DMA Locker Strikes Back

blog.malwarebytes.com/threat-analysis/2016/02/dma-locker-strikes-back/

hasherezade February 9, 2016

A few days ago we published a post about a new ransomware – DMA Locker (read more **here**). At that time, it was using a pretty simple way of storing keys. Having the original sample was enough to recover files. Unfortunately, the latest version (discovered February 8) comes with several improvements and RSA key. Let's take a look at the changes.

DMA Locker in recent campaigns have been found installed by the attackers via Remote Desktop (similar distribution method was used by LeChiffre ransomware).

[UPDATE] READ ABOUT THE LATEST VERSION OF DMA LOCKER: 4.0

UPDATE: version 3.0 (discovered 22-th Feb) fixed the bug in the cryptography implementation. Due to this fact, encrypted files cannot be recovered by external tools (although it was possible in case of the earlier version, described in this article). Sorry, but our decryptor can no longer help!

***PREVENTION TIP**: Create these files to protect yourself from this version of DMA Locker. Content doesn't matter. In presence of these files, the program will go by other path of execution and display the red message only – but not deploy the encryption.*

- *C:\Documents and Settings\All Users\decrypting.txt*

- *C:\Documents and Settings\All Users\start.txt*
- *C:\ProgramData\decrypting.txt*
- *C:\ProgramData\start.txt*

This trick works only as a PREVENTION – once your files are encrypted, it is not going to help. For more info about why it happens, please read this post.

Analyzed sample

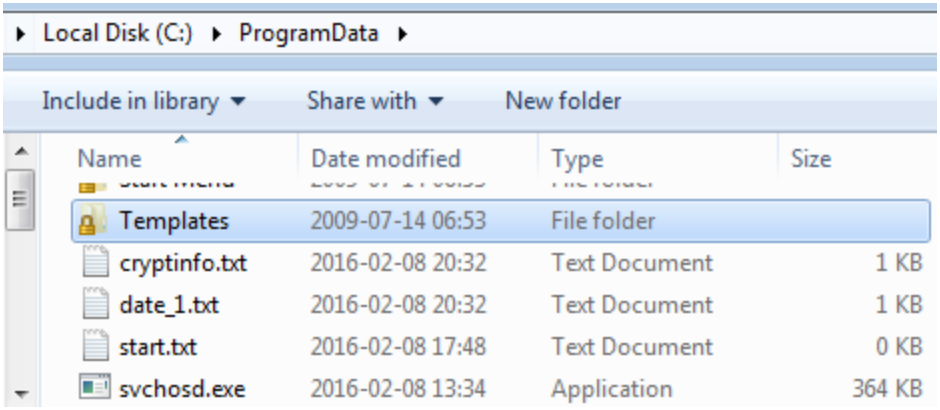
28b44669d6e7bc7ede7f5586a938b1cb

Behavioral analysis

Again we are alerted by a red window – almost identical like before, only the locker image is added:

This time the key necessary to decrypt files must be supplied not as a text, but as RSA key file. Author added also key validation.

Similarly, it drops files in **C:\ProgramData** (or **C:\Documents and Settings\All Users**). Now, the dropped copy is named **svchosd.exe**.



And created registry keys to autorun the file and to autodisplay ransom note via notepad at system startup.

