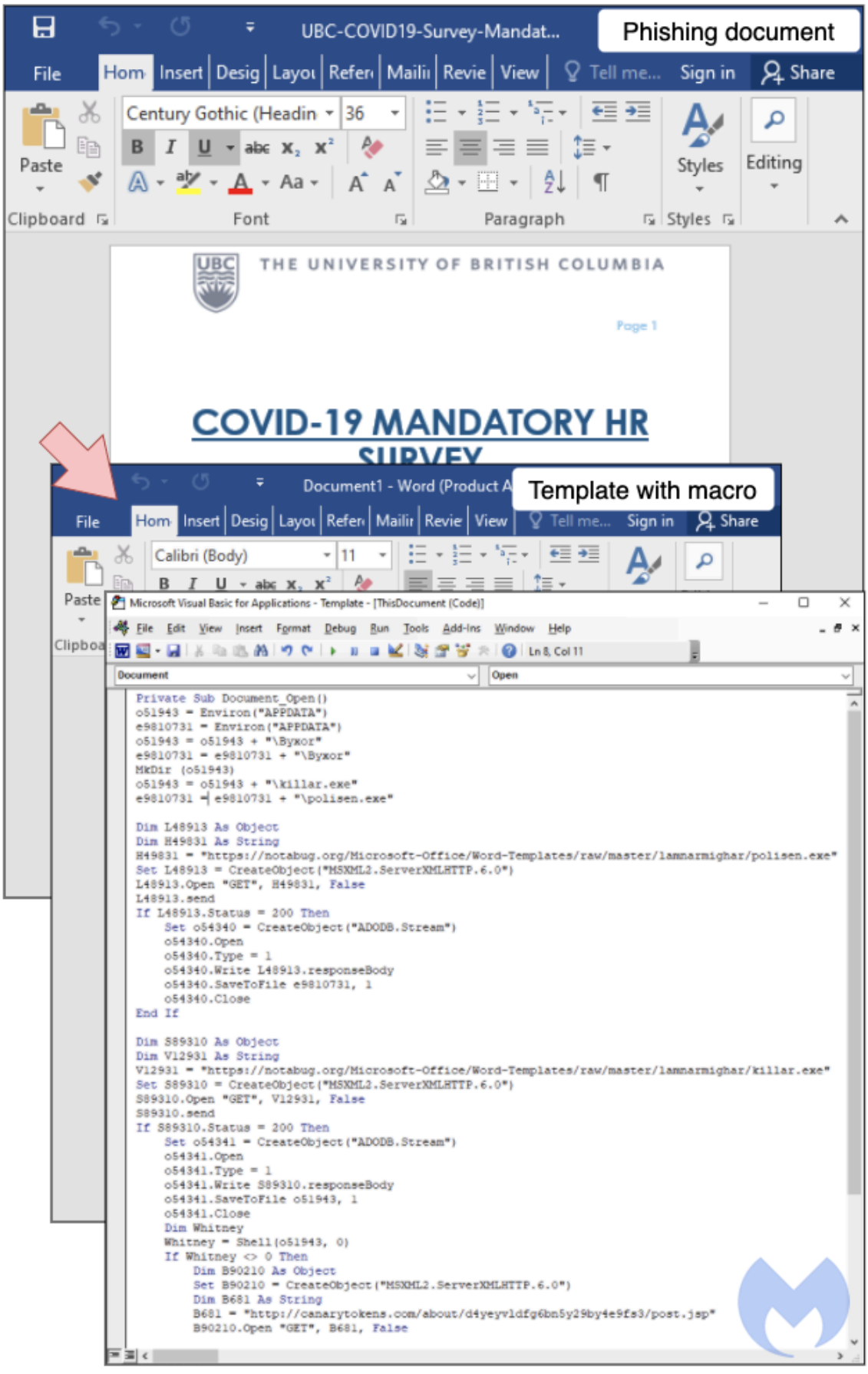
Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

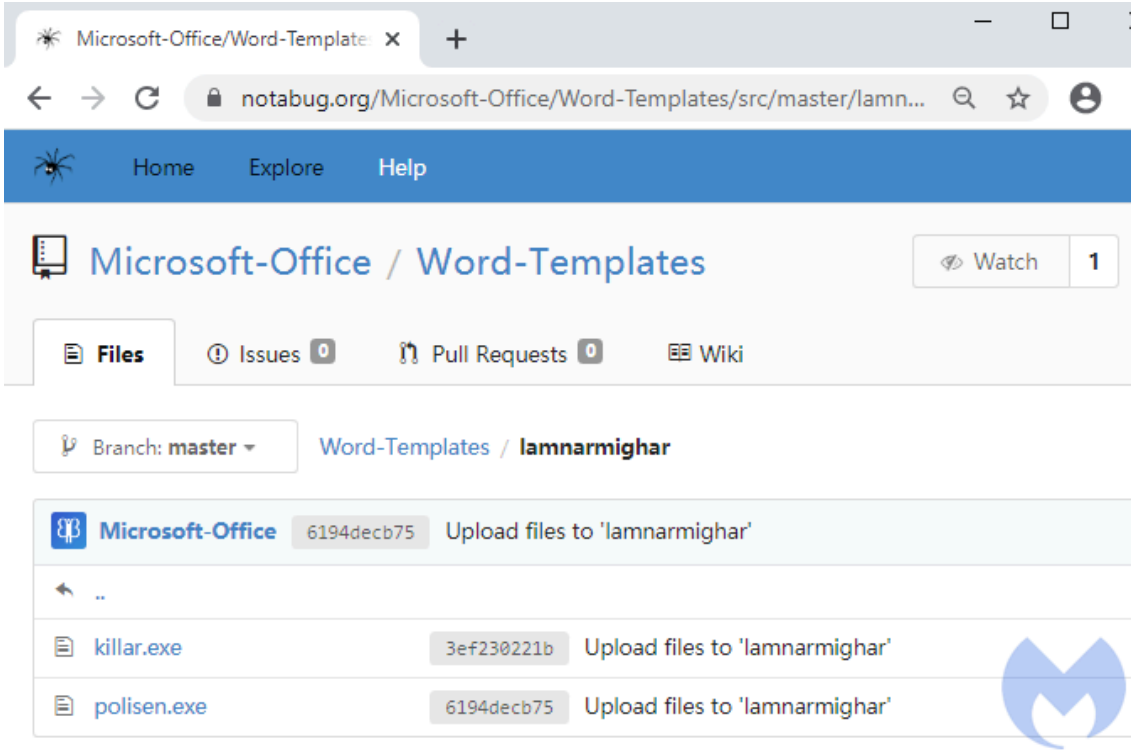
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



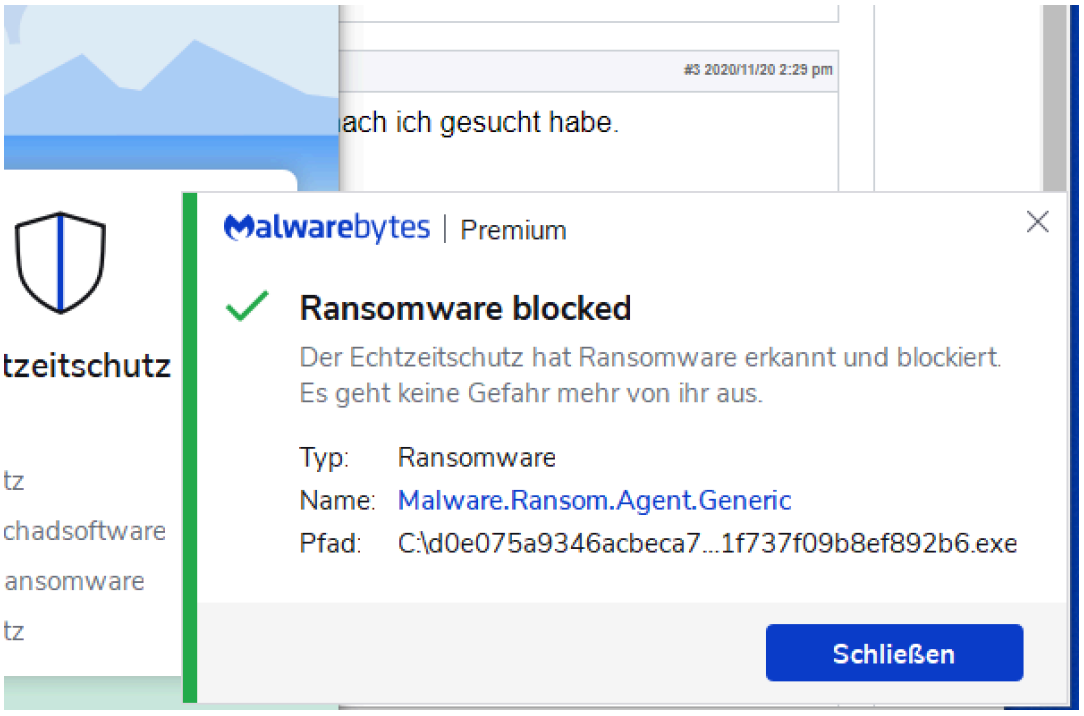
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

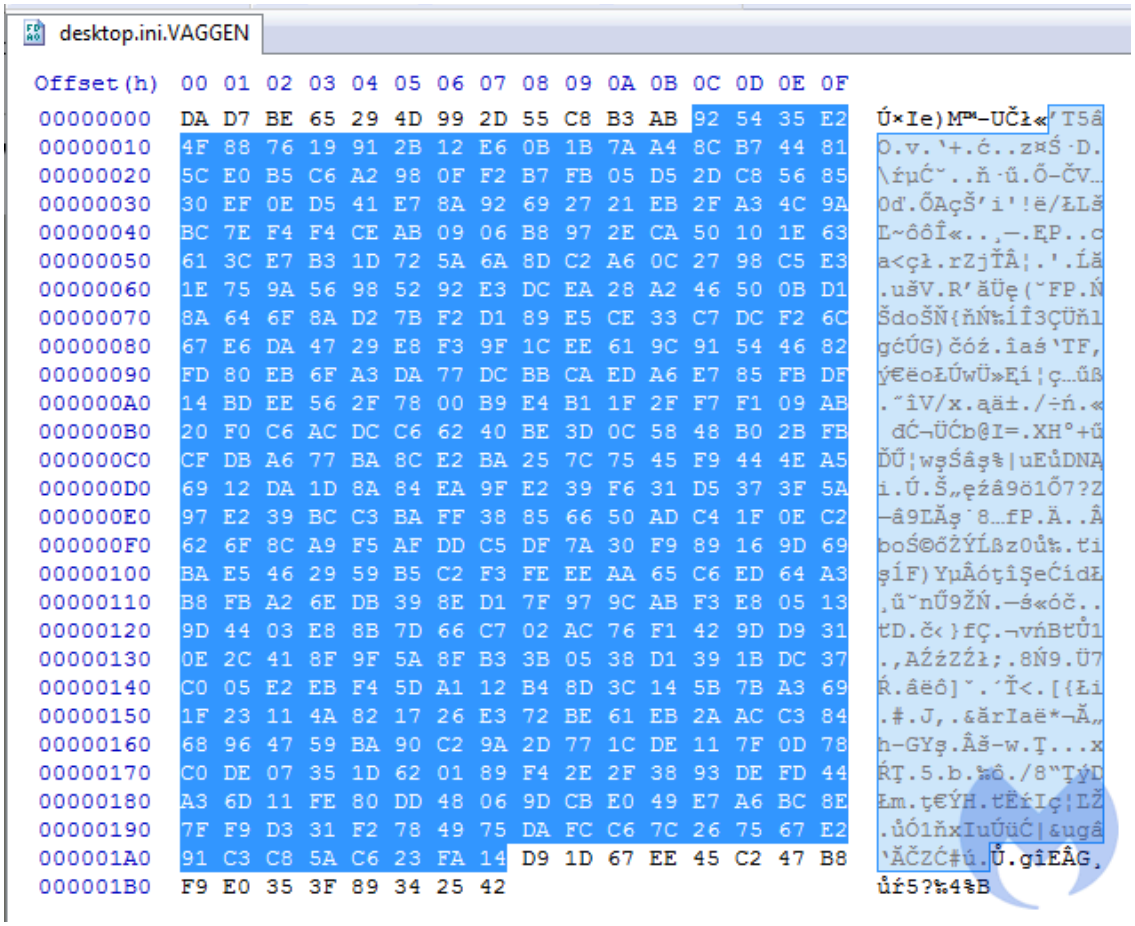
```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

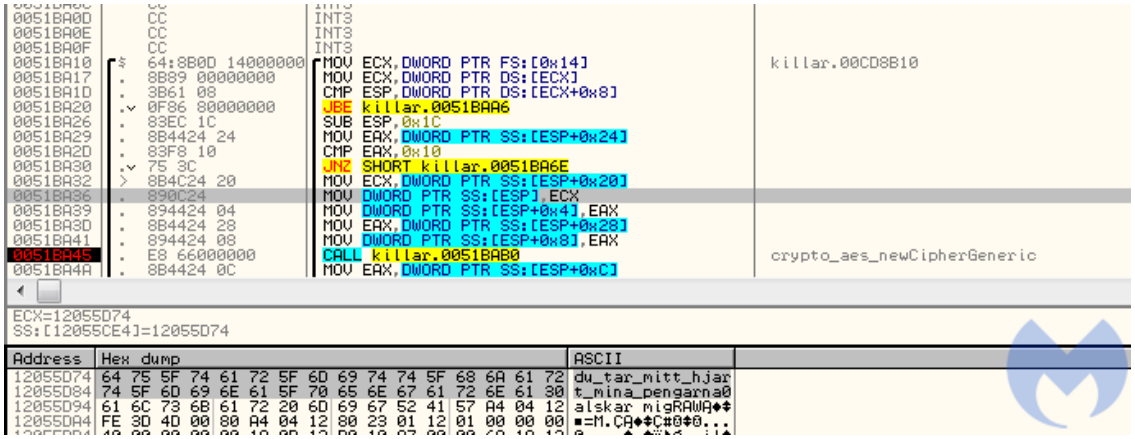


The content of the output file (with .VAGGEN extension) contains:

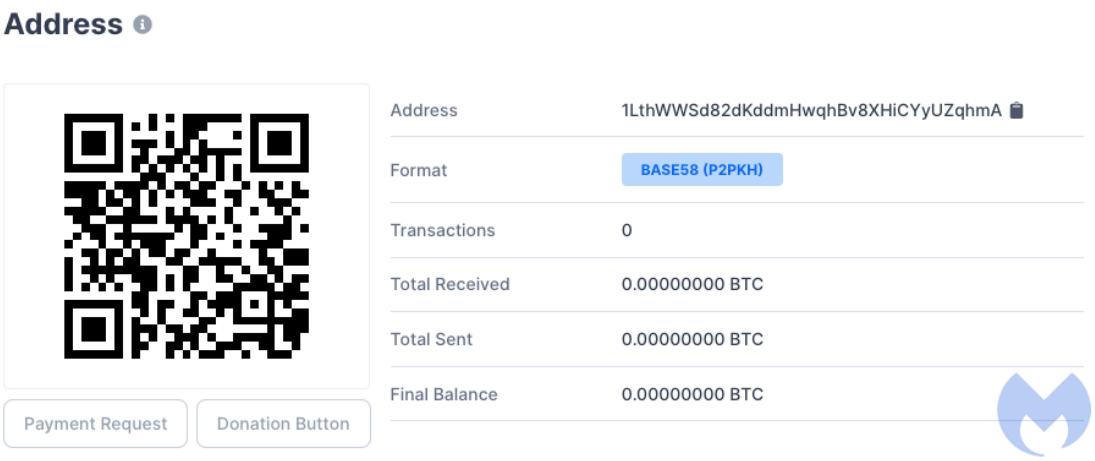
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

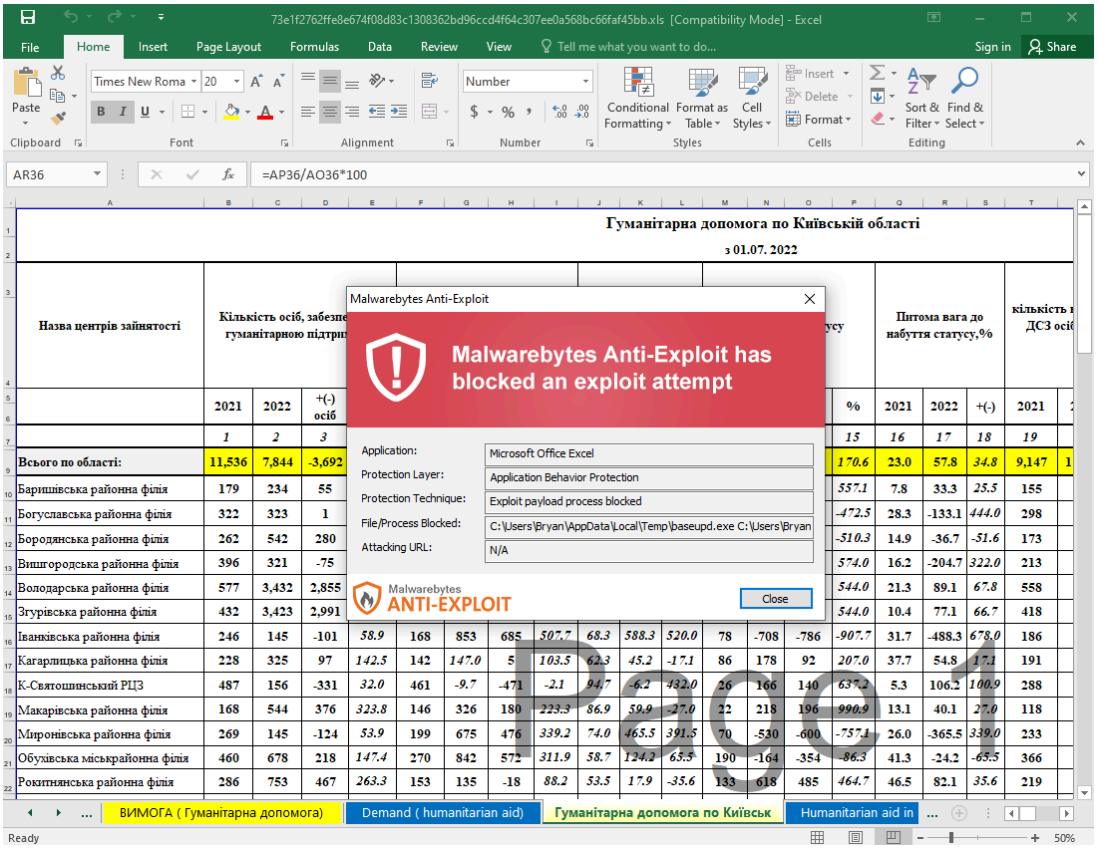
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

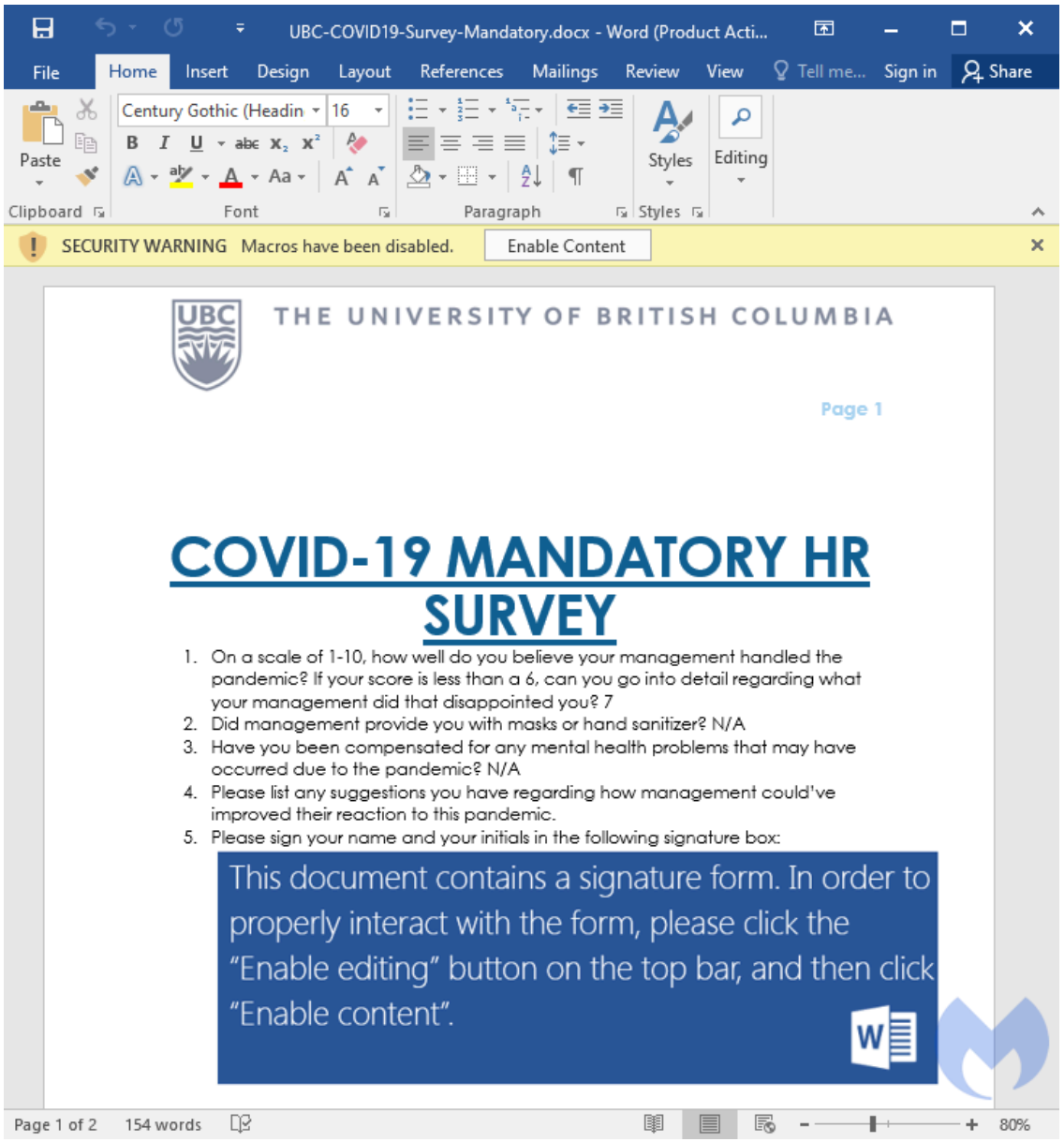
Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees

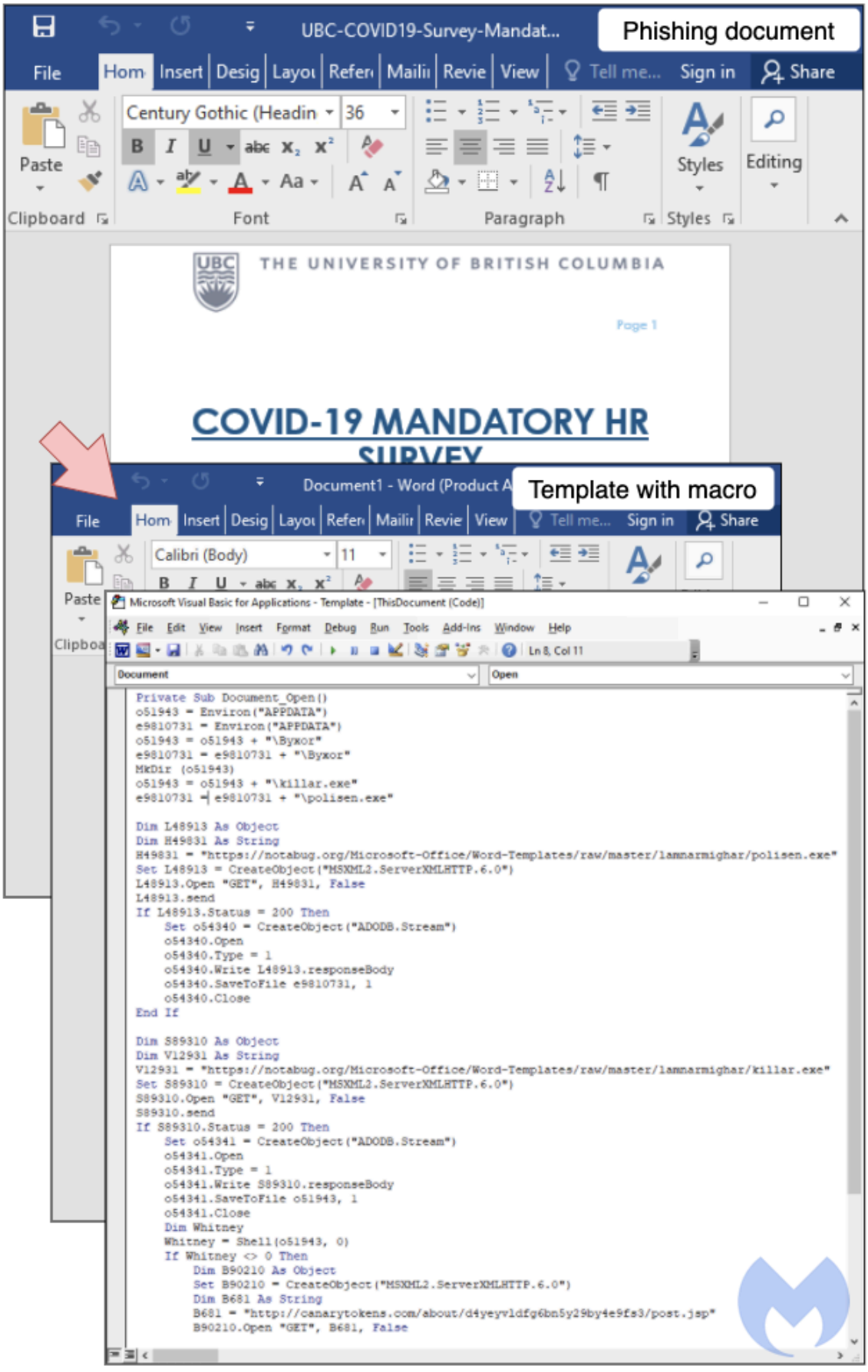
> who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

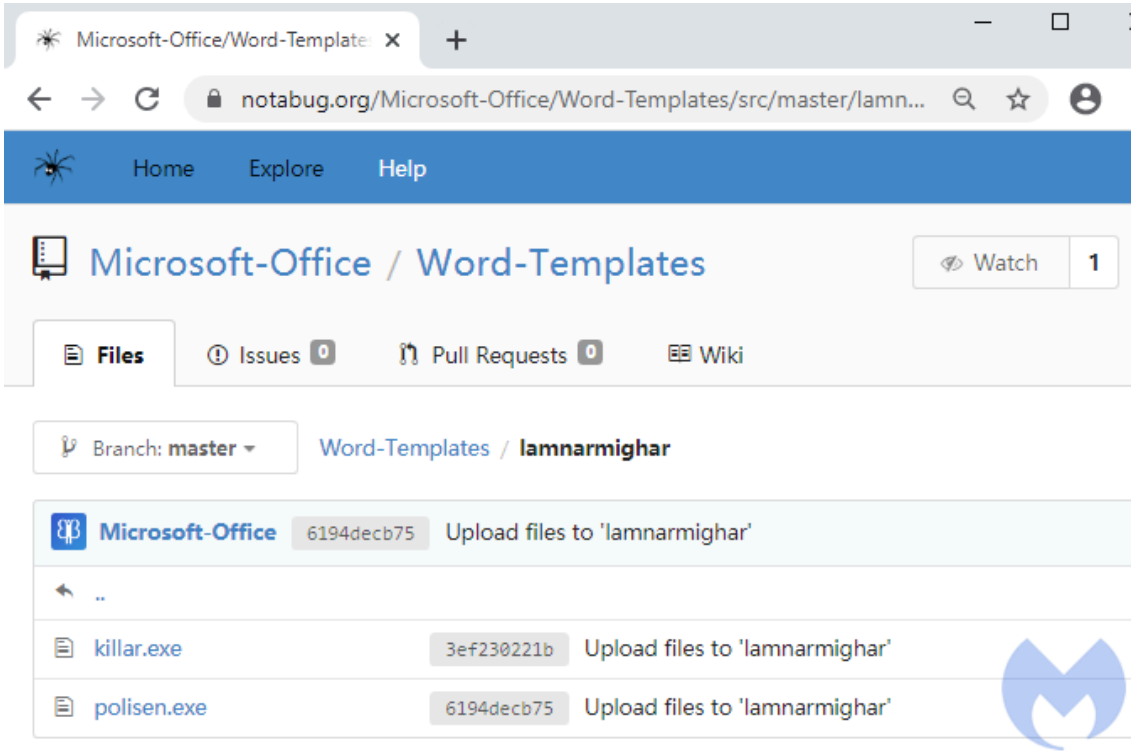
