

Magniber ransomware improves, expands within Asia | Malwarebytes Labs

By Malwarebytes Labs

Published: 2018-07-15 · Archived: 2026-04-05 19:43:07 UTC

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

- [72fce87a976667a8c09ed844564adc75](#) – loader DLL
 - [19599cad1bbca18ac6473e64710443b7](#) – Magniber’s core DLL

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.

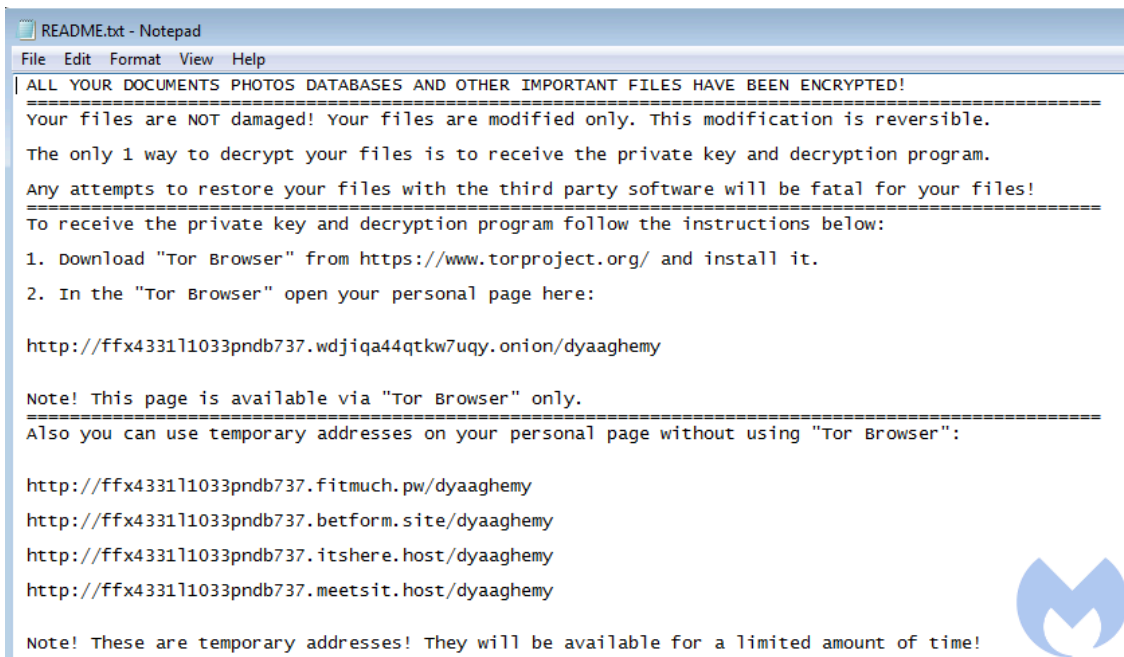


Figure 6. Ransom note left on the infected machine

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

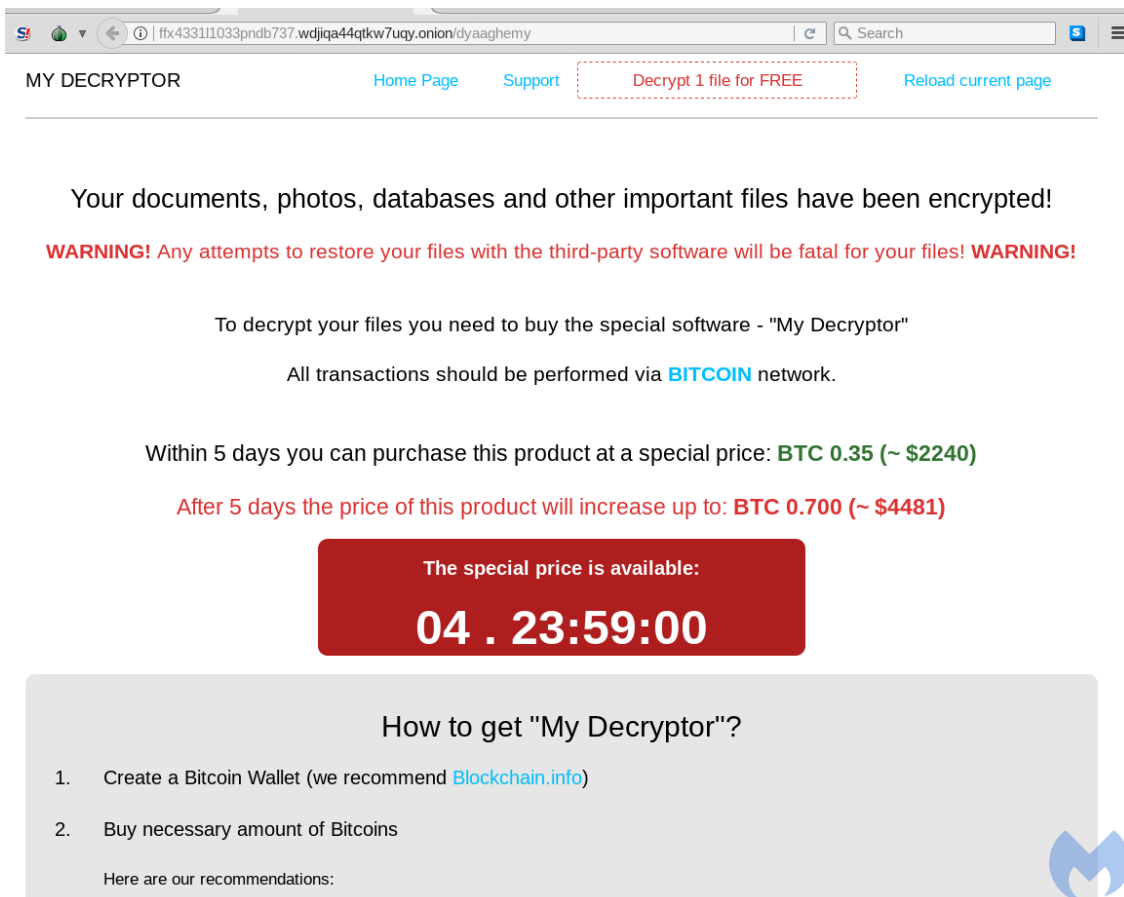


Figure 7. Magniber's payment page