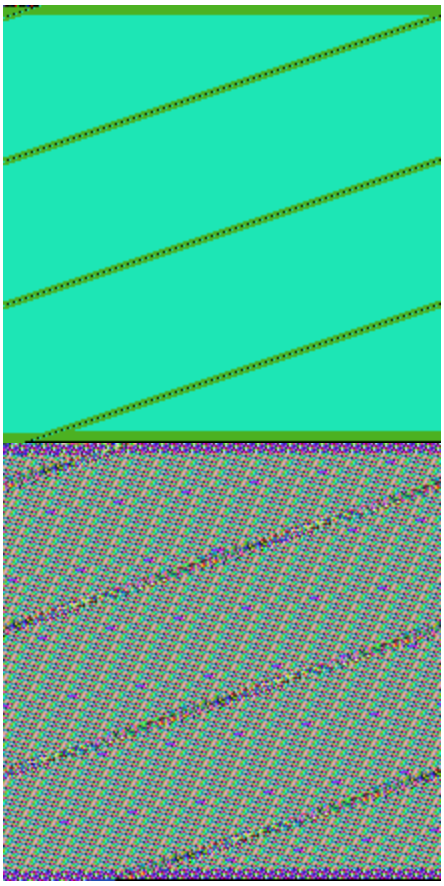
Encrypted files again have unchanged extensions – they can be only recognized by 8 byte long prefix at the beginning of the content. In the previous edition it was "**ABCXYZ11**", in current it is "**!DMALOCK**":

```
00000000  21 44 4d 41 4c 4f 43 4b  ef 02 37 2a b3 56 ff 3c  |!DMALOCK..7*.V.<|
00000010  c1 af b9 d9 f2 5f cf 4b  7f 18 5f 28 c3 1d 41 d9  |....._.K.._(..A.|
00000020  d3 27 8b 7f 7f 97 ee 38  5a f8 37 ab a6 18 df 3a  |.'.....8Z.7....:|
00000030  2f 53 6a ac 9d 48 02 3f  35 1a 8f fb f3 97 95 01  |/Sj..H.?5.......|
00000040  59 2b c4 2f d9 1f ce 3f  c5 7f 1f 35 1b 98 49 e1  |Y+./...?...5..I.|
00000050  f0 ba d1 8e 17 64 8f ad  ea 95 f6 ae b7 a1 c2 93  |.....d..........|
00000060  69 c3 32 9c 8c 10 f5 3f  9c bf 3f 8d c1 71 5c a9  |i.2....?..?..q\.|
```

Experiment

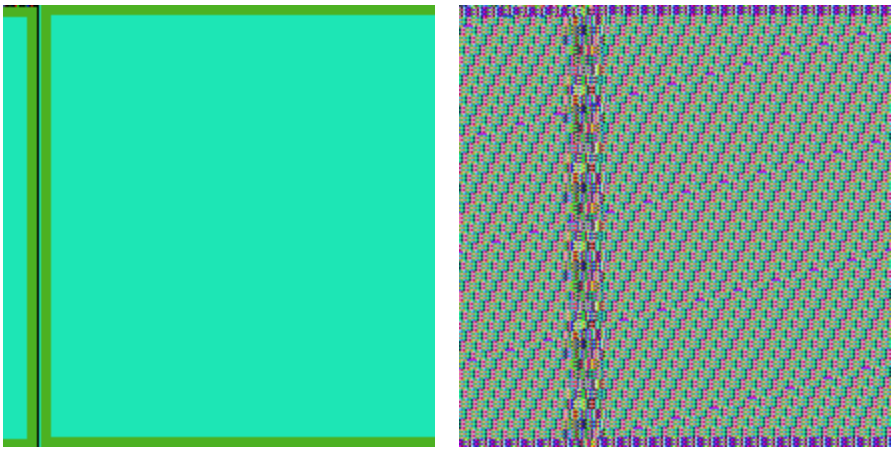
Let's compare how the encrypted files look

From the left we can see visualizations of raw bytes of following files: original, encrypted by previous DMA Locker, encrypted by current DMA Locker



Previous DMA Locker(middle picture) was encrypting files by AES-256 ECB mode, applied on 16 byte long chunks of input. Now (last picture), also repetitive patterns exist – so probably AES-256 ECB mode was used again.

However, pay attention to the strips in the BMP – in a new file they are shifted a bit more. It would suggest that the header is longer than previously. Let's visualize the same files with a different width, to make sure that this impression is right. The header of the file is visualized as a line at the top left corner – it ends where the vertical line starts.



Now it is visible clearly – the header is really longer. Why? To answer this question, code analysis is required – but it can signify, that some additional data have been stored there (it can be for example the AES key, encrypted by RSA).