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



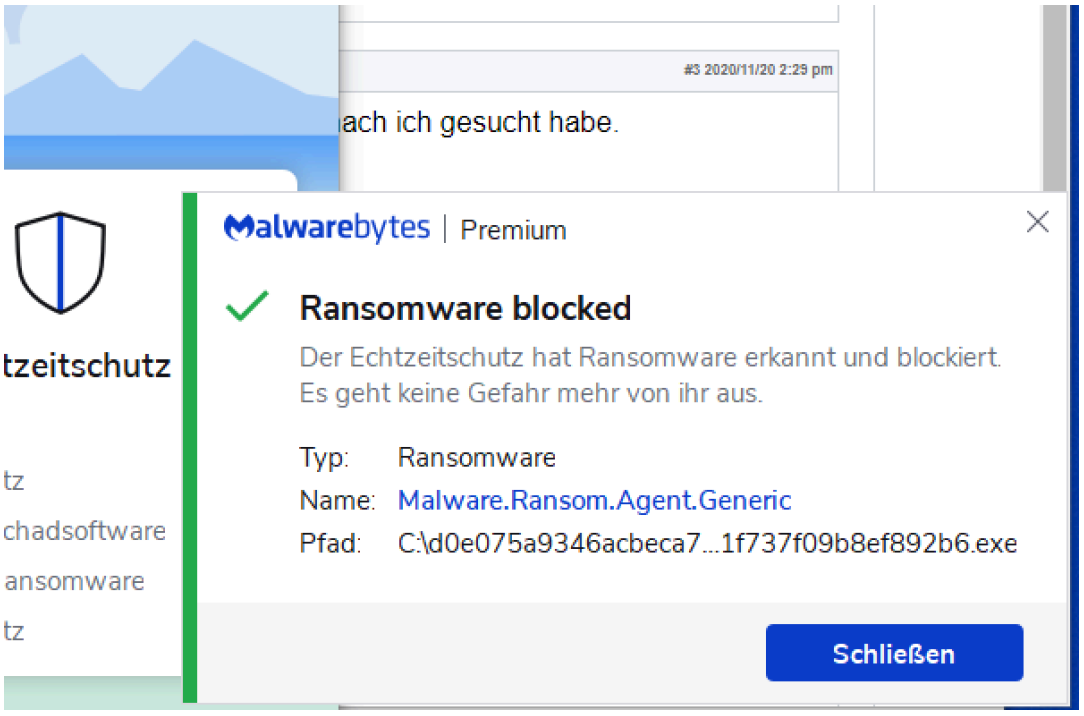
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-Bh]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```

Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

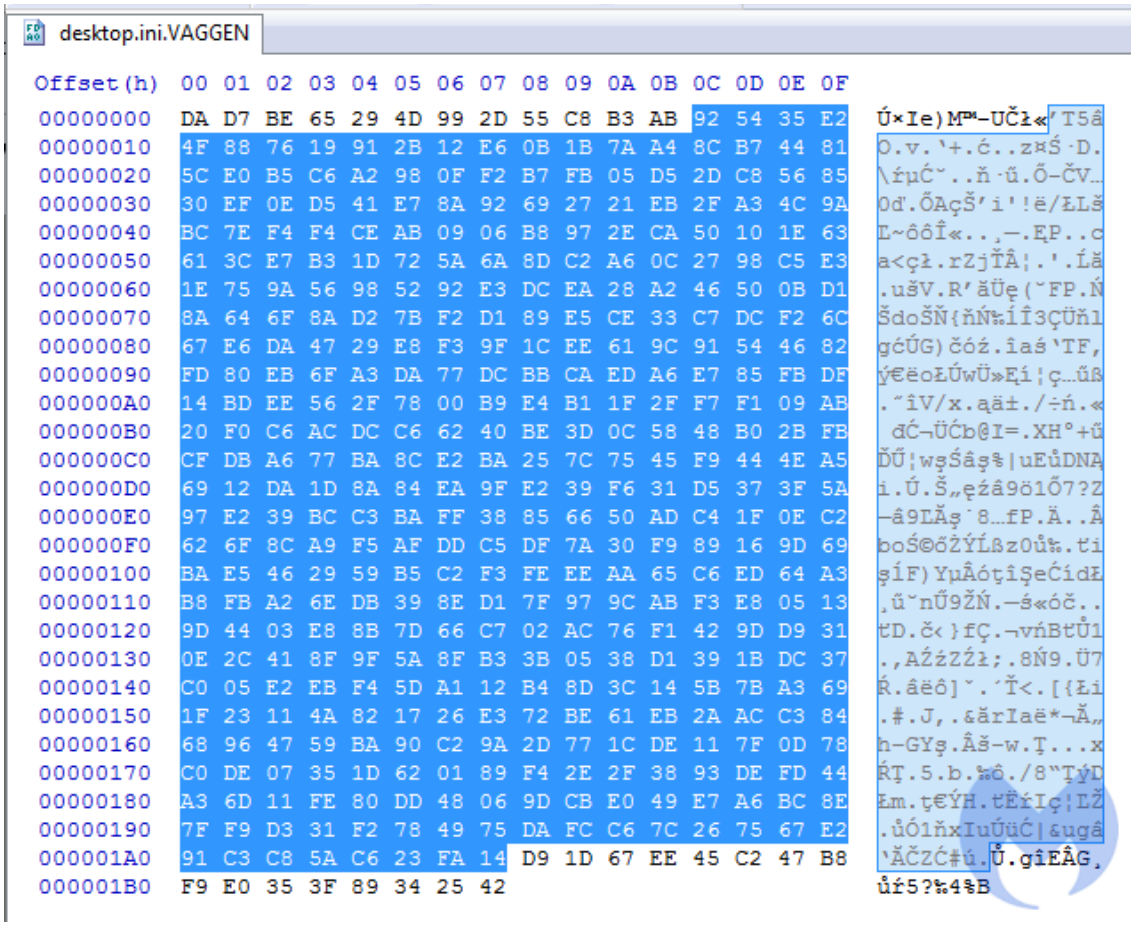
The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.



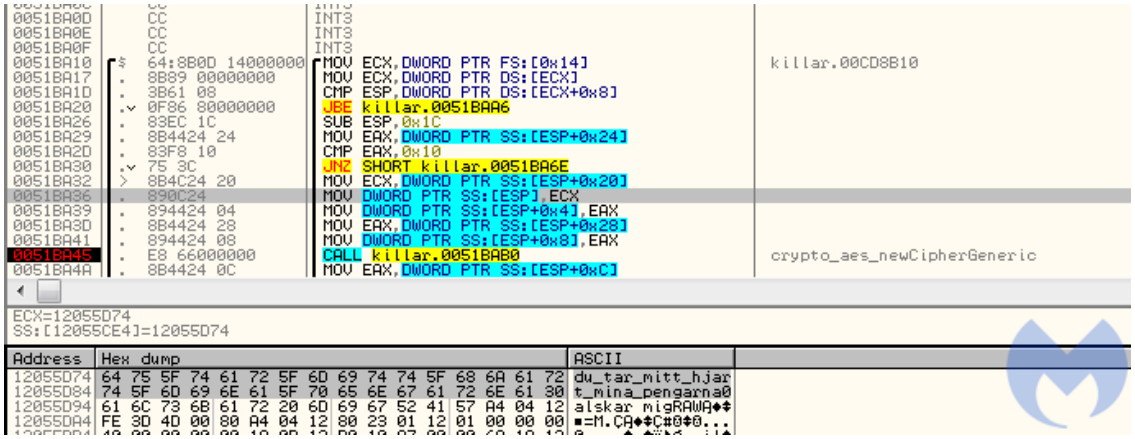
```
008069AB loc_8069AB:
008069AB mov     eax, [esp+0D0h+var_84]
008069AF mov     eax, [eax+1Ch]
008069B2 mov     ecx, [esp+0D0h+crypto_context]
008069B9 mov     [esp+0D0h+var_D0], ecx
008069BC mov     ecx, [esp+0D0h+nonce]
008069C3 mov     [esp+0D0h+var_CC], ecx
008069C7 mov     edx, [esp+0D0h+nonce_size]
008069CB mov     [esp+0D0h+var_C8], edx
008069CF mov     [esp+0D0h+var_C4], edx
008069D3 mov     [esp+0D0h+var_C0], ecx
008069D7 mov     [esp+0D0h+var_BC], edx
008069DB mov     [esp+0D0h+var_B8], edx
