When a malware is more complex than the paper.

By Sebdraven

Published: 2018-08-29 · Archived: 2026-04-29 08:00:34 UTC

8 min read

Aug 28, 2018

Fireye has published a paper of the backdoor Felixroot after using two vulnerabilities CVE-2017–0199 and CVE-2017–11882. The RTF document drops an executable.

And the Analyst explains:

The dropped executable (MD5: 78734CD268E5C9AB4184E1BBE21A6EB9) contains the compressed FELIXROOT dropper component in the Portable Executable (PE) binary overlay section. When it is executed, it creates two files: an LNK file that points to %system32%\rundll32.exe, and the FELIXROOT loader component. The LNK file is moved to the startup directory. Figure 5 shows the command in the LNK file to execute the loader component of FELIXROOT.

But it’s no so easy. The dropper copies two PE files after using RC4 and a decompression function custom in memory.

The two PE file are an installer and the backdoor Felixroot.

The installer puts on the disk the backdoor and after decrypting strings, it creates the persistance (a lnk in startup folder) and execute run32dll with Felixroot and uses some technics anti forensic to change the timestamps of the backdoor.

Now all technical details !

Encryption and decompression

load the overlay in memory

After a look with Pestudio, the overlay is 53% of the dropper and the entropy is 7,994. it’s too high for a compression.

Press enter or click to view image in full size

property	value
offset	0x0000D800
size	64000 bytes
signature	unknown
md5	6FB79339783EED85A47F5F041541F27A
sha1	2744483F32BCD0FE111A2DA76FE78086376FFEF3
sha256	9DAB5492B69EC34E623888E2BE09D9FE11C4EACB716F16EF6868F04B2120434
entropy	7.994
first-bytes (hex)	1B 73 B4 17 5E 5F 14 59 AF F7 BA AF DA 75 AB F5
first-bytes (text)	.s..._.Y.....u..
file-ratio	53.65 %
virustotal	-
strings-ascii	-
strings-unicode	-

Overlay of the dropper

to dump the overlay, two lines of python are enough. The overlay starts at 0xD800 in the file.

overlay = open('573ea78afb50100f896185164da3b8519e2e0f609a34a7c70460eca5b4ae640d','rb').read()[0xD800:]

open('overlay.dump','wb').write(overlay)

So launch the debugger to understand how this overlay is used by the dropper.

The dropper starts to read itself.

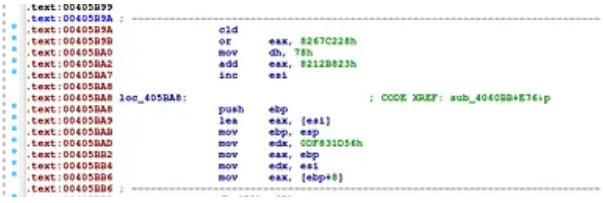
Press enter or click to view image in full size

Read itself

The file handler is in EAX as value 288.

If we check in IDA, this part is badly interpreted by IDA. It's patched at runtime.

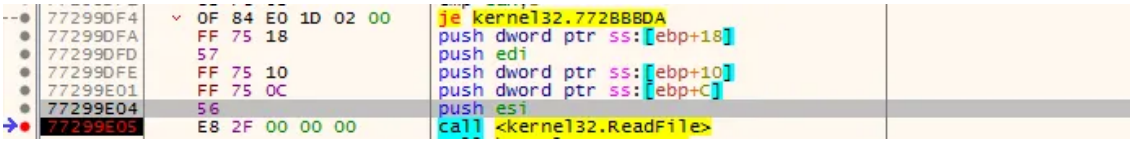
Press enter or click to view image in full size



erro in IDA

So the best way, it's to set a breakpoint at the CreateFile and ReadFile.

So it reads and stores the result in [ebp + C].(here 194408)

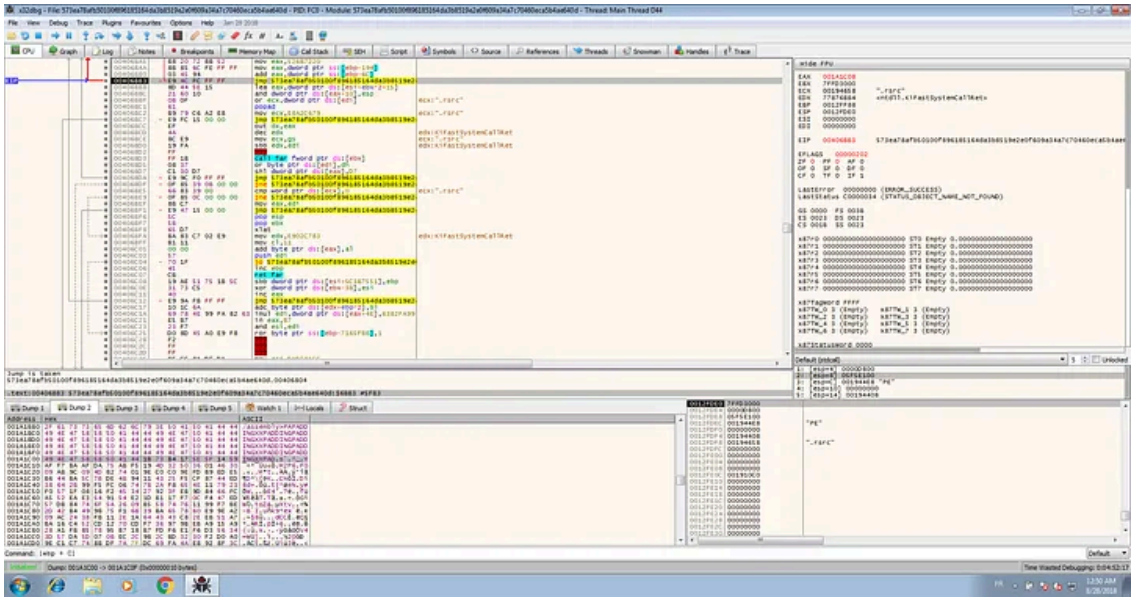


```
00194400 53 A8 57 28 63 1B 00 18 4D 5A 90 00 03 00 00 00 S¨W(c...MZ......
00194410 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 ....ÿÿ..¸.......
00194420 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 @...............
00194430 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00194440 00 00 00 00 E0 00 00 00 0E 1F BA 0E 00 B4 09 CD ....à.....°..´.Í
00194450 21 B8 01 4C CD 21 54 68 69 73 20 70 72 6F 67 72 !¸.LÍ!This progr
00194460 61 6D 20 63 61 6E 6E 6F 74 20 62 65 20 72 75 6E am cannot be run
00194470 20 69 6E 20 44 4F 53 20 6D 6F 64 65 2E 0D 0D 0A  in DOS mode....
00194480 24 00 00 00 00 00 00 00 84 12 43 4C C0 73 2D 1F $.........CLÀs-.
00194490 C0 73 2D 1F C0 73 2D 1F DE 21 A9 1F E3 73 2D 1F Às-.Às-.Þ!©.ãs-.
001944A0 DE 21 B8 1F D0 73 2D 1F DE 21 AE 1F A0 73 2D 1F Þ!¸.Ðs-.Þ!®. s-.
001944B0 E7 B5 56 1F C7 73 2D 1F C0 73 2C 1F B2 73 2D 1F çµV.Çs-.Às,.²s-.
001944C0 DE 21 A7 1F C1 73 2D 1F DE 21 BC 1F C1 73 2D 1F Þ!§.Ás-.Þ!¼.Ás-.
001944D0 52 69 63 68 C0 73 2D 1F 00 00 00 00 00 00 00 00 RichÀs-.........
001944E0 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 04 00 ........PE..L...
001944F0 D7 E9 14 4F 00 00 00 00 00 00 00 00 E0 00 03 01 xé.O........à...
00194500 0B 01 09 00 00 96 00 00 00 3E 00 00 00 00 00 00 .....–...>......
```

And the dropper seeks to D800 to set EAX at the start of the overlay.

RC4 and Custom decompression

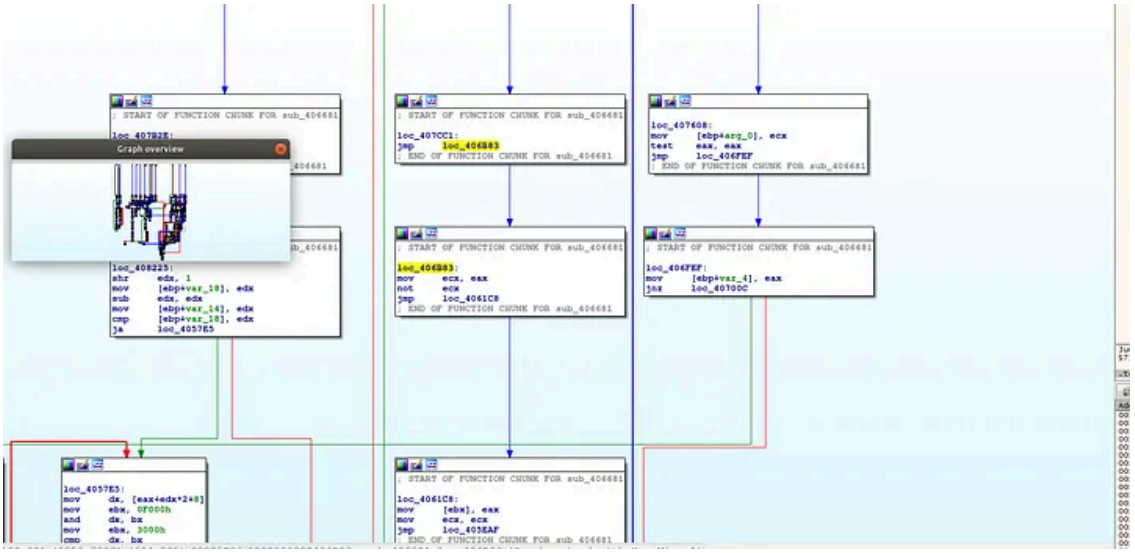
Press enter or click to view image in full size



Overlay in memory

if we check in IDA where we are.