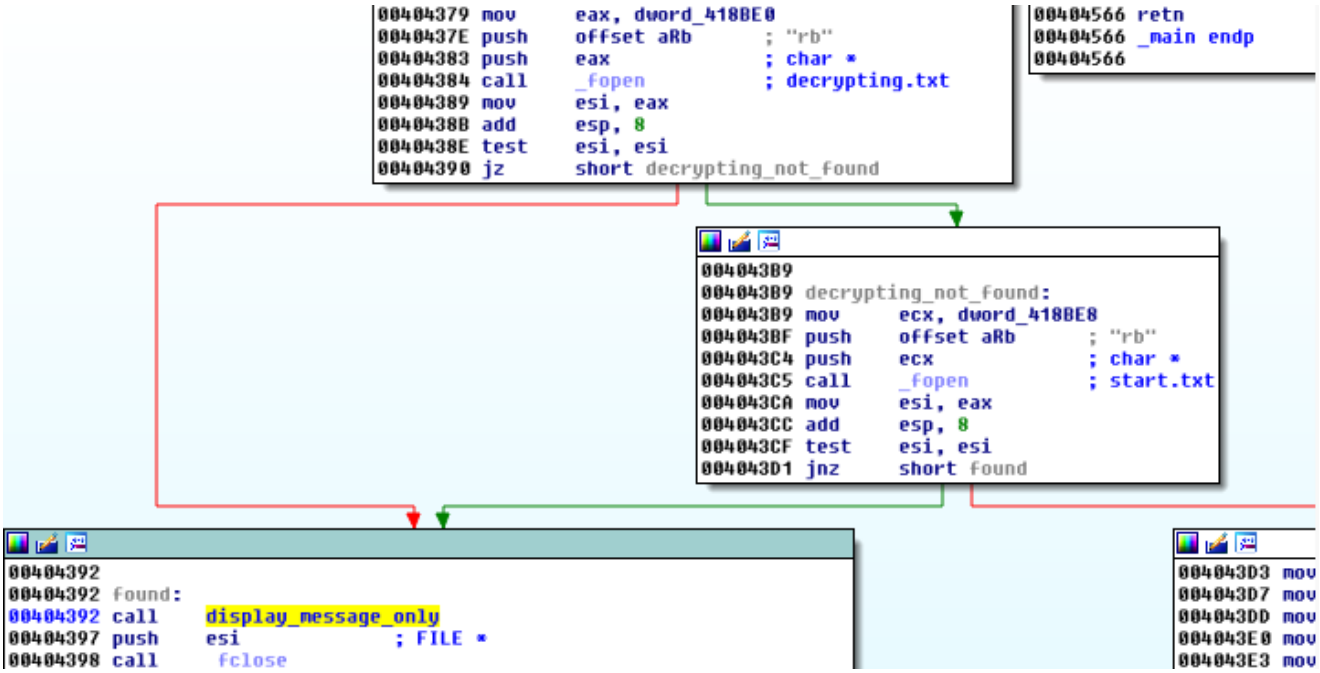
Inside

When does the encryption start?

At the beginning of execution, (as in the previous version) the malware terminates applications used for backups. Also, adds registry keys for its persistence.
Then, execution of the main function may follow 3 alternative paths.

- if system is already infected -> do not deploy encryption, only display the red window with ransom note
- system is not yet infected, malware is not yet installed (current file name is different than the expected one – **svchosd.exe**) -> install the malware in **ProgramData** and then deploy again the dropped file
- system is not yet infected, but malware is installed -> **deploy encryption**, after finishing display the red window with ransom note

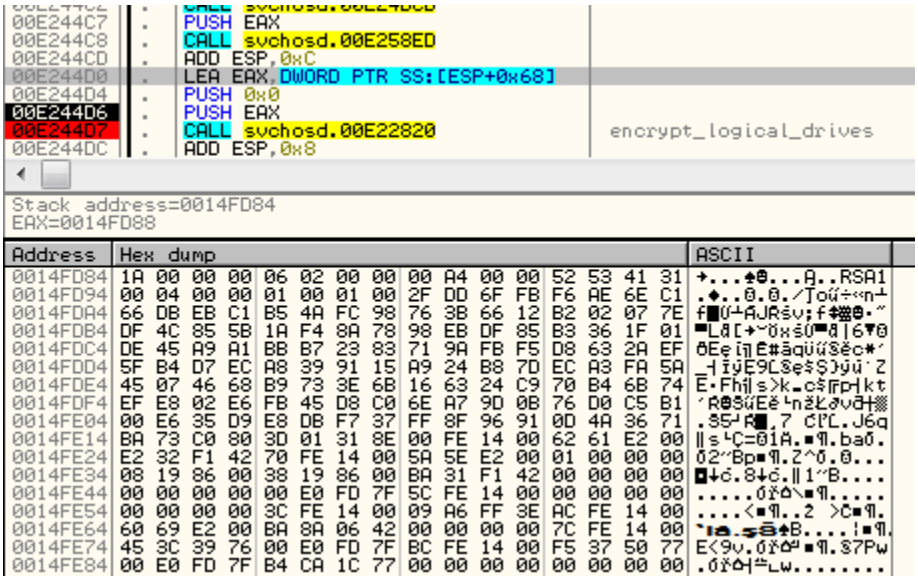
The recognition, in which state is the system, is performed basing on the presence of some predefined files. Presence of file **decrypting.txt** informs that system is already infected. File **start.txt** informs that encrypting started (and no need to start it again):



Knowing this fact, we can easily drop those files by our own and fake that our system is infected. It will prevent this version of DMA Locker from attacking our system (it will display the ransom note but not touch our files).