008069DF mov     ecx, [esp+0D0h+input_buffer]
008069E6 mov     [esp+0D0h+var_B4], ecx
008069EA mov     ecx, [esp+0D0h+key_size]
008069EE mov     [esp+0D0h+var_B0], ecx
008069F2 mov     ecx, [esp+0D0h+var_88]
008069F6 mov     [esp+0D0h+var_AC], ecx
008069FA mov     [esp+0D0h+var_A8], 0
00806A02 mov     [esp+0D0h+var_A4], 0
00806A0A mov     [esp+0D0h+var_A0], 0
00806A12 call    eax             ; 5192a0 -> crypto_cipher__ptr_gcm_Seal
00806A14 mov     eax, [esp+0D0h+var_9C]
00806A18 mov     ecx, [esp+0D0h+var_98]
00806A1C mov     edx, [esp+0D0h+var_94]
00806A20 mov     ebx, [esp+0D0h+var_20]
00806A27 mov     [esp+0D0h+var_D0], ebx
00806A2A mov     [esp+0D0h+var_CC], eax
00806A2E mov     [esp+0D0h+var_C8], ecx
00806A32 mov     [esp+0D0h+var_C4], edx
00806A36 call    os__ptr_File_Write ; os__ptr_File_Write
00806A3B mov     eax, [esp+0D0h+var_20]
00806A42 test    eax, eax
00806A44 jnz     short loc_806A67
```