Press enter or click to view image in full size



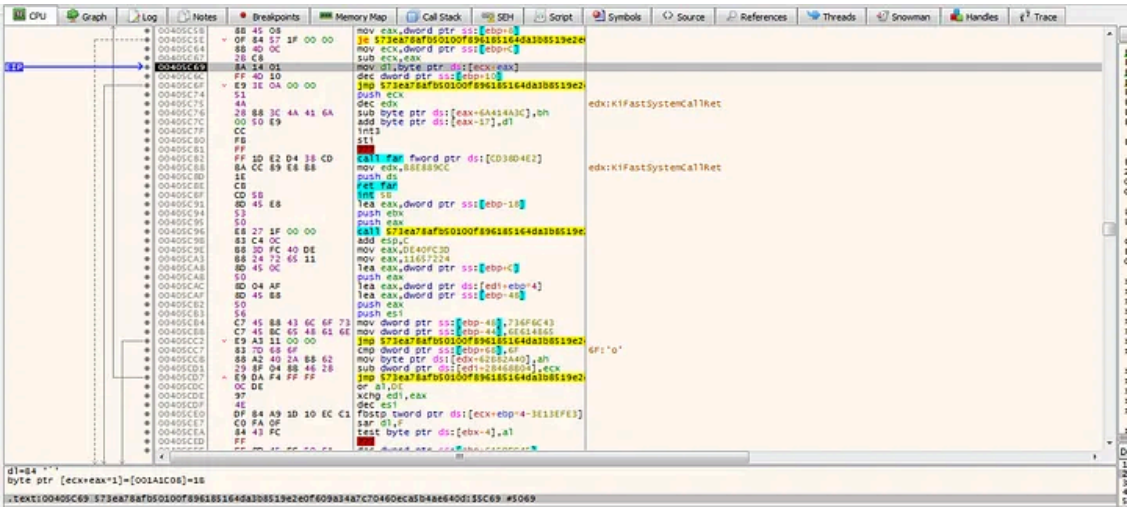
sub_406681

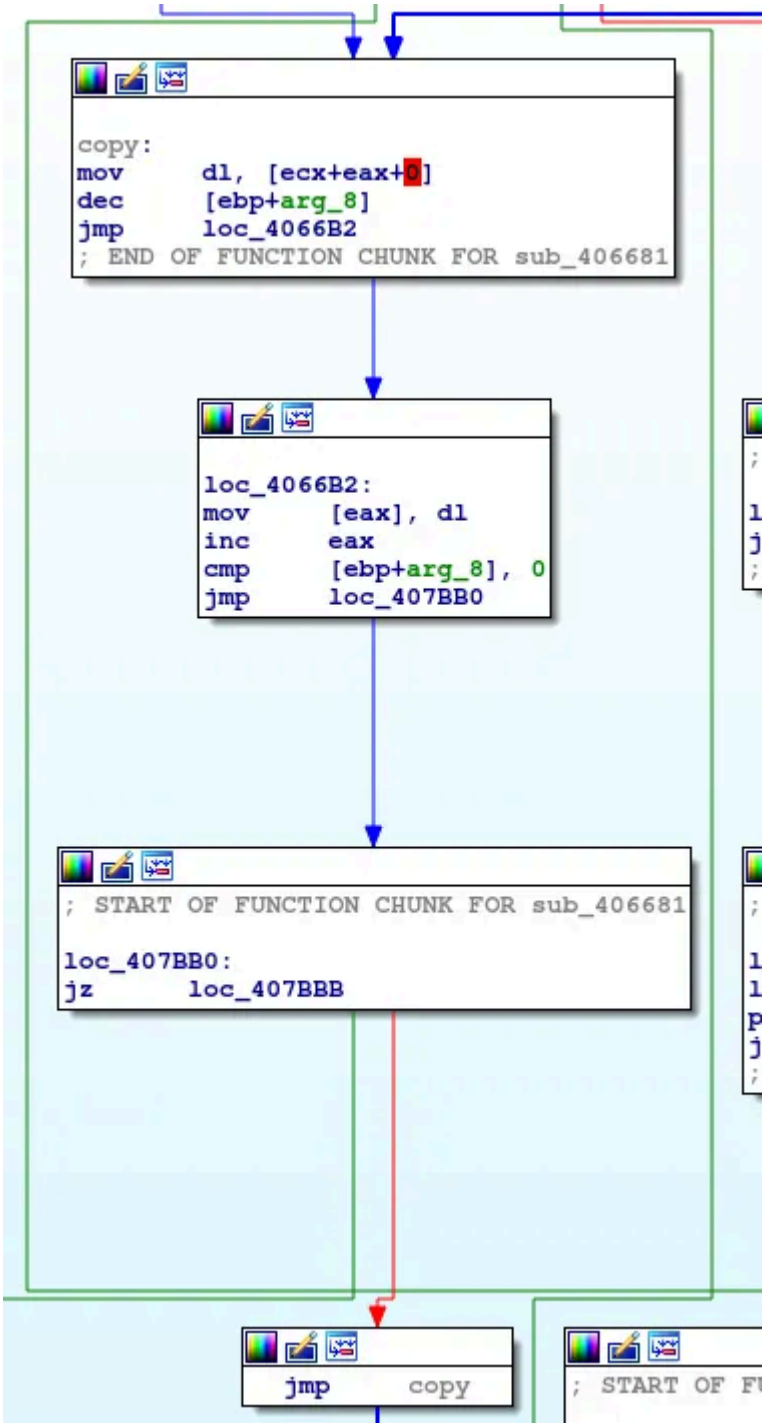
If we check the the graph overview,this function is in a huge function with many jmp.

It seems this function is like a packer.

In the second step, it reads 40 bytes (from C08 to C30)

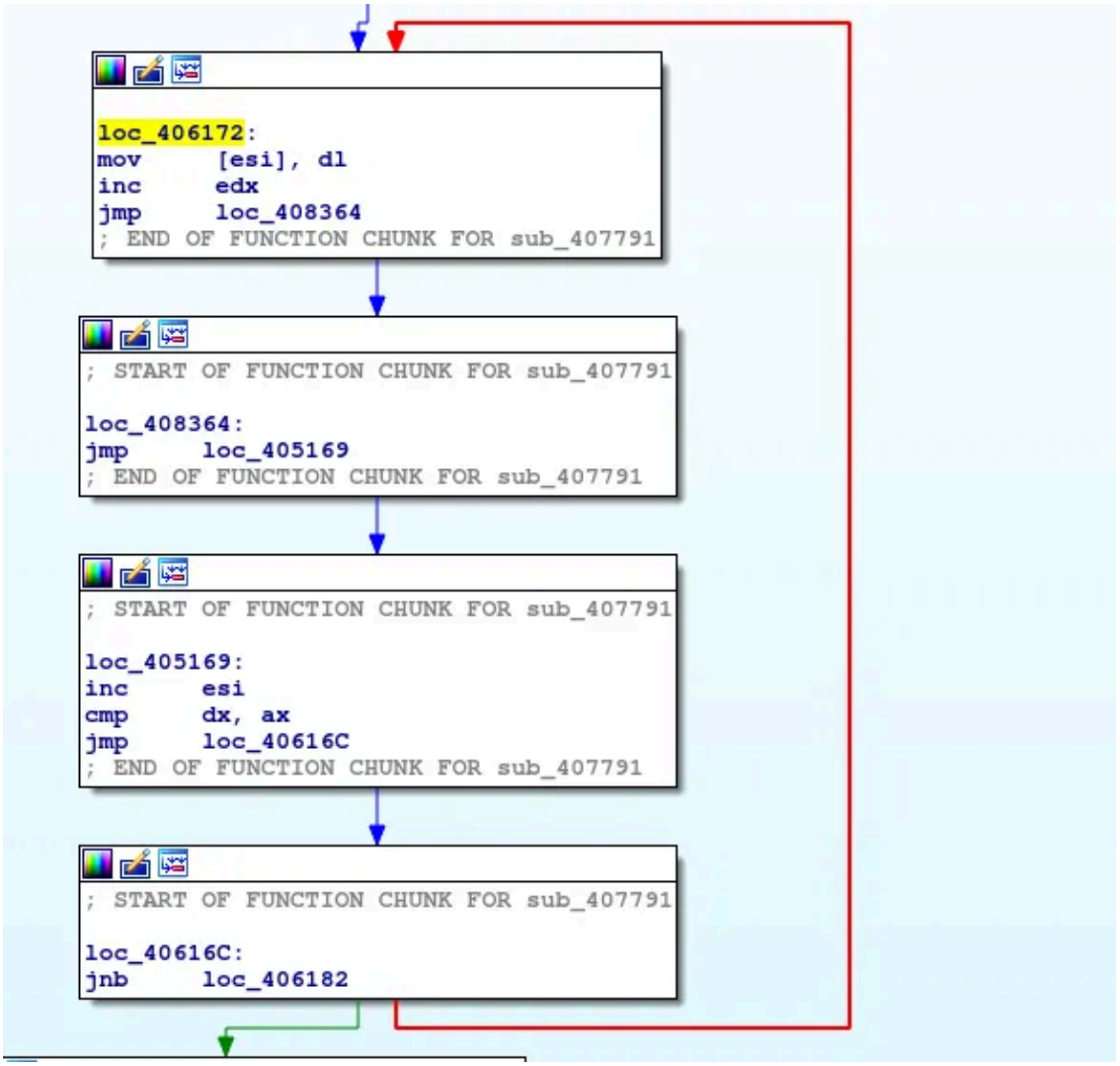
Press enter or click to view image in full size