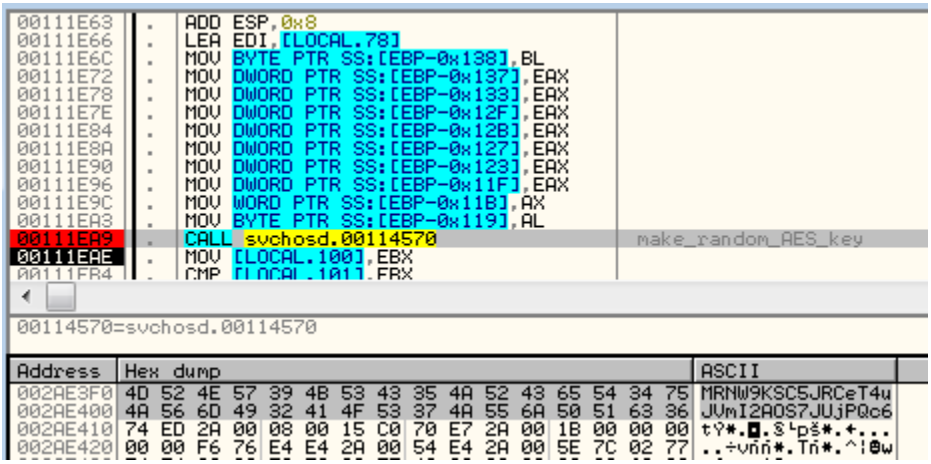
How does the encryption work?

This time the author decided to practice what he preached and really used RSA key (previous version supplied to the encrypting function just a text key, read from the end of the original sample).



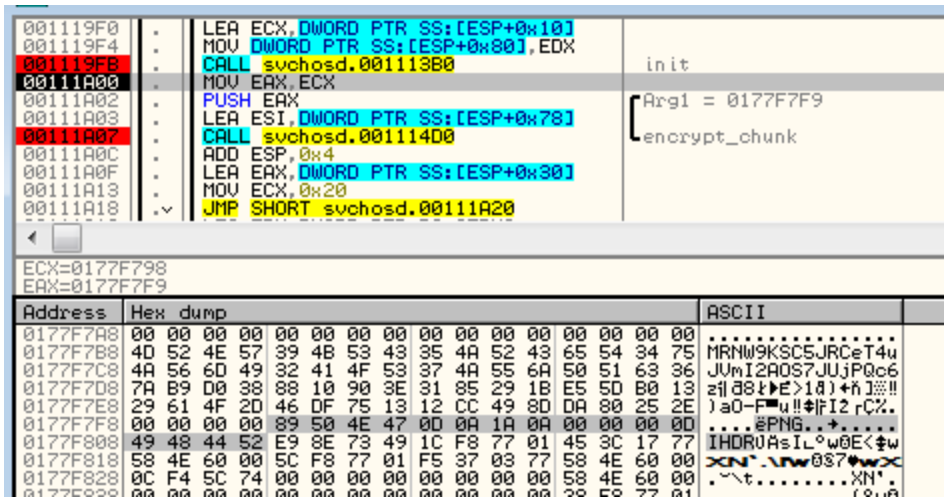
In contrast to the previous edition, where one AES key was used for all the files, here a new random key is generated per every file.