The content of the output file (with .VAGGEN extension) contains:

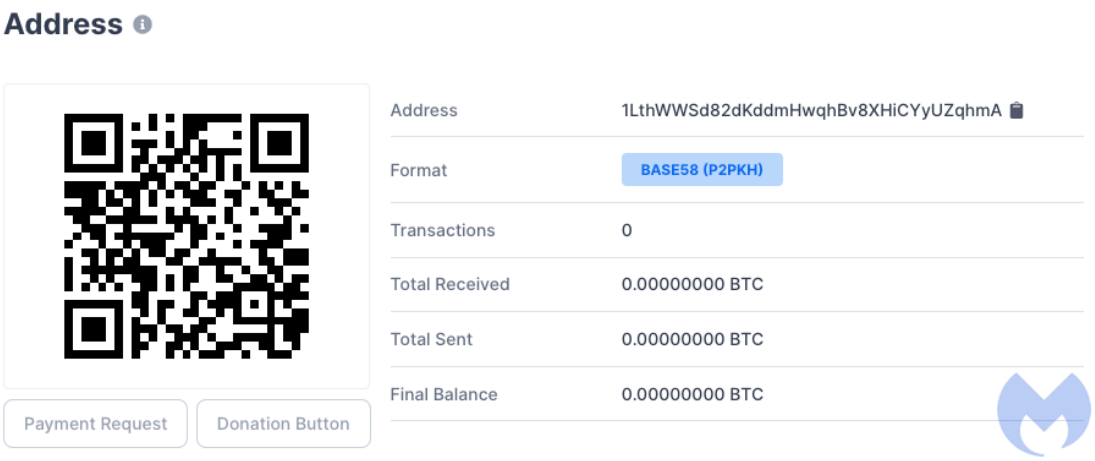
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

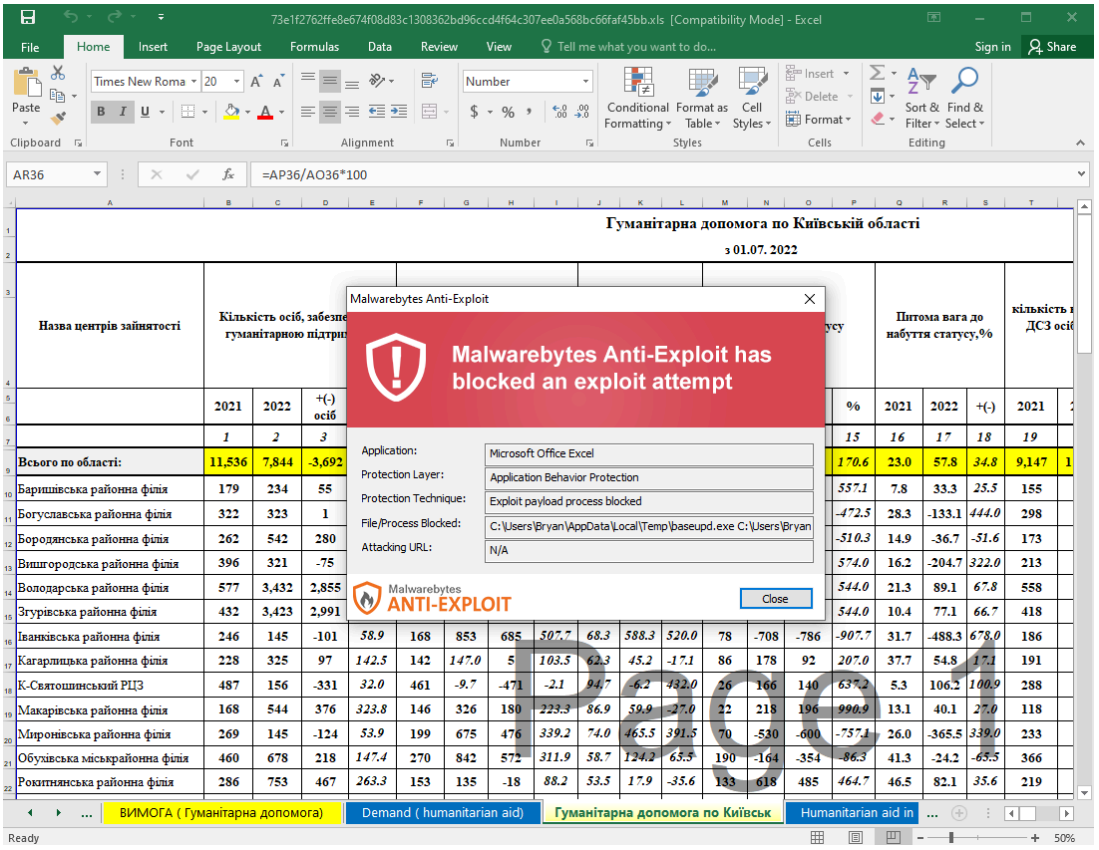
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a

This post was authored by Jérôme Segura with contributions from Hossein Jazi, Hasherezade and Marcelo Rivero.

In recent weeks, we've observed a number of phishing attacks against universities worldwide which we attributed to the Silent Librarian APT group. On October 19, we identified a new phishing document targeting staff at the University of British Columbia (UBC) with a fake COVID-19 survey.

However, this attack and motives are different than the ones previously documented. The survey is a malicious Word document whose purpose is to download ransomware and extort victims to recover their encrypted files.