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:

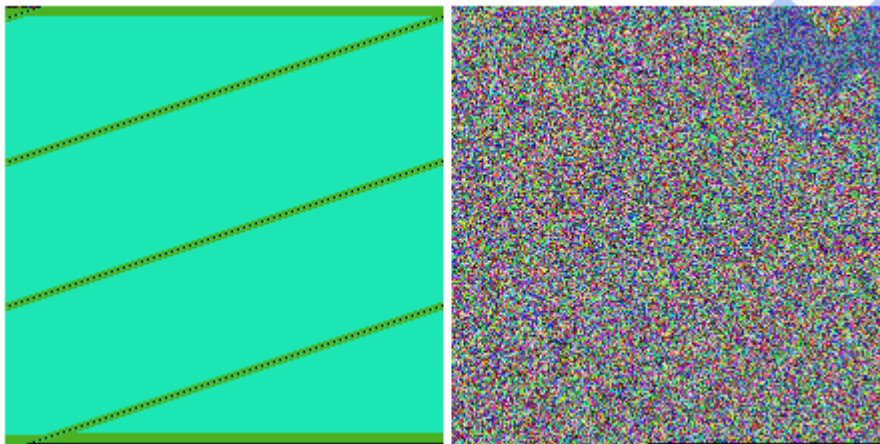


Figure 8. Visualizing a file before and after encryption“>

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	IstrcpyW	10009020	IstrcpyW
1.00	0.99	-----	00409058	IstrcatW	10009024	IstrcatW
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	IstrlenW	10009028	IstrlenW
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	IstrcmpiW	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](https://www.malwarebytes.com/blog/news/2018/07/magniber-ransomware-improves-expands-within-asia).