As you can see – in example below the randomly generated key was **MRNW9KSC5JRCeT4uJVmI2AOS7JUjPQc6**

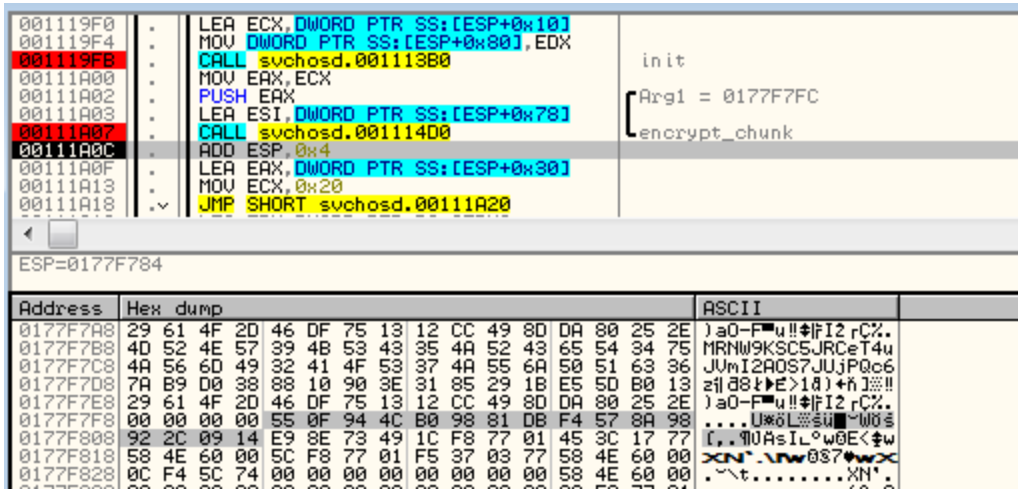


Then, the key is used in the same way like the previous one – to encrypt 16 byte long chunk with AES ECB mode.

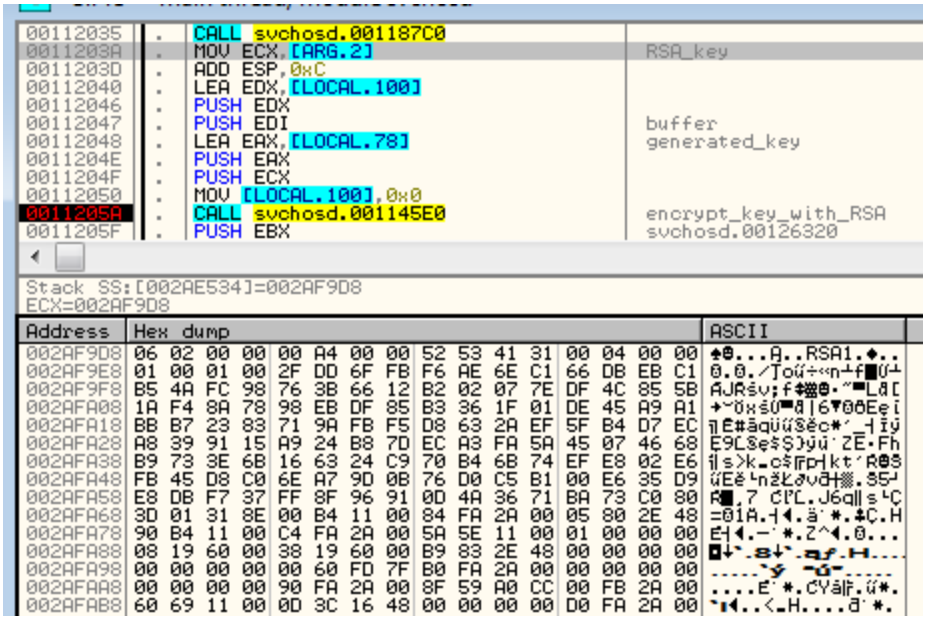
Below – buffer before encryption (fragment of the input is selected on the hex dump – it is a header of a PNG file):



The same chunk encrypted (result in bytes -> “55 0F 94 4C B0 98 81 DB F4 57 8A 98 92 2C 09 14”)