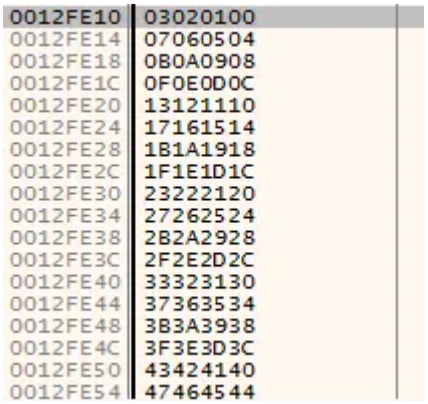


The loop is made by the the jmp to go to the start of the function if the 40bits are not read.

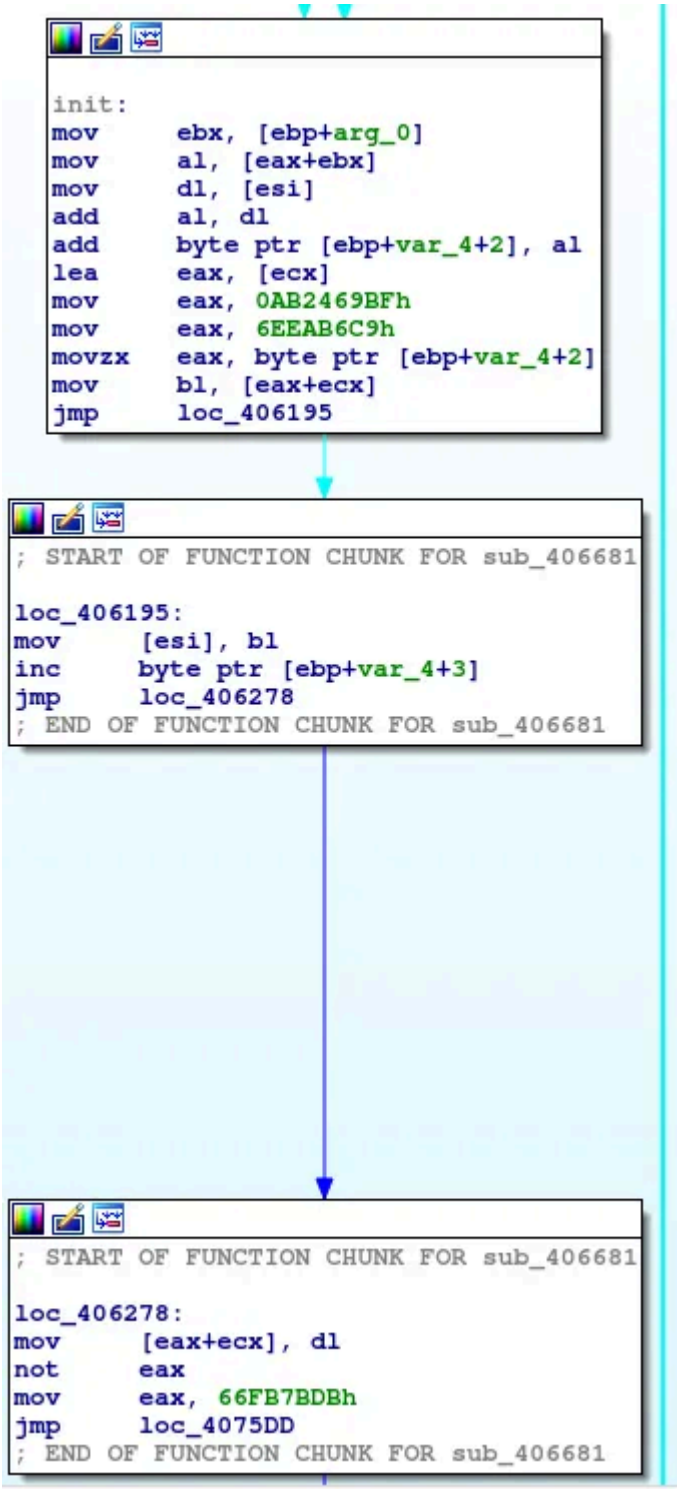
After reading the 40 bits, we have a loop of 256 steps, to store 01 to 256 on the stack.



init of RC4



And it manipulates the 40 bytes and stores the result on the stack in a loop of 256 steps.

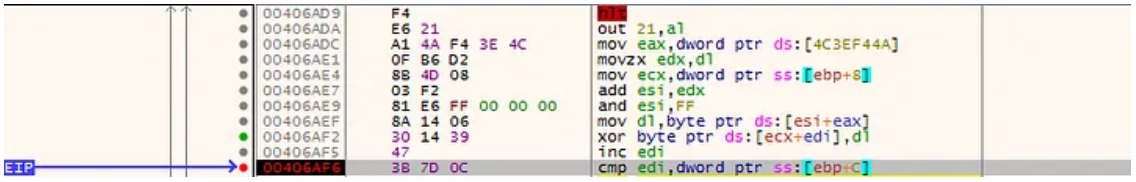


Init of RC4

Press enter or click to view image in full size

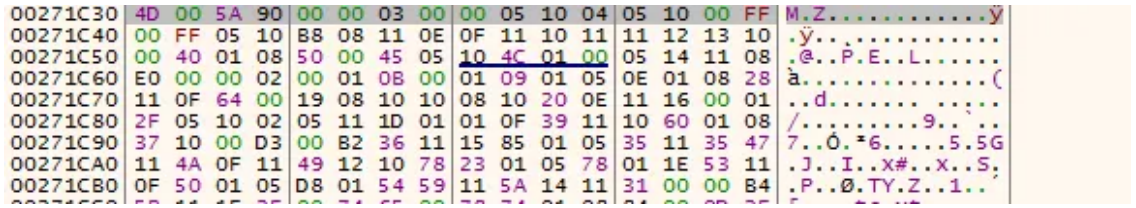
And a third function with aritmethic operations works on the overlay with the results of the two lasted functions

Press enter or click to view image in full size



[EBP+C] store F9D8. Remember !? It's the size of file.

The result of this function is stored at the same place of the overlay in C30.



Ok, everybody has recognized the three steps of RC4.

if we do a comparaison using python langage, the first functions is:

```
self.state = list(range(256))
```

The second function is:

```
def init(self, key):
        for i in range(256):
            self.x = (ord(key[i % len(key)]) + self.state[i] + self.x) & 0xFF
            self.state[i], self.state[self.x] = self.state[self.x], self.state[i]
        self.x = 0
```

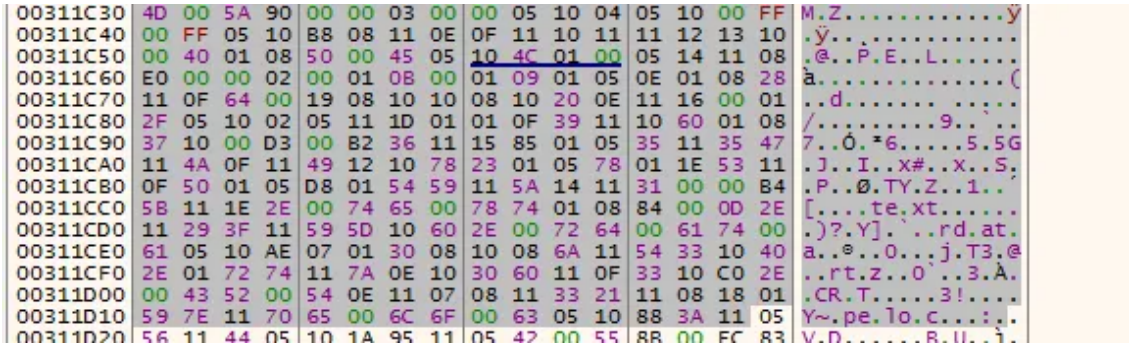
And the third function is:

```
def decrypt(self, input):
        output = [None]*len(input)
        for i in range(len(input)):
            self.x = (self.x + 1) & 0xFF
            self.y = (self.state[self.x] + self.y) & 0xFF
            self.state[self.x], self.state[self.y] = self.state[self.y], self.state[self.x]
            output[i] = chr((ord(input[i]) ^ self.state[(self.state[self.x] + self.state[self.y]) &
        return ''.join(output)
```

So here, the key of the RC4 is the first 40 bytes of the overlay.