On discovery, we got in touch with UBC to report our findings. They were already aware of this phishing campaign and were kind enough to share more information with us about the incident. Ultimately, this attack was not successful due to the rapid response of the UBC cybersecurity team.

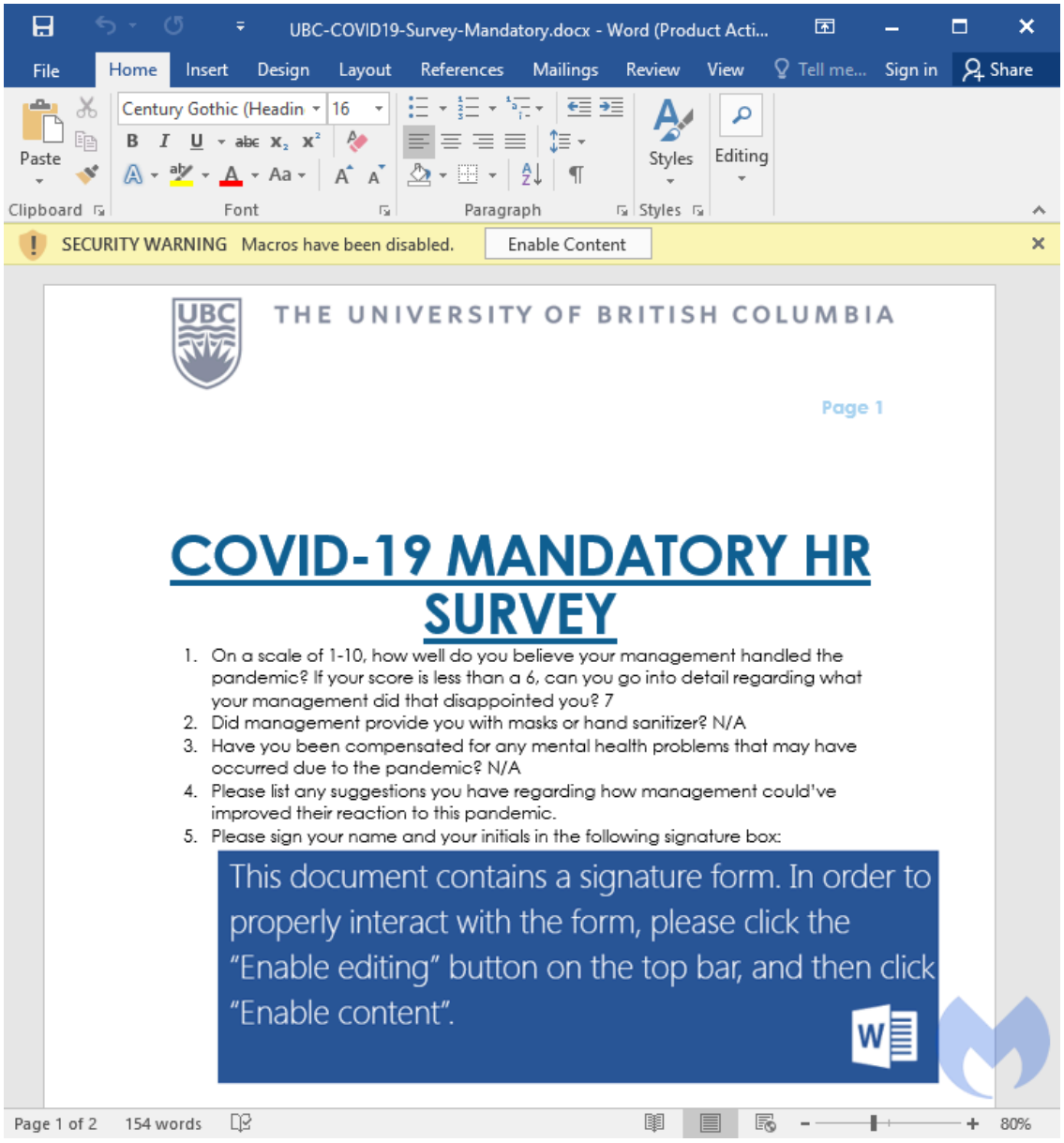
Mandatory COVID-19 survey distributed to targeted recipients

The attacker created an email address with the mailpoof.com service in order to register accounts with Box.net and DropBox. Rather than directly sending the fake survey via email, the attacker uploaded the document onto Box and DropBox and used the share functionality from these platforms to distribute it.

This was probably done to evade spam and phishing filters that would have blocked messages coming from a newly registered email address with a low reputation. In comparison, it is much more difficult to detect spam from file sharing services without creating a number of false positives.