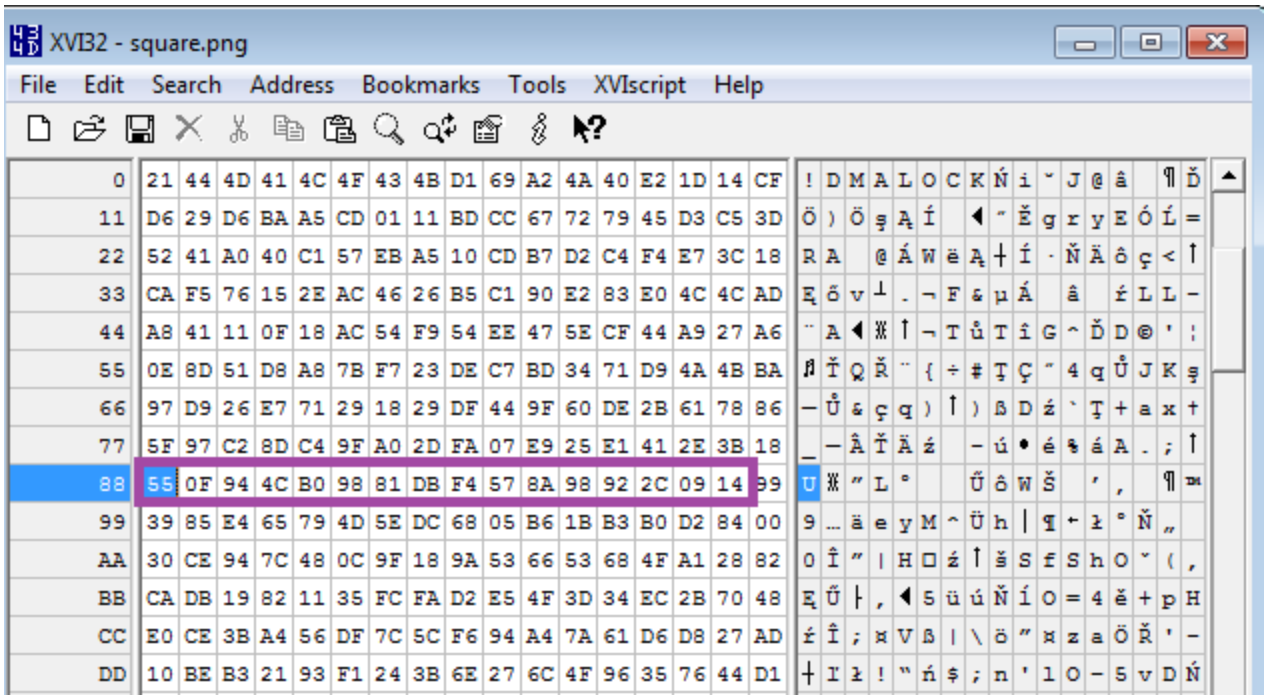
After use, the random AES key is RSA encrypted:



and then, appended to the beginning of the AES encrypted file (just after the “!DMALOCK” signature):

```
00402028 push    1024            ; size_t
0040202D mov     edi, eax
0040202F push    0               ; int
00402031 push    edi             ; void *
00402032 call    _memset
00402037 mov     ecx, [ebp+RSA_key]
0040203A add     esp, 0Ch
0040203D lea     edx, [ebp+var_190]
00402043 push    edx             ; int
00402044 push    edi             ; void *
00402045 lea     eax, [ebp+generated_AES_key]
0040204B push    eax             ; int
0040204C push    ecx             ; int
0040204D mov     [ebp+var_190], 0
00402057 call    encrypt_key_with_RSA
0040205C push    ebx             ; FILE *
0040205D push    8               ; size_t
0040205F push    1               ; size_t
00402061 push    offset aDmalock ; "!DMALOCK"
00402066 call    _fwrite         ; write 'magic' prefix
0040206B mov     edx, [ebp+var_190]
00402071 push    ebx             ; FILE *
00402072 push    edx             ; size_t
00402073 push    1               ; size_t
00402075 push    edi             ; void *
00402076 call    _fwrite         ; write encrypted AES key
```