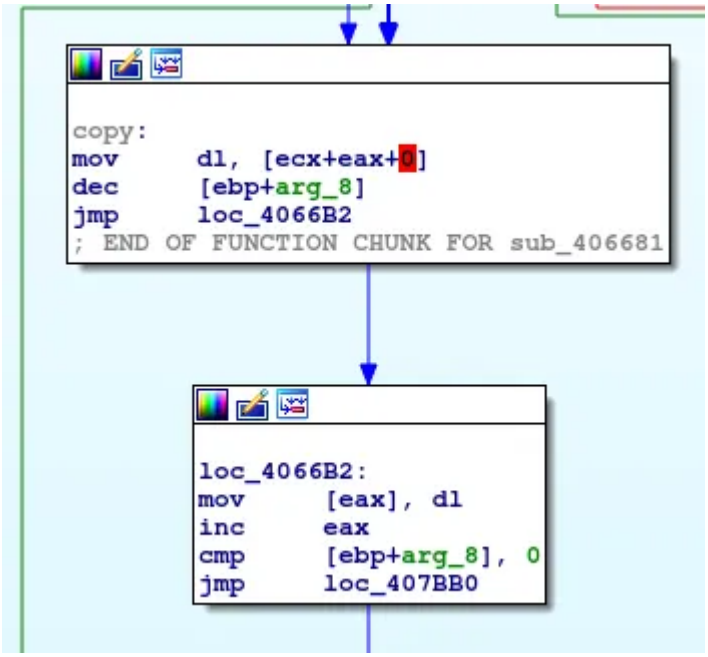
1B 73 B4 17 5E 5F 14 59 AF F7 BA AF DA 75 AB F5 19 4D 32 50 36 01 46 30 09 AB 9C 09 4D B2 74 01 9E C0 C0 9E FD B9 ED E5

The result seems to be a PE file but not totally.

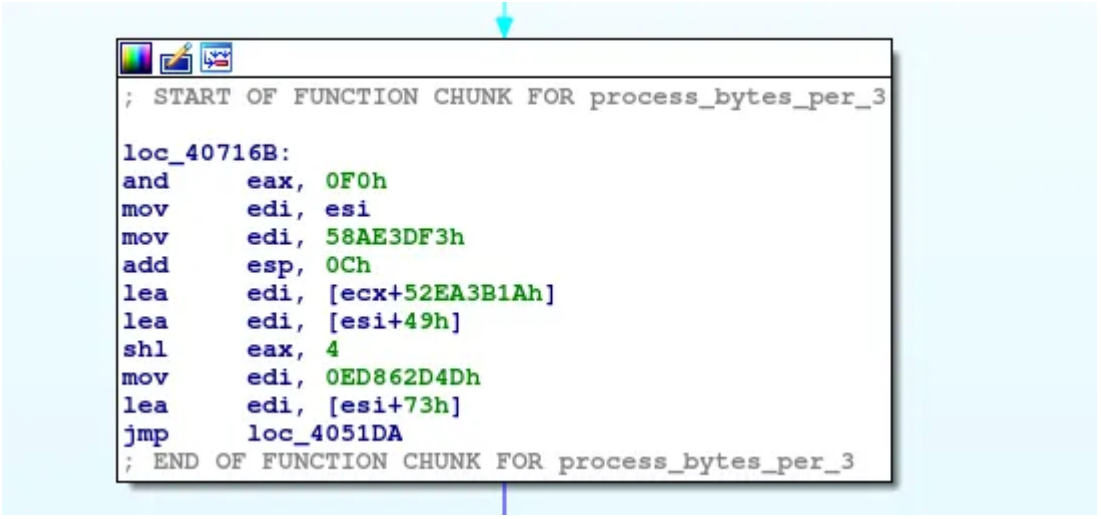


After that, the dropper increases the stack from 12F000 to 12C000 before launching the decompression function (seemly custom after many searches, but if it's not that, write a comment at the end of this post !)

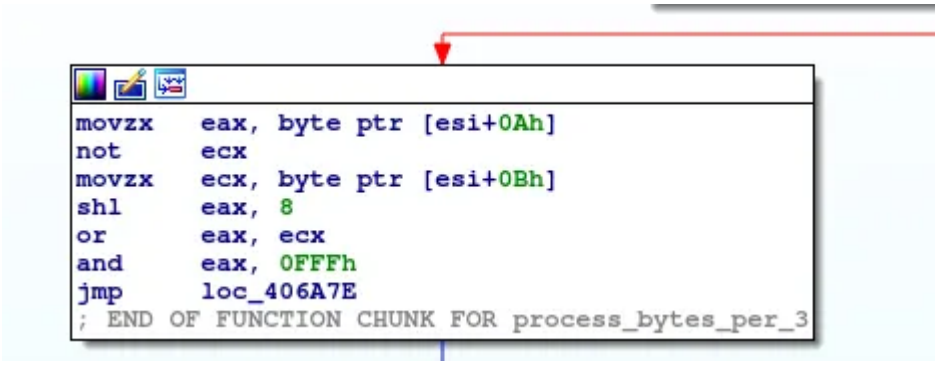
the dropper puts the three bytes [AB,CD,EF] in the stack.



The algorithm used in the first AB,C and secondly D, EF in this function sub_004055FC.



AB,C



D,EF

If C or D == 0 then the dropper writes AB or EF.

If C != 0 then the dropper writes 0 and the number of 0 depends on AB

if D!= 0 then the dropper write 0 and the number of 0 depends on EF

Get Sebdraven's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

We develop many examples to better understand:

4D 00 5A -> 4D 5A

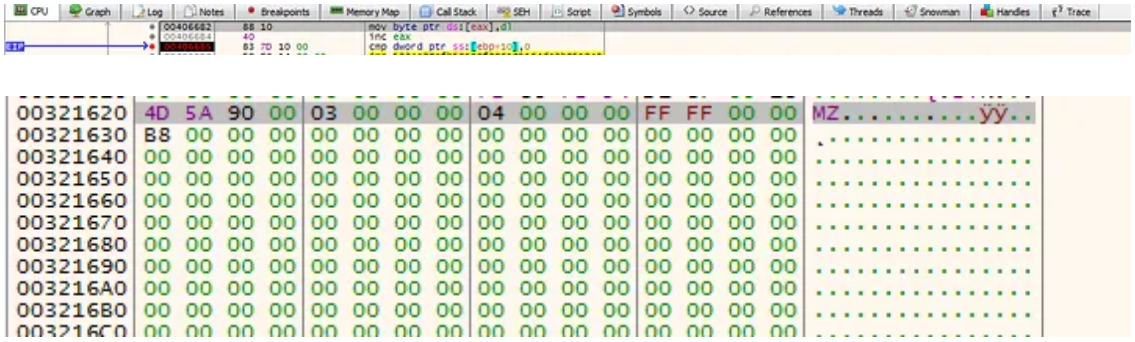
90 00 00 -> 90 00

03 00 00 -> 03 00

51 10 04 -> 00 00 04

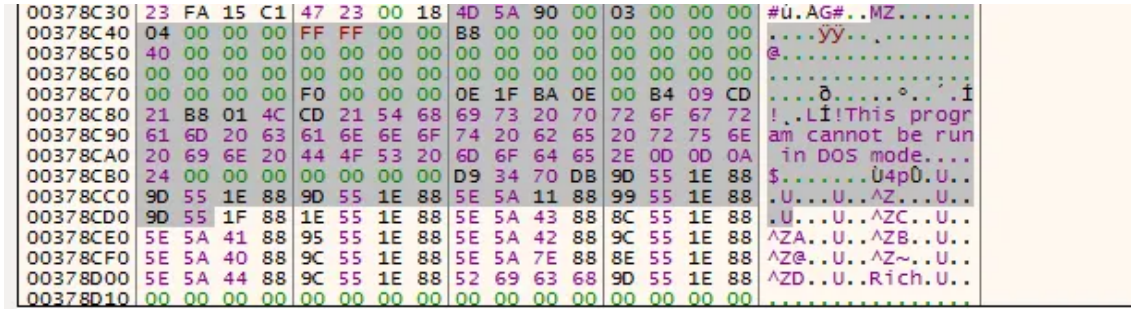
The result is stored in a buffer. The address of the buffer is stored by EAX.

Press enter or click to view image in full size



The dropper decodes two PE Files: another dropper which we name drop and the backdoor Felixroot.

Felixroot is copyied to 378C38.



