

# 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 persistence (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	2744483F2BCD0FE11A2DA76F7E78086376FFFE3
sha256	9DAB5492B69EC34E62388BBE2BE0909FE11C4EACB716F16EF6868F04B2120434
entropy	7.994
first-bytes (hex)	1B 73 B4 17 5E 5F 14 99 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.

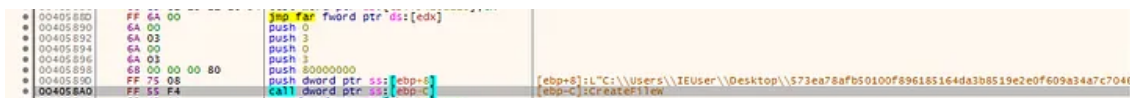
```
overlay = open('573ea78afb50100f896185164da3b8519e2e0f609a34a7c70460eca5b4ae640d', 'rb').read()[0xD800:]
```

```
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

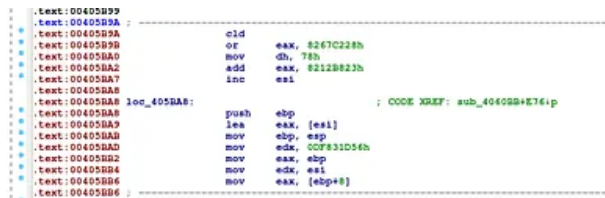


### Read itself

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.

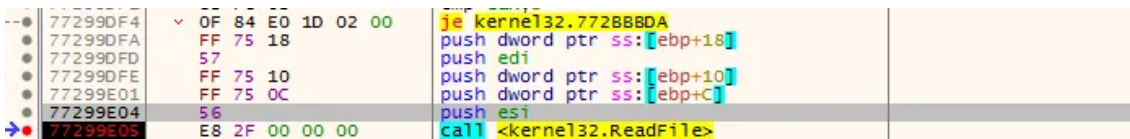
Press enter or click to view image in full size



### 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)



```

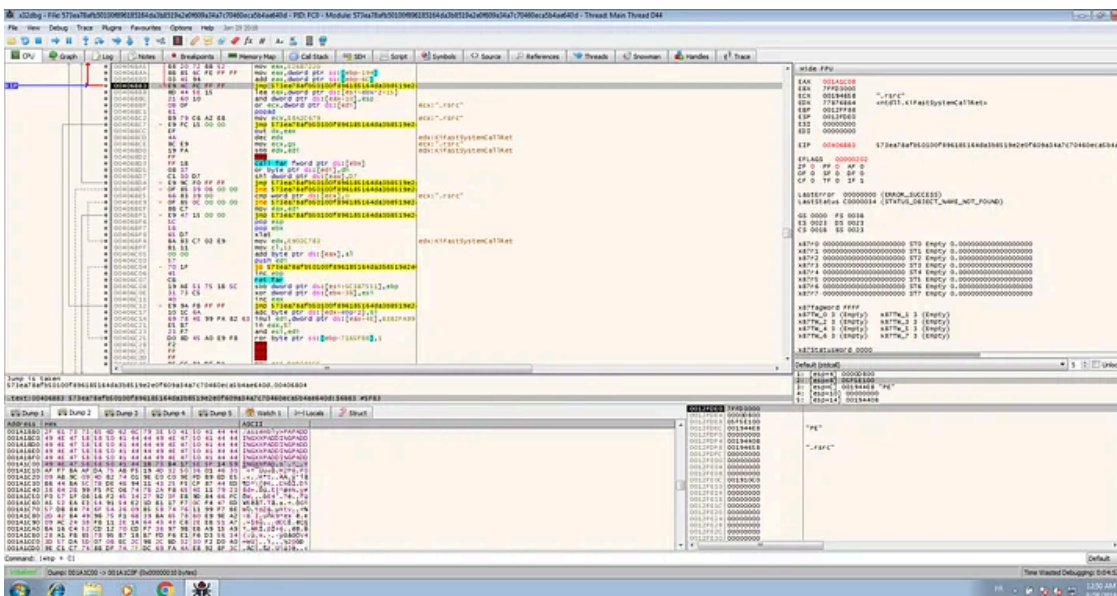
00194400 53 A8 57 28 63 18 00 18 4D 5A 90 00 03 00 00 00 5'w(c...MZ.....
00194410 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 .....yy.....
00194420 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 @.....
00194430 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00194440 00 00 00 00 E0 00 00 00 0E 1F BA 0E 00 B4 09 CD .....a...o..i
00194450 21 B8 01 4C CD 21 54 68 69 73 20 70 72 6F 67 72 !.Li!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 0C 73 2D 1F $......CLAs-
00194490 0C 73 2D 1F C0 73 2D 1F DE 21 A9 1F E3 73 2D 1F As-.As-.p!@.as-
001944A0 DE 21 B8 1F D0 73 2D 1F DE 21 AE 1F A0 73 2D 1F p!..@s-.p!@.s-
001944B0 E7 B5 56 1F C7 73 2D 1F C0 73 2C 1F B2 73 2D 1F çuV.Çs-.As..s-
001944C0 DE 21 A7 1F C1 73 2D 1F DE 21 BC 1F C1 73 2D 1F p!ç.Às-.p!%.As-
001944D0 52 69 63 68 C0 73 2D 1F 00 00 00 00 00 00 00 00 RichAs-
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 00 00 03 01 xé.O.....à...
00194500 08 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