The attacker claimed to be a manager and added the following comment in the file sharing invitation (shared with us by UBC):

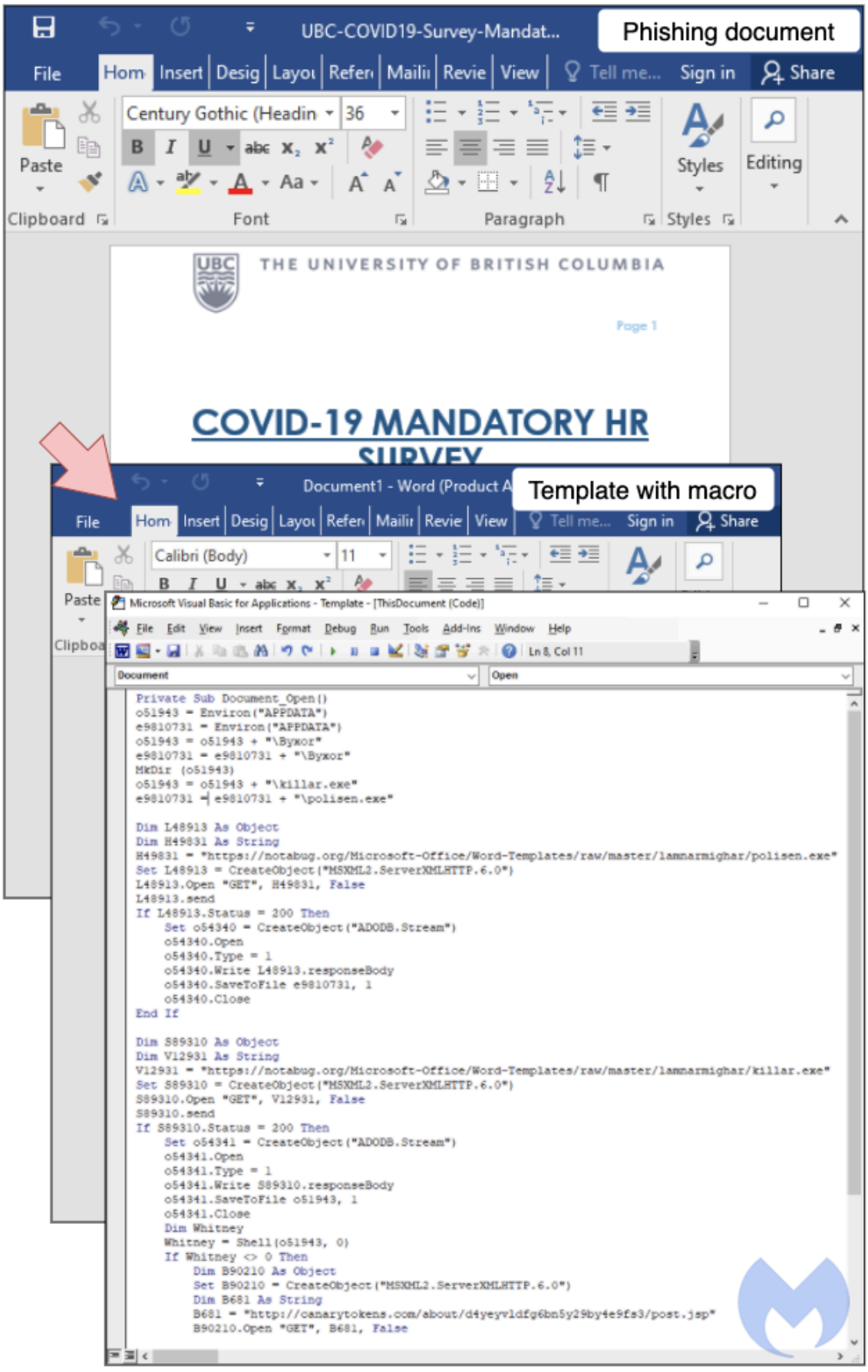
> Good evening gals and guys! [redacted] here, [redacted] manager for [redacted]. I am sharing a mandatory survey with you that must be completed by Monday. It asks a few questions about how you believe our company responded to the pandemic regarding remote working and much more. Please fill it out ASAP!
>
> You will also find a form at the end that you can fill out if you need any necessities! Necessities include: gloves, hand sanitizer, masks, or disinfectant spray. We will be providing it to those employees who fill out the form for free! Simply sign your initials and put what you need as well as the quantity! In advance, we appreciate your feedback! Thanks all! Stay strong! I understand times like this can be difficult!



According to UBC, less than a hundred people within a specific department received the link to access the shared document. A Box or Dropbox account was required in order to download the file since it was shared privately, instead of publicly. This may have been an effort to evade detection or perhaps the attacker expected the target organization to already be using one of these two sharing services.

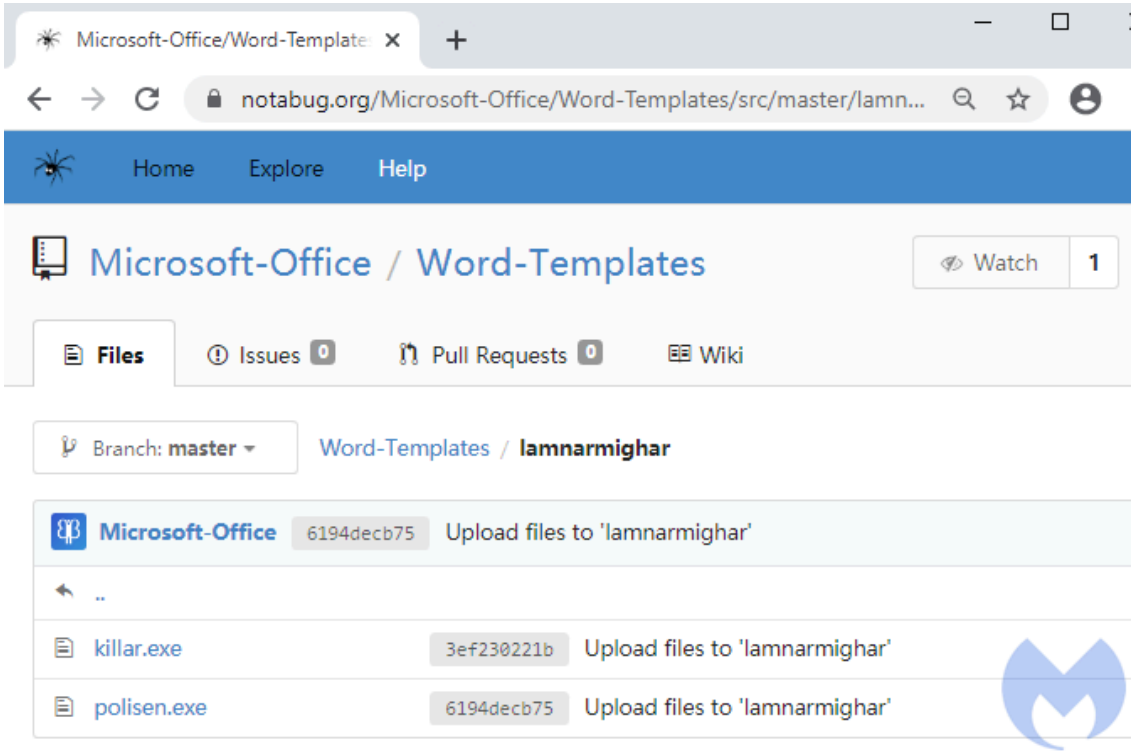
Phishing document analysis

The phishing document uses template injection to download and execute a remote template (template.dotm) weaponized with a malicious macro. That file was uploaded to a free code hosting website (notabug.org).



When the macro is executed, it does the following:

- Gets the *%APPDATA%* directory
- Creates the *Byxor* directory in %APPDATA%
- Downloads a file from the following url and writes it as *Polisen.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe*
- Downloads a file from the following url and writes it as *Killar.exe*
- *notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe*
- Calls shell function to execute *killar.exe*
- Checks the output of shell function and whether it was successful (return value would be task Id of executed application)
    - If successful, it sends a GET http request to: *canarytokens.com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp*
    - If it isn't successful, it sends a GET http request to: *canarytokens.com/articles/6dbbnd503z06qitej1sdzzcvv/index.html*



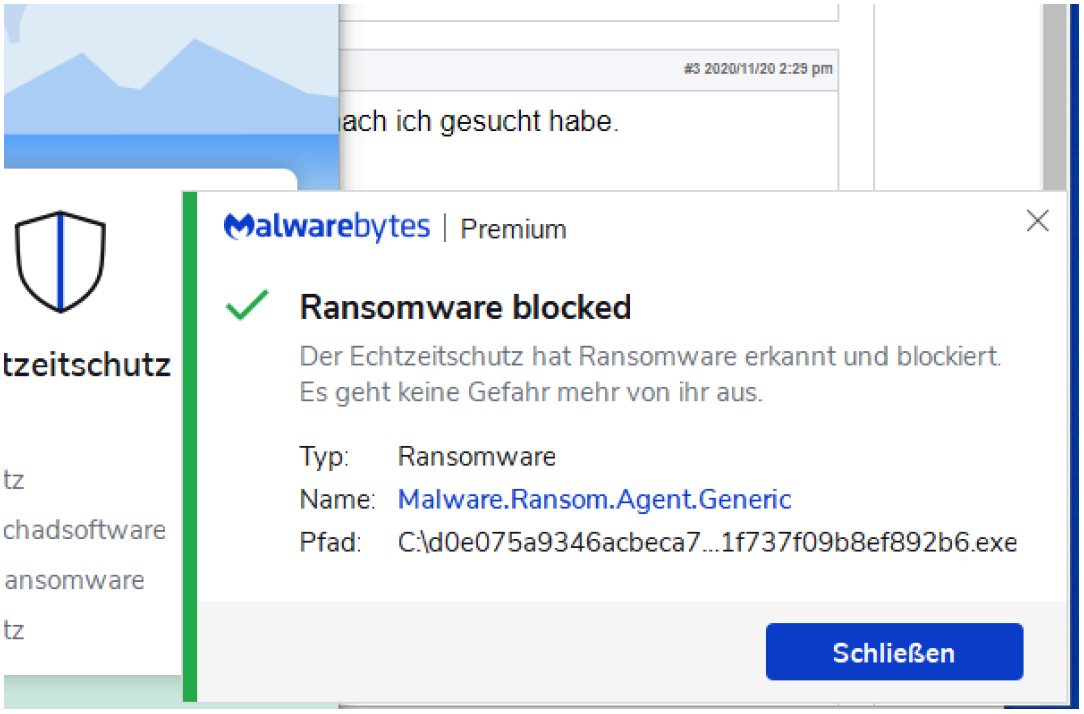
We were able to identify four other variants of the remote templates and payloads. In some of the folders, we found several artifacts using Swedish words, which could indicate that the threat actor is familiar with the language.

Opening the phishing document will trigger a notification via the canarytokens.com website. Typically, people use this type of service to get alerted for a particular event.

This can be very useful as an early warning notification system that an intruder has had access to a network. In this case, the attacker is probably interested in how many people opened the document and perhaps where they are from.

Vaggen ransomware

After being deployed, the ransomware starts encrypting the user's files and adding the .VAGGEN extension to them. When the encryption process is finished, it drops a ransom note on the Desktop, demanding a payment equivalent to 80 USD to be paid in Bitcoin.