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    '1'
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:

```

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 .....«.RSA1....
00000010 01 00 01 00 DB D0 6A F1 79 3C BD 4D 8F 25 89 64 ....ŮĐjňy<"MŽ%kd
00000020 3B 3C B4 3B CA 11 9E 3D 3F 92 47 85 F0 62 EF 2E ;<' ;E.ž=?'G...dbd.
00000030 9E FD 95 FB AF 2A 90 60 88 84 E8 E3 3D 16 A2 B7 žý•úž*.'.,čã=.´·
00000040 BE 29 1F DE 61 B7 43 C7 FA 45 21 B5 49 BF 0E 22 I).Ṭa·CÇúE!µIž."
00000050 0F 20 20 31 AD EA 64 B2 81 19 4E AC 43 DC 57 38 . 1.ęd...N-CŪW8
00000060 5F A5 FC D6 60 0A 99 26 06 E7 91 0F 28 EF 8D 81 _AüÖ`.™&.ç`. (dŤ.
00000070 FA 43 A2 F7 33 A0 40 18 5C 51 92 A2 DB 3B CA 9F úC~+3 @.\Q'`Ū;Ež
00000080 93 AB B2 35 85 04 00 EC 6F 19 E4 E6 E7 91 9F 04 "«.5....ěo.áčç'ž.
00000090 E0 31 97 A3 F0 8C 10 81 2C 02 D9 89 F5 DD EC 9F ř1-ŁdŠ...Ů%óŸžž
000000A0 D5 86 A0 B0 69 CD 65 29 CF 61 33 2A 7C B6 9F 50 Ōt °iÍe)Đa3*|ŸžP
000000B0 72 6F 04 68 5C 56 1C D0 A2 22 61 96 60 CA 81 CE ro.h\V.Đ~"a-`E.Ī
000000C0 37 DC B6 AA 93 15 D6 E0 CD 03 85 8F 94 7F 17 FF 7ŪŸŸ".Ōří...ž"...
000000D0 D7 1D 6E 55 61 11 2B 2C 4D 14 5D A8 21 33 FD 84 ×.nUa.+ ,M. ]`!3Ÿ.
000000E0 C2 7C D7 73 6D 4A C1 F7 00 2C 74 92 88 D3 BF 74 Ā|×smJĀ÷., t'.Ōžt
000000F0 B5 8C 94 F7 78 24 54 47 48 35 FA 58 97 3C B0 D1 µš"÷x$TGH5úX<-°N
00000100 8F 7E A6 C8 8C 01 03 FC 6D 2F 6C 50 CD A9 B1 26 ž~|ČŠ..üm/lPÍ@±&
00000110 B7 49 A4 D0 77 00 -I&Đw.

```

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.


```

100086E7 get_next_val proc near
100086E7
100086E7 min_val= dword ptr 4
100086E7 max_val= dword ptr 8
100086E7
100086E7 push esi
100086E8 push 6BCED369h
100086ED call calc_function_by_checksum
100086F2 pop ecx
100086F3 mov ecx, g_storedVal1
100086F9 mov esi, eax
100086FB test cl, 1
100086FE jnz short loc_1000870D

10008700 or ecx, 1
10008703 mov g_storedVal1, ecx
10008709 call esi ; kernel32.GetTickCount
1000870B jmp short loc_10008712

1000870D loc_1000870D:
1000870D mov eax, g_storedVal2

10008712 loc_10008712:
10008712 xor eax, 1
10008715 mov g_storedVal2, eax
1000871A call esi ; kernel32.GetTickCount
1000871C mov ecx, g_storedVal2
10008722 add ecx, eax
    
```

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):

0FB12815	ADD ESP, 0xC	
0FB12818	LEA EAX, DWORD PTR SS:[ESP+0x18]	
0FB1281C	PUSH ESI	
0FB1281D	PUSH EAX	
0FB1281E	PUSH EDI	pdwDataLen
0FB1281F	PUSH 0x0	pbData
0FB12821	XOR EBX, EBX	dwFlags
0FB12823	INC EBX	
0FB12824	PUSH EBX	Final
0FB12825	PUSH 0x0	hHash
0FB12827	PUSH DWORD PTR SS:[ESP+0x2C]	hKey
0FB1282B	CALL DWORD PTR SS:[ESP+0xD4]	advapi32.CryptEncrypt
0FB12832	TEST EAX, EAX	

EDI=001F7D88, (ASCII "7q0cf63vk5896tebuz7guvf93d52259v")

Address	Hex dump	ASCII
004503B0	20 00 00 00 37 71 30 63 66 36 33 76 68 35 38 39	...7q0cf63vk589
004503C0	36 74 65 62 75 7A 37 67 75 76 66 39 33 64 35 32	6tebuz7guvf93d52
004503D0	32 35 39 76 40 00 69 00 63 00 72 00 6F 00 73 00	259vM.i.c.r.o.s.
004503E0	6F 00 66 00 74 00 20 00 45 00 6E 00 68 00 61 00	o.f.t. .E.n.h.a.