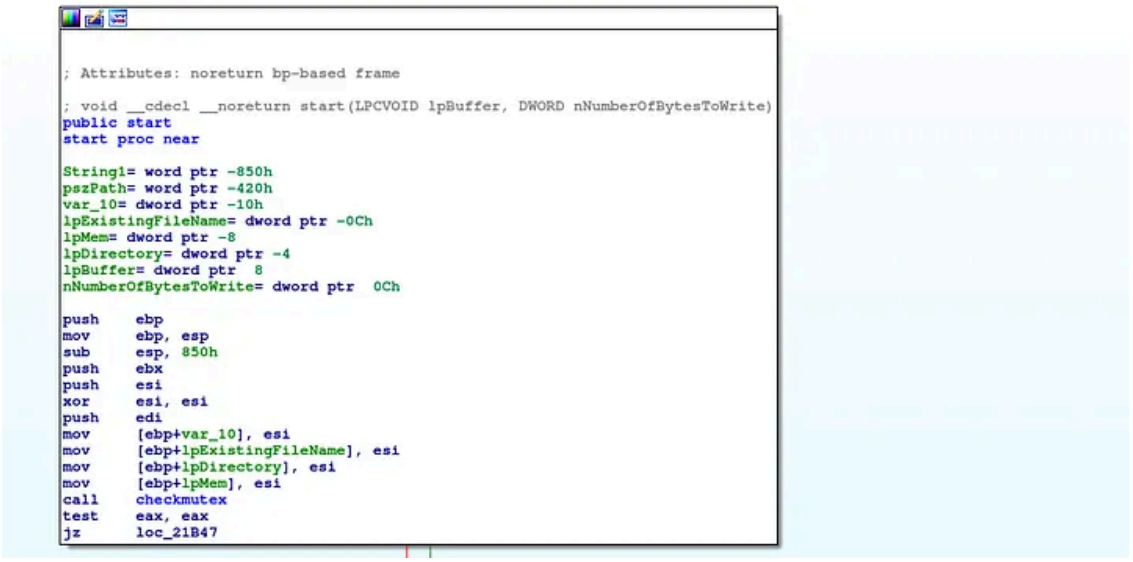
drop is copied after a Virtualloc at 20000 and drop is executed in 00021964.

Installation

drop verifies if it’s executed after the dropper by checking a mutex.

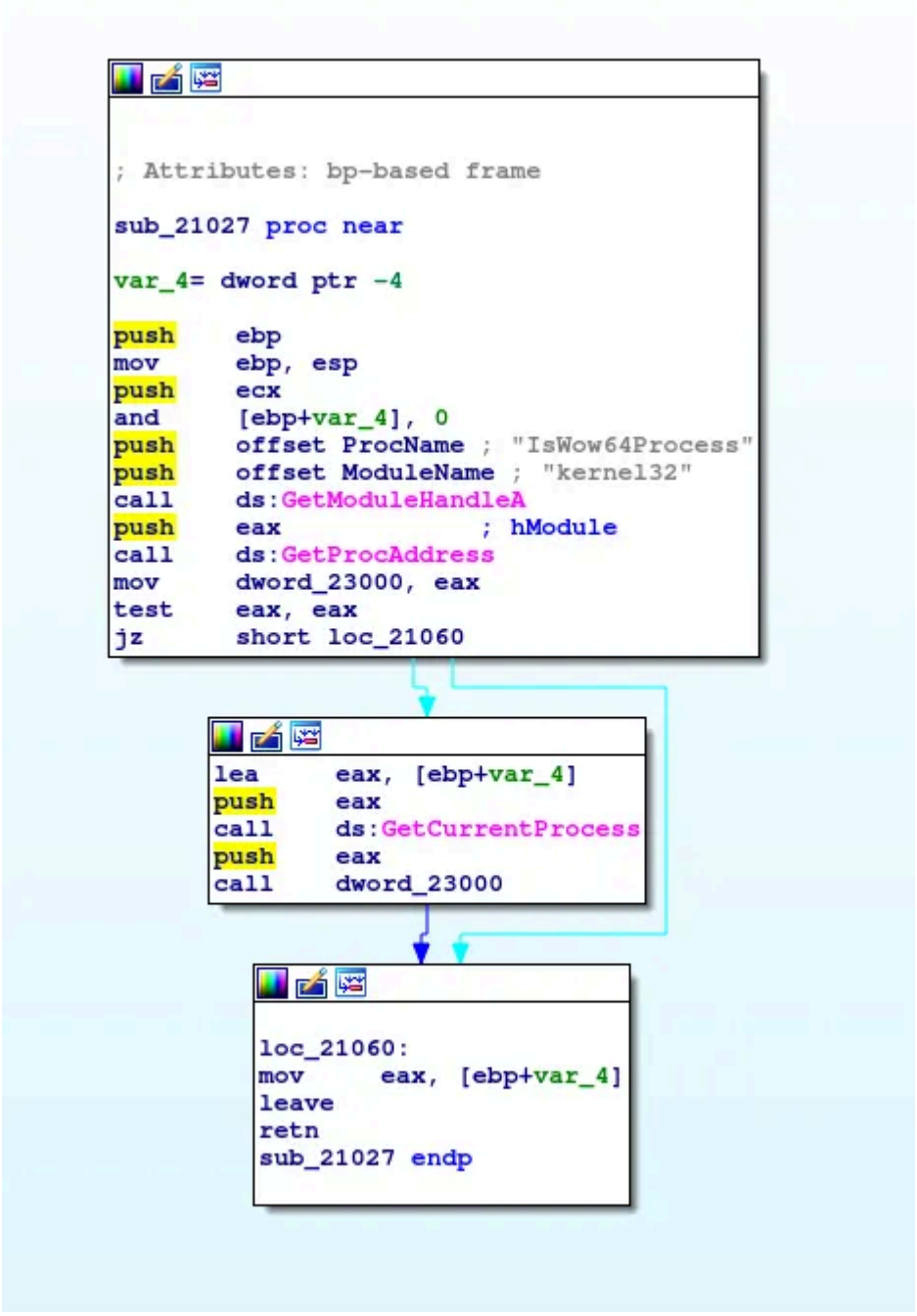
Press enter or click to view image in full size

Press enter or click to view image in full size

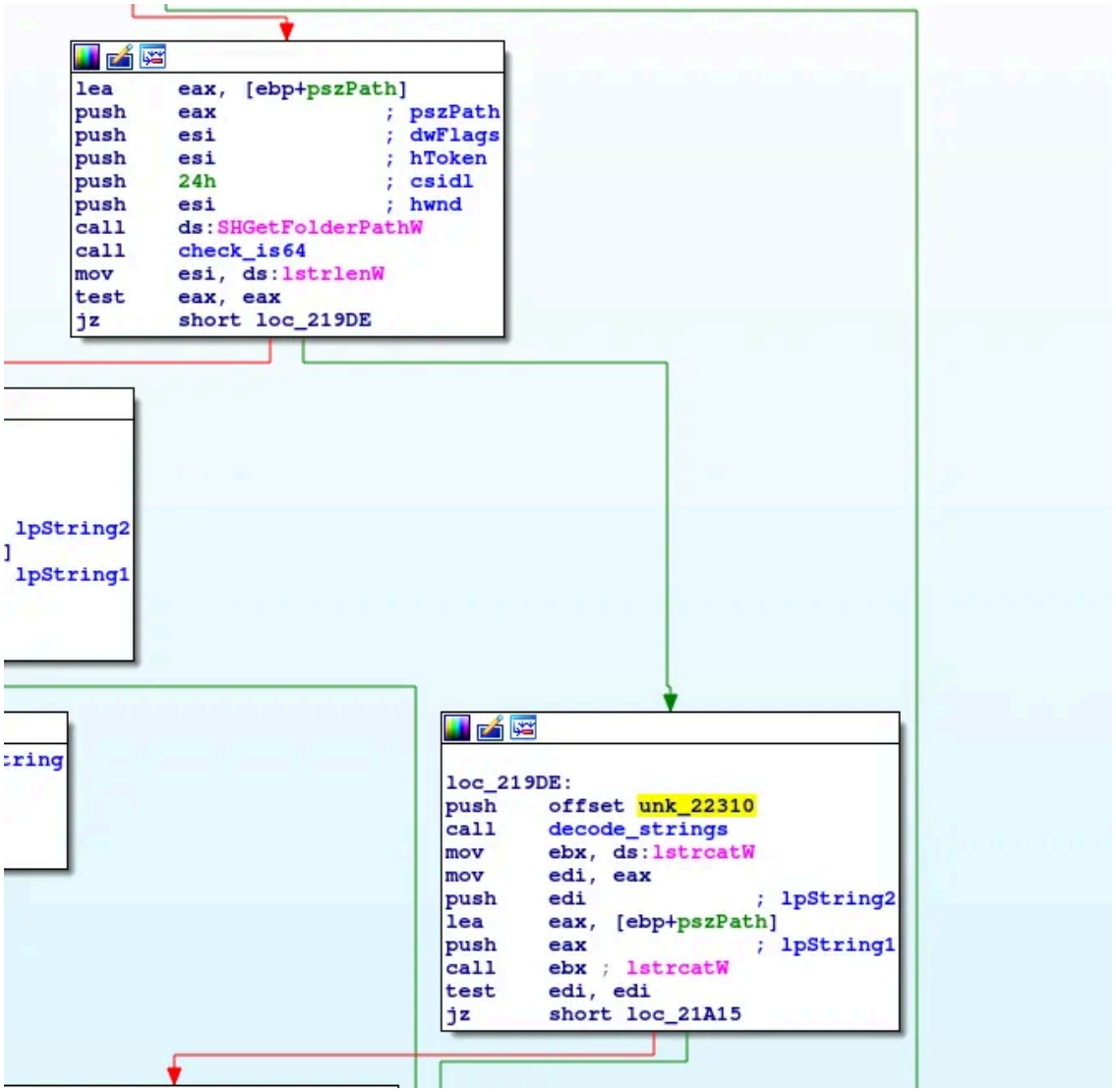


```
; Attributes: noreturn bp-based frame

; void __cdecl __noreturn start(LPCVOID lpBuffer, DWORD nNumberOfBytesToWrite)
public start
start proc near

String1= word ptr -850h
pszPath= word ptr -420h
var_10= dword ptr -10h
lpExistingFileName= dword ptr -0Ch
lpMem= dword ptr -8
lpDirectory= dword ptr -4
lpBuffer= dword ptr  8
nNumberOfBytesToWrite= dword ptr  0Ch

push    ebp
mov     ebp, esp
sub     esp, 850h
push    ebx
push    esi
xor     esi, esi
push    edi
mov     [ebp+var_10], esi
mov     [ebp+lpExistingFileName], esi
mov     [ebp+lpDirectory], esi
mov     [ebp+lpMem], esi
call    checkmutex
test    eax, eax
jz      loc_21B47
```