The ransomware appears to be coded from scratch and is a relatively straightforward application written in Go which starts with the function denoted as 'main_main'.

Other functions belonging to the main application have obfuscated names, such as: main_FOLOJVAG, main_DUVETVAD, main_ELDBJORT, main_HIDDENBERRIES, main_LAMNARDETTA, main_SPRINGA.

```
main_LAMNARDETTA -> main_enumDir main_ELDBJORT -> main_encryptFile main_SPRINGA -> main_encryptAndRe
```

A full list of the functions, along with their RVAs can be found here.

```
int main_main()
{
  int v0; // ST0C_4
  int v1; // ST10_4
  int note_path; // ST18_4
  int v3; // ST1C_4
  int v4; // ST0C_4
  int v5; // ST10_4
  int v6; // ST08_4
  char v7; // ST08_1
  int v8; // ST0C_4
  int v9; // ST14_4
  int v11; // [esp+Ch] [ebp-28h]
  int v12; // [esp+10h] [ebp-24h]
  int v13; // [esp+14h] [ebp-20h]
  int v14; // [esp+18h] [ebp-1Ch]
  int v15; // [esp+1Ch] [ebp-18h]
  int v16; // [esp+29h] [ebp-8h]
  int v17; // [esp+30h] [ebp-4h]
  void *retaddr; // [esp+34h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  encoding_base64__ptr_Encoding_DecodeString(g_someSeed, aEpeeahuqqp4qvs, 24);
  v16 = 'LATS';
  *(int *)((char *)&v16 + 3) = 'REKL';
  github_com_xxtea_xxtea_go_xxtea_Decrypt(v11, v12, v13, &v16, 7, 7, v14, v15);
  runtime_slicebytetostring(0, note_path, v3, v0, v1);
  v17 = v4;
  main_enumAndEncyptDir(v4, v5);
  os_Getenv(v17, v5, v6, v4);
  runtime_concatstring2(                        // fetch: "\\Desktop\\ABOUT"
    0,
    v7,
    v8,
    "\\Desktop\\ABOUT\n... omitting accept-charsetallocfreetraceannotation-xmlansi_x3.4-1968appengine.
    14);
  runtime_concatstring2(0, v9, note_path, aUr, 3);
  runtime_concatstring2(0, v9, note_path, aFile, 5);
  runtime_concatstring2(0, v9, note_path, aSTxt, 5);
  main_xteaDecryptAndWriteToFile(v9, note_path);// write ransom note: "\\Desktop\\ABOUT_UR_FILES.txt"
  return main_runCommands();
}
```

Some of the strings used by the malware (i.e. the content of the ransom note) are encrypted with the help of XXTEA (using library: xxtea-go). Encrypted chunks are first decoded from Base64. The XXTEA key is hardcoded (“STALKER”). At the end of the execution, the ransom note is dropped on the Desktop.

Encrypting and renaming of the files is deployed as the callback of the standard Golang function: path.filepath.Walk.

```
int __cdecl main_encryptAndRename(int a1, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+Ch] [ebp-1Ch]
  int v6; // [esp+10h] [ebp-18h]
  int v7; // [esp+14h] [ebp-14h]
  int v8; // [esp+18h] [ebp-10h]
  int (__cdecl *callback)(int, int, int, int, int, int, int, int); // [esp+1Ch] [ebp-Ch]
  int v10; // [esp+20h] [ebp-8h]
  int v11; // [esp+24h] [ebp-4h]
  void *retaddr; // [esp+28h] [ebp+0h]

  if ( (unsigned int)&retaddr <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  callback = main_encryptAndRename_func1;
  v10 = a3;
  v11 = a4;
  path_filepath_Walk(a1, a2, &callback);
  result = v5;
  if ( v5 )
  {
    v7 = 0;
    v8 = 0;
    if ( v5 )
      result = *(_DWORD *)(v5 + 4);
    v7 = result;
    v8 = v6;
    result = log_Println(&v7, 1, 1);
  }
  return result;
}
```



Files are encrypted with AES-256 (32 byte long key) in GCM mode.

```
  if ( (unsigned int)&v35 <= *(_DWORD *)(*(_DWORD *)__readfsdword(0x14u) + 8) )
    runtime_morestack_noctxt();
  runtime_concatstring2(&v36, (unsigned int)&unk_8DF883, 10, (const char *)&unk_8D99EF);
  ((void (*)(void))loc_45CDA0)();
  io_ioutil_ReadFile(a1, a2, v2, v3, 3);
  v38 = v4;
  v30 = v5;
  v31 = v6;
  runtime_concatstring2(&v33, (unsigned int)&unk_8F39C7, 43, (const char *)&unk_8E3DF8);
  crypto_aes_NewCipher(&v34, 32, 32, v7, 16);
  crypto_cipher_newGCMWithNonceAndTagSize(v8, v9, 12, 16, v9, v20, v23);
  v32 = v10;
  v11 = *(void (__cdecl **)(int))(v10 + 16);
  v39 = v12;
  v11(v12);
  v29 = v13;
  runtime_makeslice(&uint8_autogen_P60C80, v13, v13, v14);
  v40 = v15;
  io_ReadAtLeast(dword_CD6A10, dword_CD6A14, v15, v29, v29, v29, v16, v26);
  v17 = v27;
  if ( v27 )
  {
    if ( v27 )
      v17 = *(_DWORD *)(v27 + 4);
    v43 = v17;
    v44 = v28;
    fmt_Fprintln(&off_971070, dword_CD65E4, &v43, 1, 1, v21, v24, v27);
  }
```

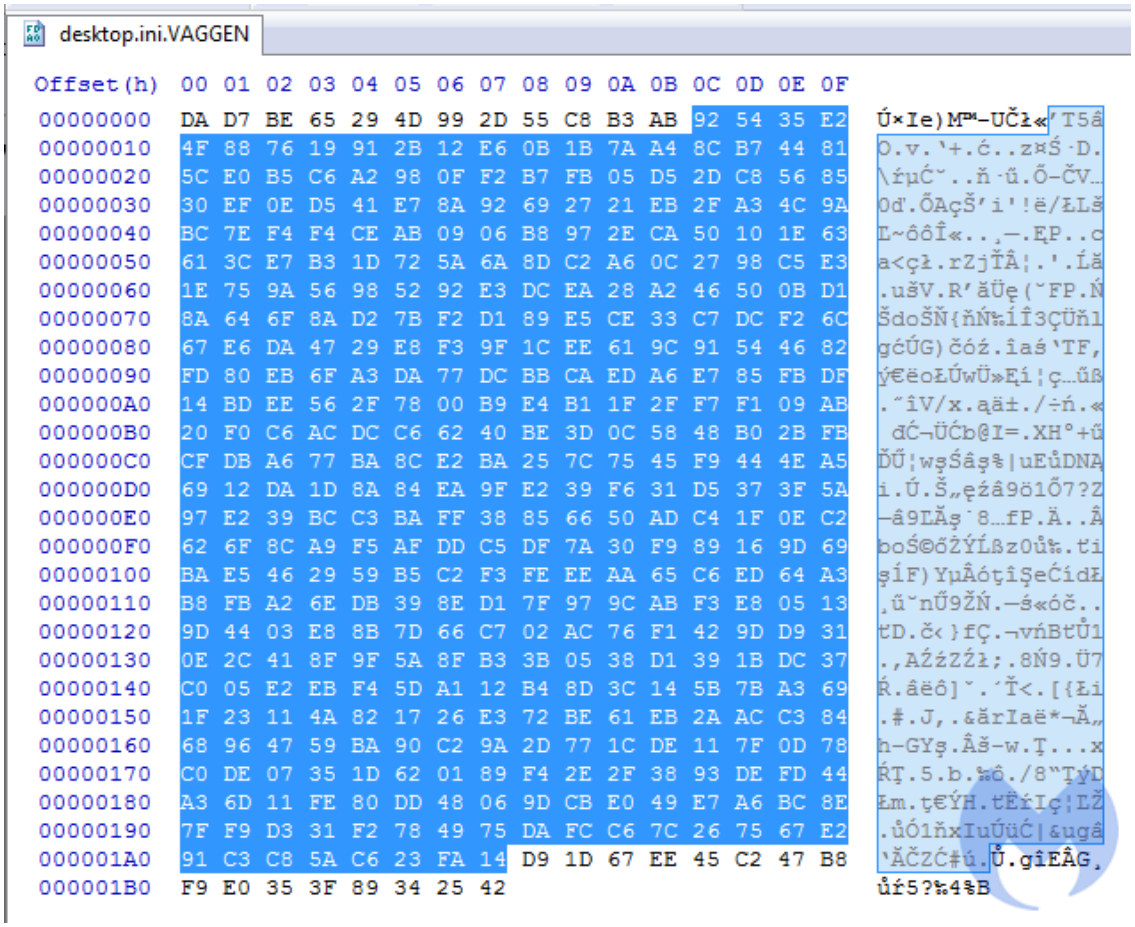


The encryption algorithm is similar to the one demonstrated here. Using a hardcoded key and 12 bytes long nonce, generated by CryptGenRandom. The file content is encrypted with the help of the gcm.Seal function.