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.

```
10005532 movsd
10005533 mov     esi, [esp+0A8h+arg_C]
1000553A push    esi
1000553B xor     edi, edi
1000553D push    edi
1000553E push    edi
1000553F push    1Ch                ; AES 128 bit
10005541 lea    eax, [esp+0B8h+var_1C]
10005548 push    eax
10005549 push    dword ptr [ebx]
1000554B call   [esp+0C0h+var_24] ; advapi32.CryptImportKey
10005552 push    edi
10005553 test   eax, eax
10005555 jz     short loc_1000558A

10005557 mov     edi, [esp+0ACh+var_20]
1000555E lea    eax, [esp+0ACh+var_28]
10005565 push    eax                ; pbData=CRYPT_MODE_CBC
10005566 push    4                 ; KP_MODE
10005568 push    dword ptr [esi] ; hKey
1000556A call   edi                ; advapi32.CryptSetKeyParam
1000556C push    0
1000556E test   eax, eax
10005570 jz     short loc_1000558A

10005572 push    [esp+0ACh+aes_iv] ; pbData
10005579 push    1                 ; KP_IV
1000557B push    dword ptr [esi] ; hKey
1000557D call   edi                ; advapi32.CryptSetKeyParam
1000557F test   eax, eax
10005581 jz     short loc_10005588
```

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 (Malaysia, Brunei).