If it’s ok, drop checks if it’s executed on a 64bits systems to correctly set the parameters of decoding strings



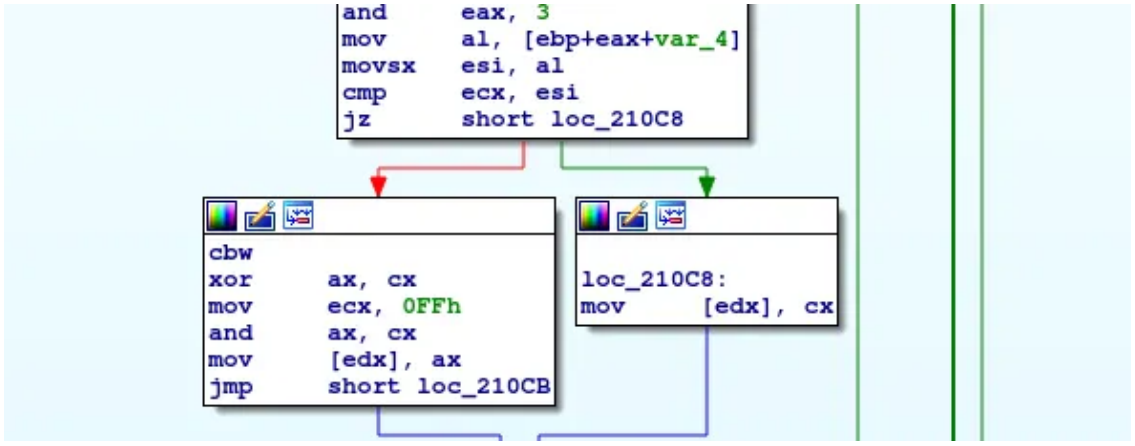
Drop decodes the first string depending on if it’s 64bits or not.



offset unk_ is the key of the decode functions.

The function is a XOR with a and with FF.

The data to decrypt is in drop ressources.



The result is the stored the result in a buffer.

The result of the first decoding function is \\System32.

Press enter or click to view image in full size

rundll32.exe is the result of the second decoding.

Press enter or click to view image in full size

drop drops the backdoor after decoding strings to set in which folder the backdoor is dropped.

in first it decodes the pattern of the path L”%lS\\%lS\\%lS.dbf”

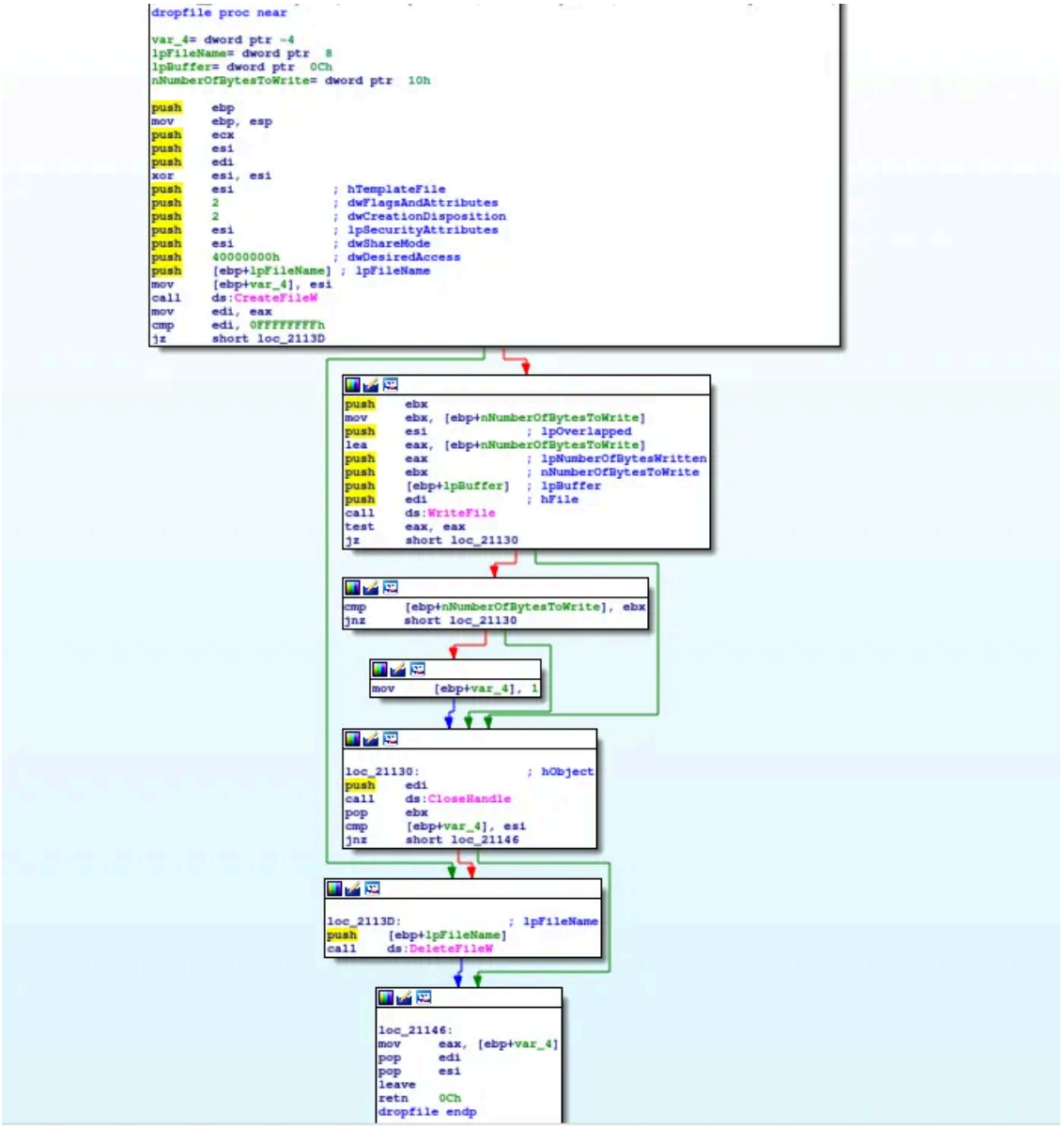
It chooses the special folder: %APPDATA%/Roaming/Microsoft/

In 32bits, this folder is not used so it’s very easy to hide a malware.

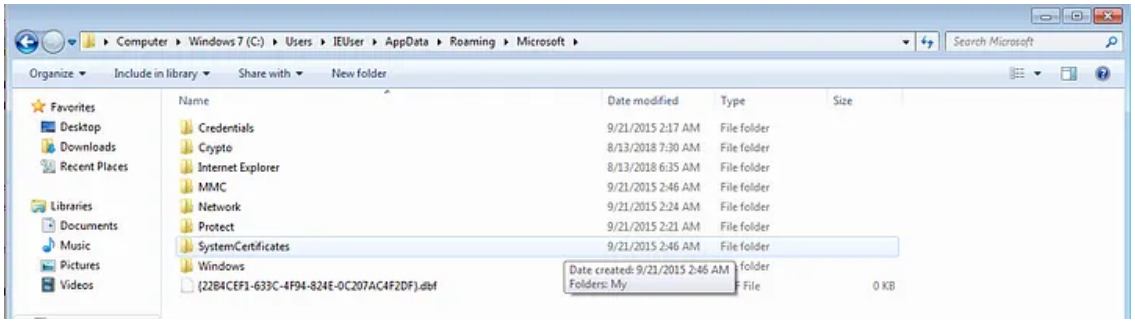
And the path of file is: ”C:\\Users\\IEUser\\AppData\\Roaming\\Microsoft\\{22B4CEF1–633C-4F94–824E-0C207AC4F2DF}.dbf”

The name of file changes at each execution.

drop writes the backoor in the disk from 378C38in sub 210E5.