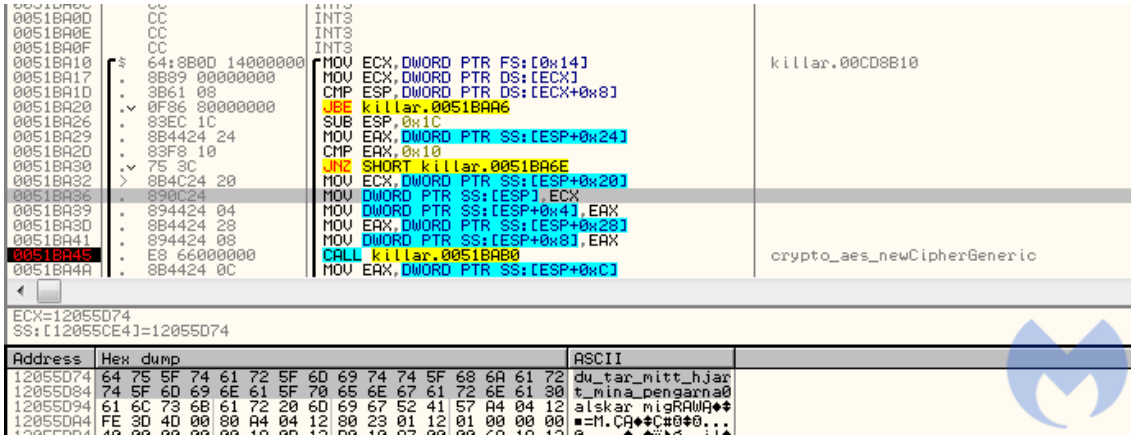


The content of the output file (with .VAGGEN extension) contains:

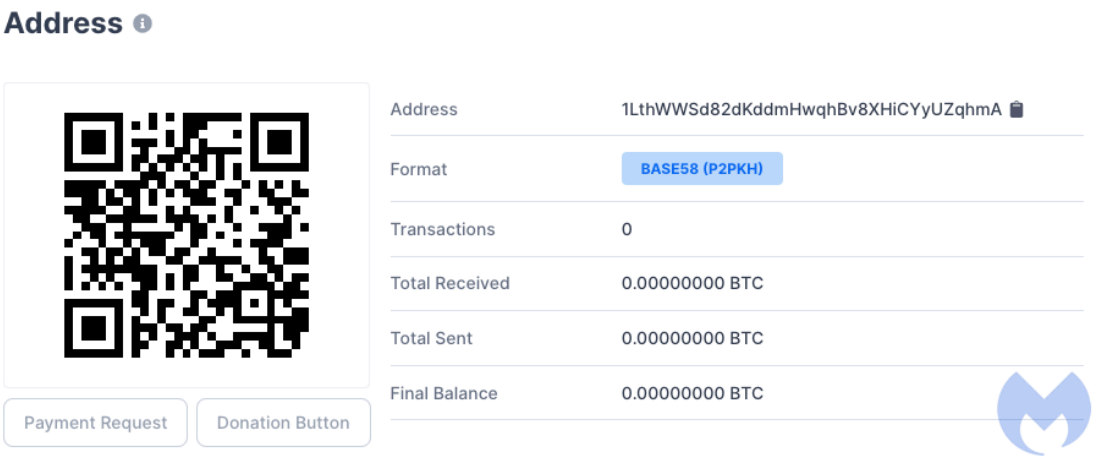
- the 12 bytes long nonce
- the encrypted content
- the 16 byte long GCM Tag



The hardcoded key “du_tar_mitt_hjart_mina_pengarna0” found inside the malware code is Swedish for “you take my heart my money”. Using this key, we can easily decrypt the content.



With all these elements, we can actually recover encrypted files without having to pay the ransom. It appears that the malware author has not received any payment so far at this Bitcoin address.



Unusually low ransom amount

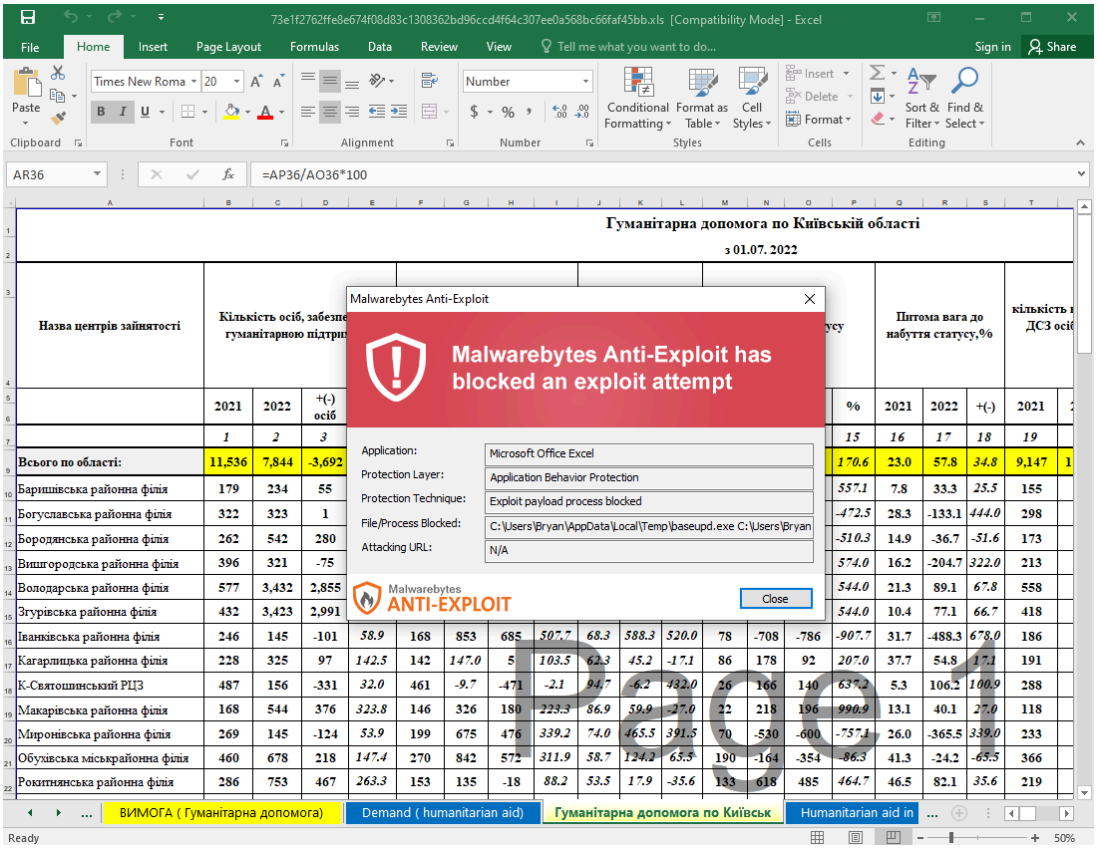
Based on our findings, we believe this is not a sophisticated threat actor, nor affiliated with any of the big ransomware gangs such as Ryuk. The ransom amount is unusually low, and unlike professional ransomware, this attack can be recovered from fairly easy.

However, the phishing attack was well conceived and the template looks well designed, with a nice touch of adding canary tokens. It's unclear at this point if the University of British Columbia was the sole target or not.

Crawling additional repositories created by the threat actor, we found other Word template files that have used a very similar macro to drop a coin miner. This casts more questions about the motivation behind this phishing attack.

We are grateful for the information shared with us by the University of British Columbia. This allowed us to paint a better picture of this attack and understand who the targets were.

Malwarebytes customers were already protected thanks to our signature-less Anti-Exploit layer.



IOCs

Variant1:

summerofficetemplate.dotm

634264996c0d27b71a5b63fb87042e29660de7ae8f18fcc27b97ab86cb92b6d4

notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe

notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe

canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html

canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html

alderson.exe

34842eff9870ea15ce3b3f3ec8d80c6fd6a22f65b6bae187d8eca014f11aafa5

irving.exe

00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant2:

UBC-COVID19-Survey-Mandatory.docx

e869e306c532aaf348725d94c1d5da656228d316071dede76be8fcd7384391c3

template.dotm

334531228a447e4dfd427b08db228c62369461bb2ccde9ab1315649efd0316b1

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/polisen.exe

notabug[.]org/Microsoft-Office/Word-Templates/raw/master/lamnarmighar/killar.exe

canarytokens[.]com/about/d4yeyvldfg6bn5y29by4e9fs3/post.jsp

canarytokens[.]com/articles/6dbbnd503z06qitej1sdzzcvv/index.html

polisen.exe
03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
killar.exe
43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f

Variant3:
template1.dotm
225e19abba17f70df00562e89a5d4ad5e3818e40fd4241120a352aba344074f4
notabug[.]org/Microsoft-Templates/Template/raw/master/irving.exe
notabug[.]org/Microsoft-Templates/Template/raw/master/alderson.exe
canarytokens[.]com/images/tags/8pkmk2o11dmp1xjv5i9svji32/contact.php
canarytokens[.]com/articles/traffic/5ayx8tydzeuzhmq6y5u2lxhpa/post.jsp

Variant4:
smoothtemplates.dotm
ada43ee41f70e327944121217473c536024cd9c90e25088a1a6c5cf895f59fe1
notabug[.]org/arstidar/VARLDVINNA/raw/master/irving.exe
notabug[.]org/arstidar/VARLDVINNA/raw/master/alderson.exe
canarytokens[.]com/traffic/jnk5rpagi54vztro6tau6g1v6/index.html
canarytokens[.]com/traffic/articles/tags/z8yobwprmmopmyfyw8sb1fb0a/index.html
alderson.exe
b4a1a0012abde1ae68f50fa1fe53df7a5d933ec5410731622ab0ad505915cfb6
irving.exe
00c60593dfdc9bbb8b345404586dcf7197c06c7a92dad72dde2808c8cc66c6fe

Variant5:
template.dotm:
7ad8a3c438f36cdfc5928e9f3c7c052463b5987055f583ff716d0382d0eb23b4
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrclean.exe
notabug[.]org/Microsoft-Office/Office-Templates/raw/master/mrmonster.exe
canarytokens[.]com/images/feedback/tags/0xu6dnwmpc1k1j2i3nec3fq2b/post.jsp
canarytokens[.]com/traffic/about/images/ff6x6licr69lmjva84rn65hao/contact.php
mrmonster.exe
f42bbb178e86dc3449662b5fe4c331e3489de01277f00a56add12501bf8e6c23
mrclean.exe
71aadf3c1744f919cddcc157ff5247b1af2e2211b567e1eee2769973b2f7332a





Source: https://blog.malwarebytes.com/cybercrime/2020/10/fake-covid-19-survey-hides-ransomware-in-canadian-university-attack/