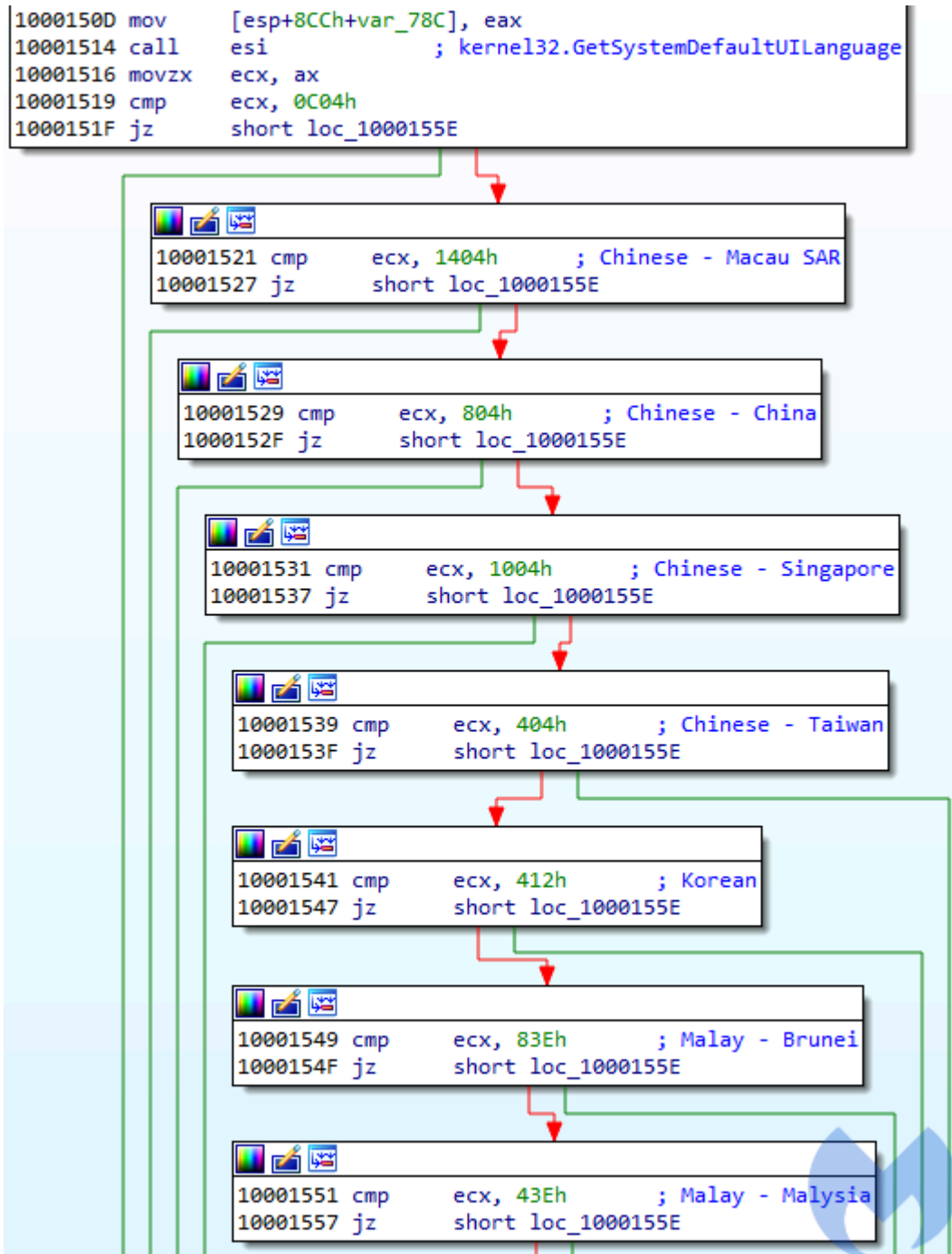


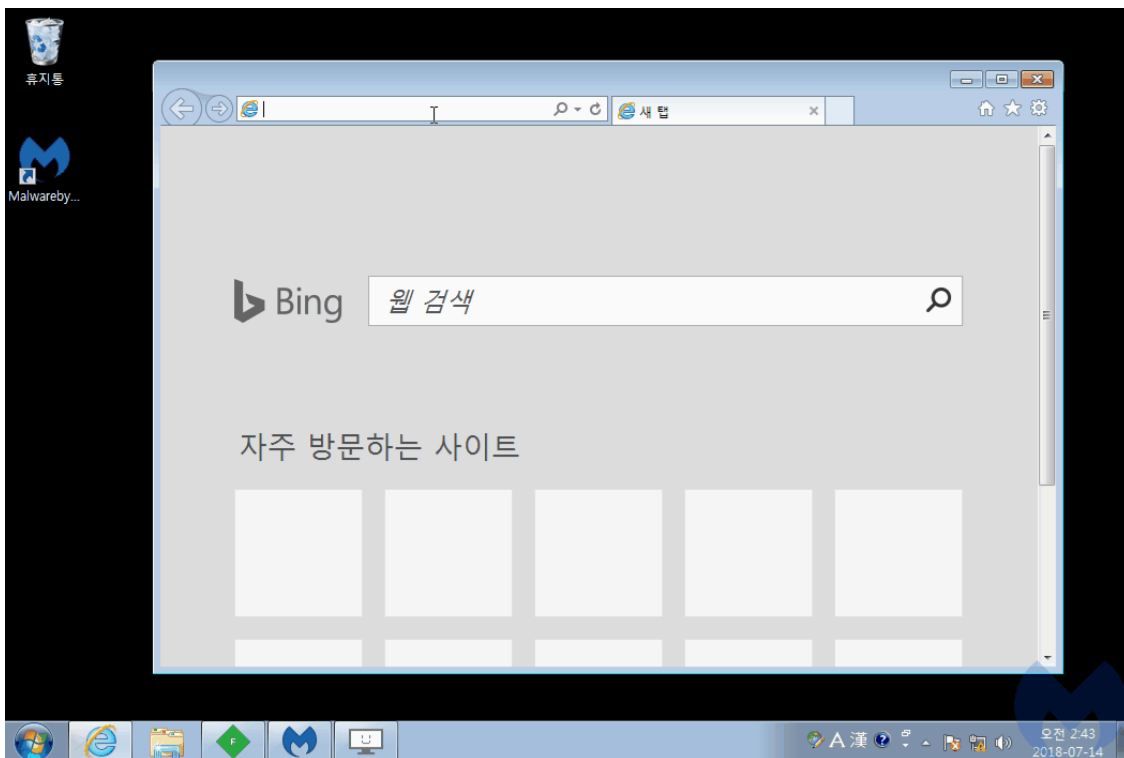
Figure 17. Expanded language checks

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[.]192,69a5010hbjd722q.feedrun[.]onli
```

Code snippets

- [Javascript](#)
- [VBScript](#)

Magniber (original)

```
6e57159209611f2531104449f4bb86a7621fb9fbc2e90add2ecdfe293aa9dfc
```

Magniber (core DLL)

fb6c80ae783c1881487f2376f5cace7532c5eadfc170b39e06e17492652581c2

Source: <https://www.malwarebytes.com/blog/news/2018/07/magniber-ransomware-improves-expands-within-asia>