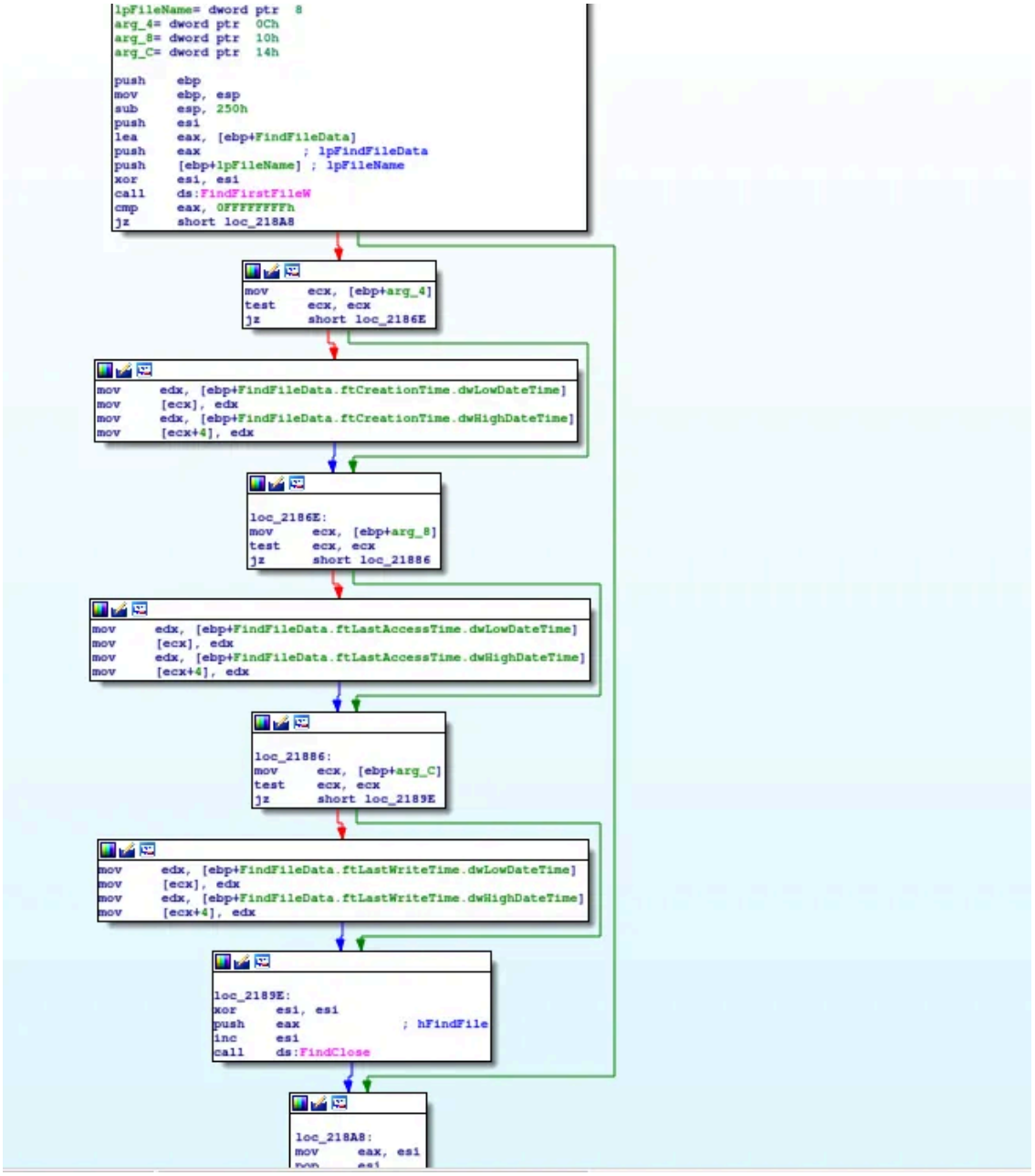
Press enter or click to view image in full size



drop decodes a string to have the path: c:\\windows\\system32\\msvcrt.dll

in sub_00021845, drop catches the creation date of c:\\windows\\system32\\msvcrt.dll

Press enter or click to view image in full size



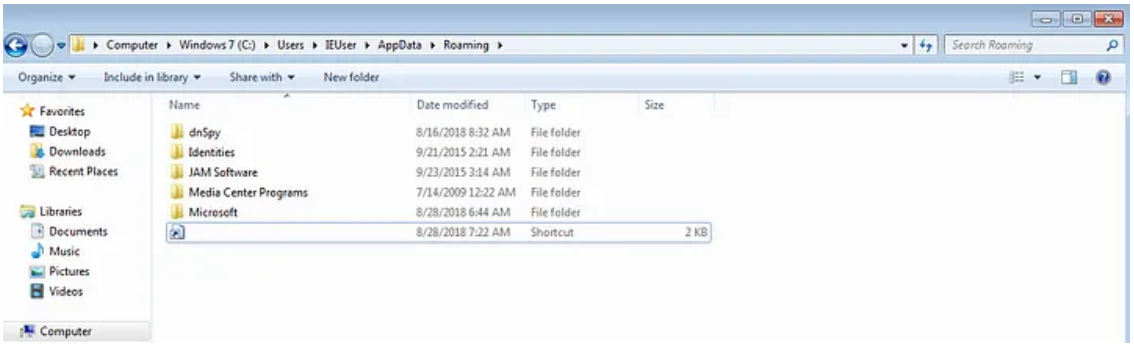
drop changes the attributes of the timestamps (creation date of the file, last modified…) of the backdoor switching the value with L”c:\\windows\\system32\\msvcrt.dll” in sub_000218AF

Press enter or click to view image in full size

the persistance is created. The first step is the decoding of the name of file in 00021A1A. the result is .lnk

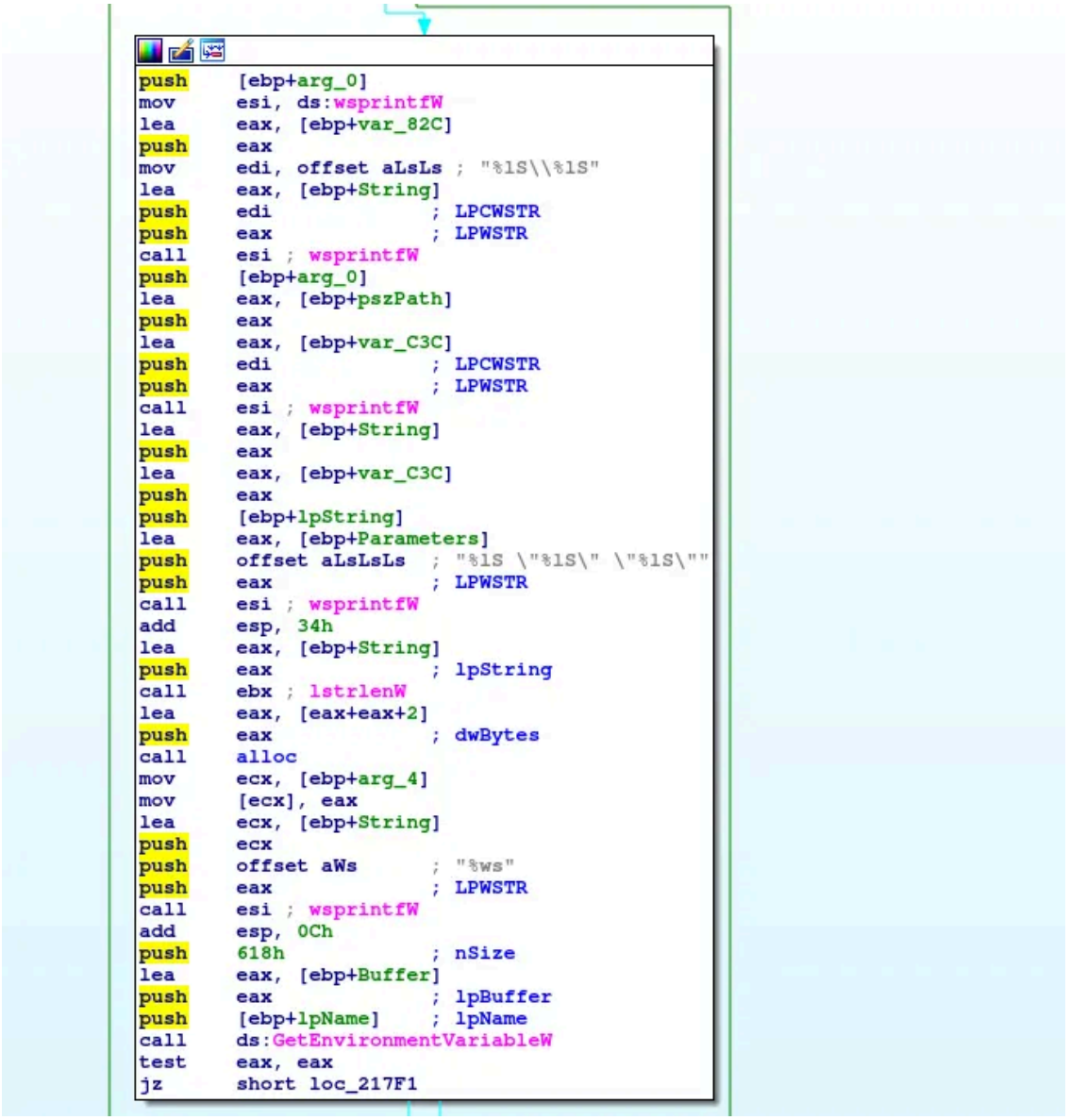
After ther create the persistance of the backdoor in sub_000211D7. It’s a shortcut installed in startup folder and create a copy in roaming folder.