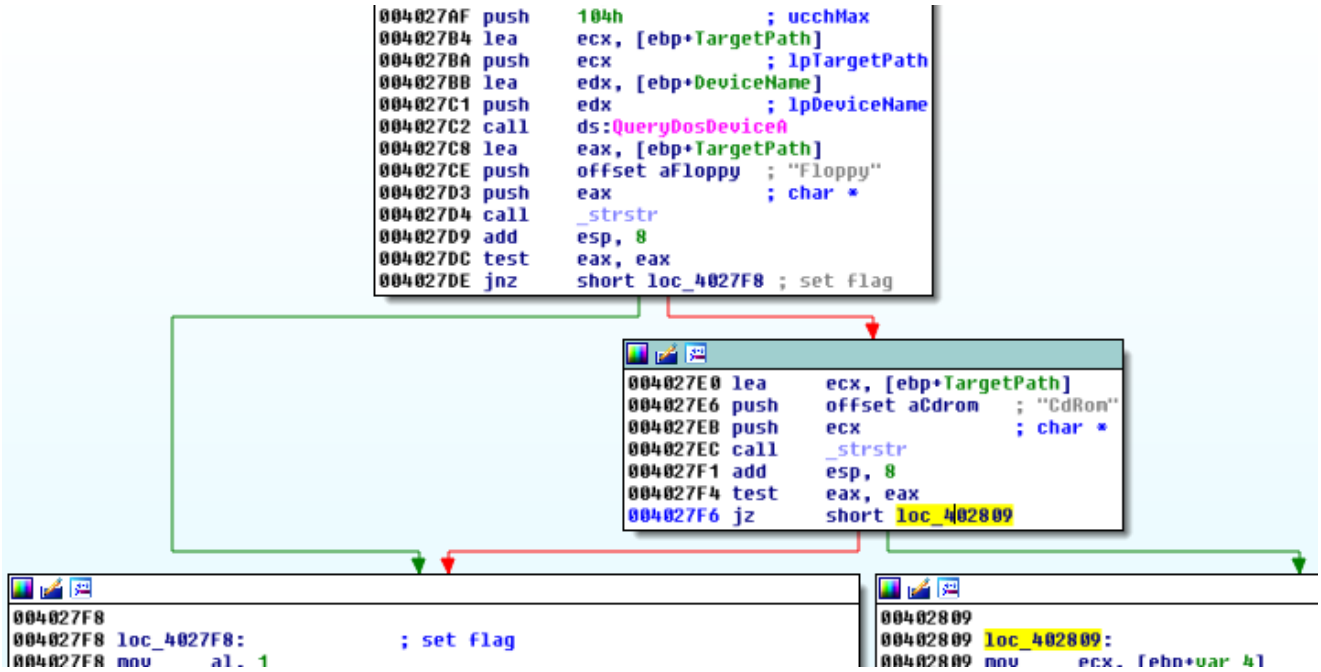
We can see that now the AES encrypted content starts with offset 0x88 (compare the selected part with the above example showing AES encryption result):



What is attacked?

As previous, attacked are logical disks as well as network shares.

This sample introduced also check against Floppy and CD using QueryDosDeviceA (floppy and CD are skipped):



Like in the previous version, skipped are some predefined folders:

```
00402633 sub     esp, 2Ch
00402636 push    esi
00402637 mov     [ebp+var_2C], offset aWindows ; "\\Windows\\"
0040263E mov     [ebp+var_28], offset aWindows_0 ; "\\WINDOWS\\"
00402645 mov     [ebp+var_24], offset aProgramFiles ; "\\Program Files\\"
0040264C mov     [ebp+var_20], offset aProgramFilesX8 ; "\\Program Files (x86)\\"
00402653 mov     [ebp+var_1C], offset aGames ; "Games"
0040265A mov     [ebp+var_18], offset aTemp ; "\\Temp"
00402661 mov     [ebp+var_14], offset aSamplePictures ; "\\Sample Pictures"
00402668 mov     [ebp+var_10], offset aSampleMusic ; "\\Sample Music"
0040266F mov     [ebp+var_C], offset aCache ; "\\cache"
00402676 mov     [ebp+var_8], offset aCache_0 ; "\\Cache"
0040267D xor     esi, esi
```

…and file extensions:

```
004026B7 mov     [ebp+var_30], offset a_exe ; ".exe"
004026BE mov     [ebp+var_2C], offset a_msi ; ".msi"
004026C5 mov     [ebp+var_28], offset a_dll ; ".dll"
004026CC mov     [ebp+var_24], offset a_pif ; ".pif"
004026D3 mov     [ebp+var_20], offset a_scr ; ".scr"
004026DA mov     [ebp+var_1C], offset a_sys ; ".sys"
004026E1 mov     [ebp+var_18], offset a_msp ; ".msp"
004026E8 mov     [ebp+var_14], offset a_com ; ".com"
004026EF mov     [ebp+var_10], offset a_lnk ; ".lnk"
004026F6 mov     [ebp+var_C], offset a_hta ; ".hta"
004026FD mov     [ebp+var_8], offset a_cpl ; ".cpl"
00402704 mov     [ebp+var_4], offset a_msc ; ".msc"
0040270B xor     esi, esi
```

Conclusion

The author of this malware, despite appearing inexperienced in programming, seems to be very determined to gradually improve the quality of the product. The disparity between the quality of the first edition, second (described in the previous article) and the third (current) is significant. We will keep eye on the evolution of this malware family and provide you with updates and possible tips on dealing with this threat.