Press enter or click to view image in full size



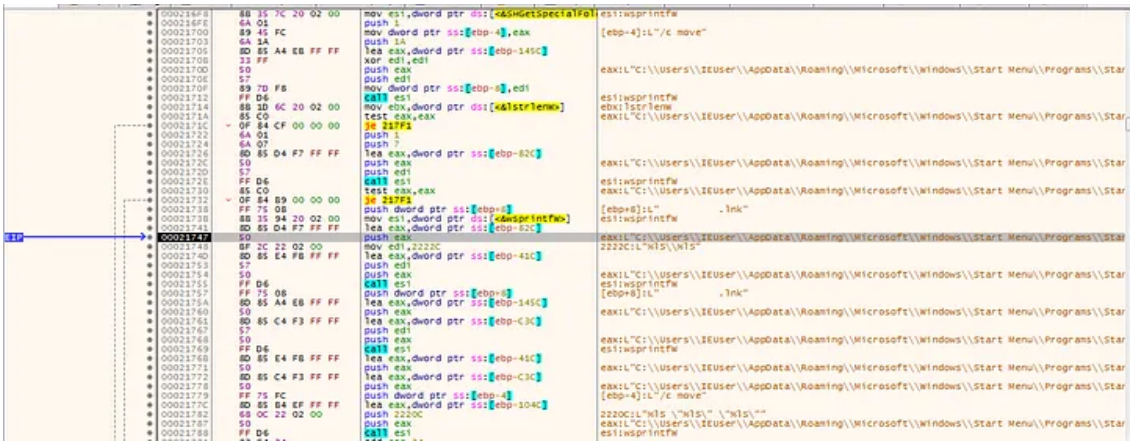
in sub_216D1, drop installs the shortcut in the Startup Folder and copies the shorcut and execute with the function ShellExecute.

0012DC84 0012DCA0 L”C:\\Windows\\system32\\cmd.exe”
0012DC88 0012E6C8 L”/c move \”C:\\Users\\IEUser\\AppData\\Roaming\\ .lnk\”
\”C:\\Users\\IEUser\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\ .lnk\””

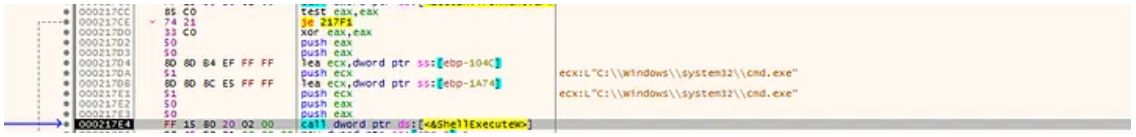


```
push    [ebp+arg_0]
mov     esi, ds:wsprintfW
lea     eax, [ebp+var_82C]
push    eax
mov     edi, offset aLsLs ; "%ls\\%ls"
lea     eax, [ebp+String]
push    edi             ; LPCWSTR
push    eax             ; LPWSTR
call    esi ; wsprintfW
push    [ebp+arg_0]
lea     eax, [ebp+pszPath]
push    eax
lea     eax, [ebp+var_C3C]
push    edi             ; LPCWSTR
push    eax             ; LPWSTR
call    esi ; wsprintfW
lea     eax, [ebp+String]
push    eax
lea     eax, [ebp+var_C3C]
push    eax
push    [ebp+lpString]
lea     eax, [ebp+Parameters]
push    offset aLsLsLs  ; "%ls \"%ls\" \"%ls\""
push    eax             ; LPWSTR
call    esi ; wsprintfW
add     esp, 34h
lea     eax, [ebp+String]
push    eax             ; lpString
call    ebx ; lstrlenW
lea     eax, [eax+eax+2]
push    eax             ; dwBytes
call    alloc
mov     ecx, [ebp+arg_4]
mov     [ecx], eax
lea     ecx, [ebp+String]
push    ecx
push    offset aWs      ; "%ws"
push    eax             ; LPWSTR
call    esi ; wsprintfW
add     esp, 0Ch
push    618h            ; nSize
lea     eax, [ebp+Buffer]
push    eax             ; lpBuffer
push    [ebp+lpName]    ; lpName
call    ds:GetEnvironmentVariableW
test    eax, eax
jz      short loc_217F1
```

Press enter or click to view image in full size



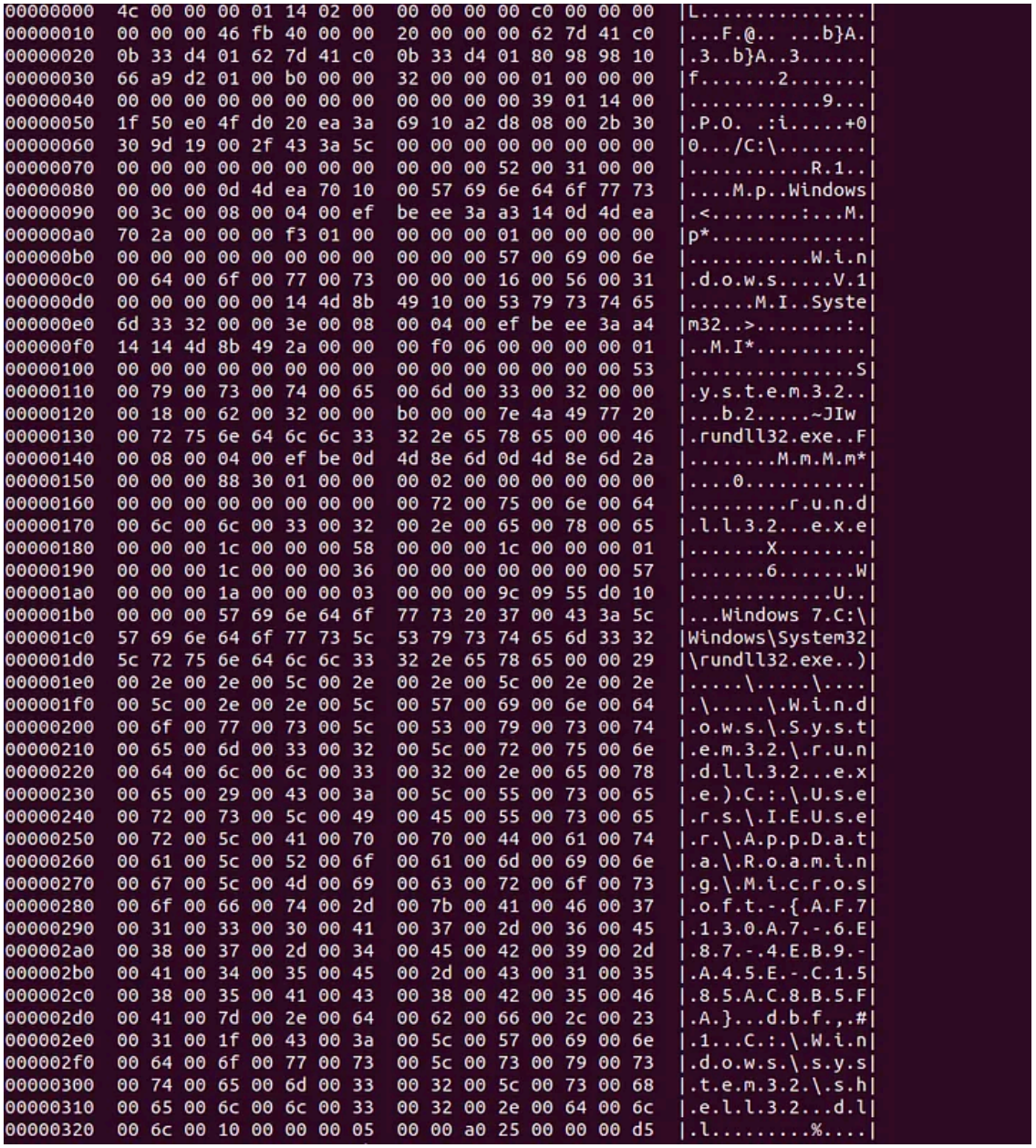
Press enter or click to view image in full size



Shortcut to restart

The lnk runs run32dll.exe with the .dbf and ordinal 1 of the export of the dll.

Press enter or click to view image in full size



Few words about Threat Intel

It’s strange that FireEye hasn’t published a full paper with the analysis of the dropper and the backdoor.

The dropper uses many very interesting techniques:

- rc4 encryption
- decompression function custom
- run32dll to bypass applocker and AV
- change timestamps of the installed files
- decrypting strings to install the backdoor
- decrypting strings for the persistance settings

The function sub_406681 is very interesting and it’s very difficult to make a yara rules on it because there are many jump to have enough binaries to make a rule. the rc4 encryption and a decompression function are in this function.

Thanks

thank to the *zone de confort* to try to understand this f… decompression function and thank to @FliegenEinhorn to find the sample !