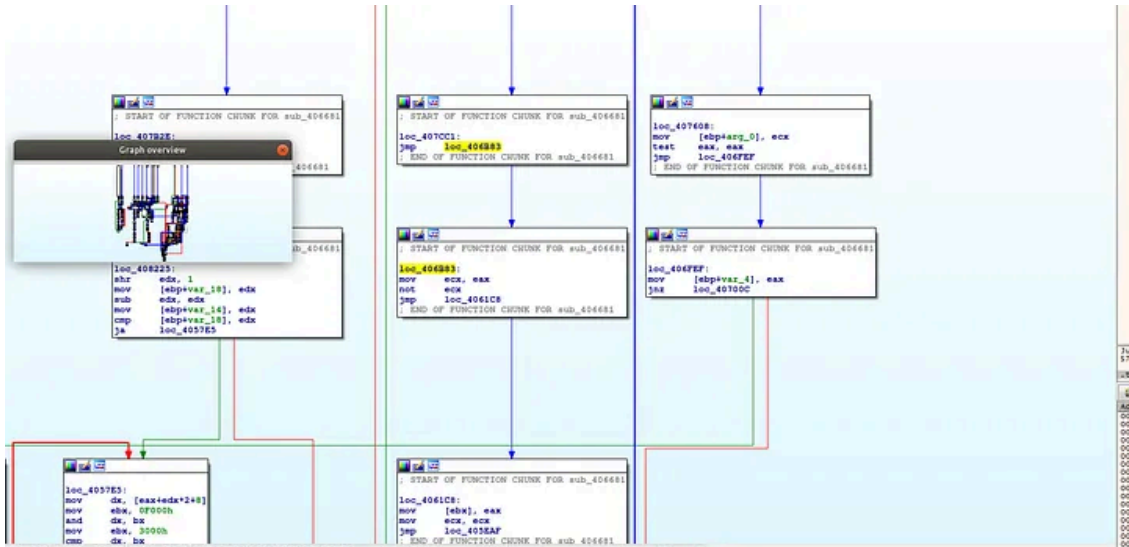
Press enter or click to view image in full size



Overlay in memory

if we check in IDA where we are.

Press enter or click to view image in full size



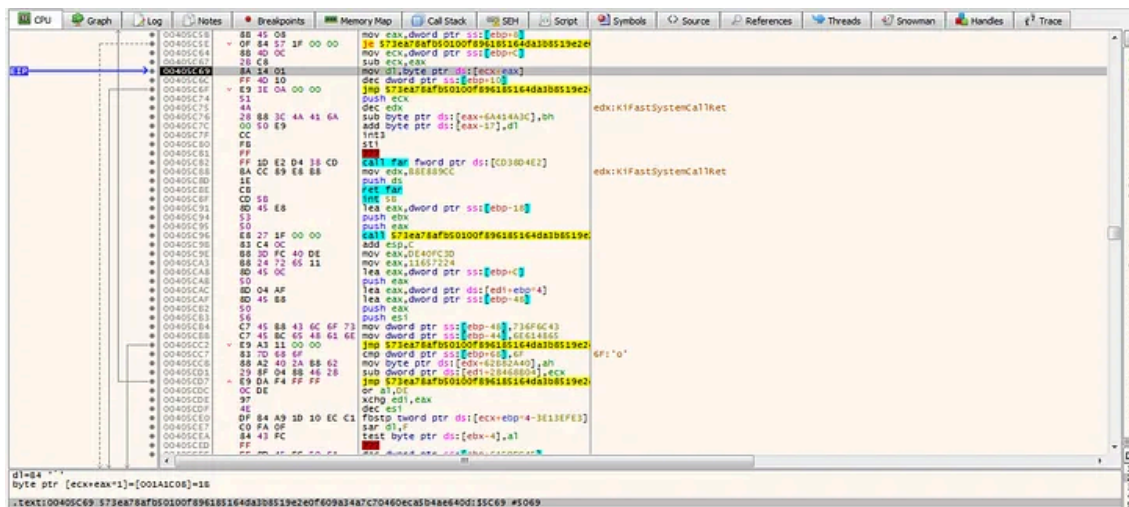
sub\_406681

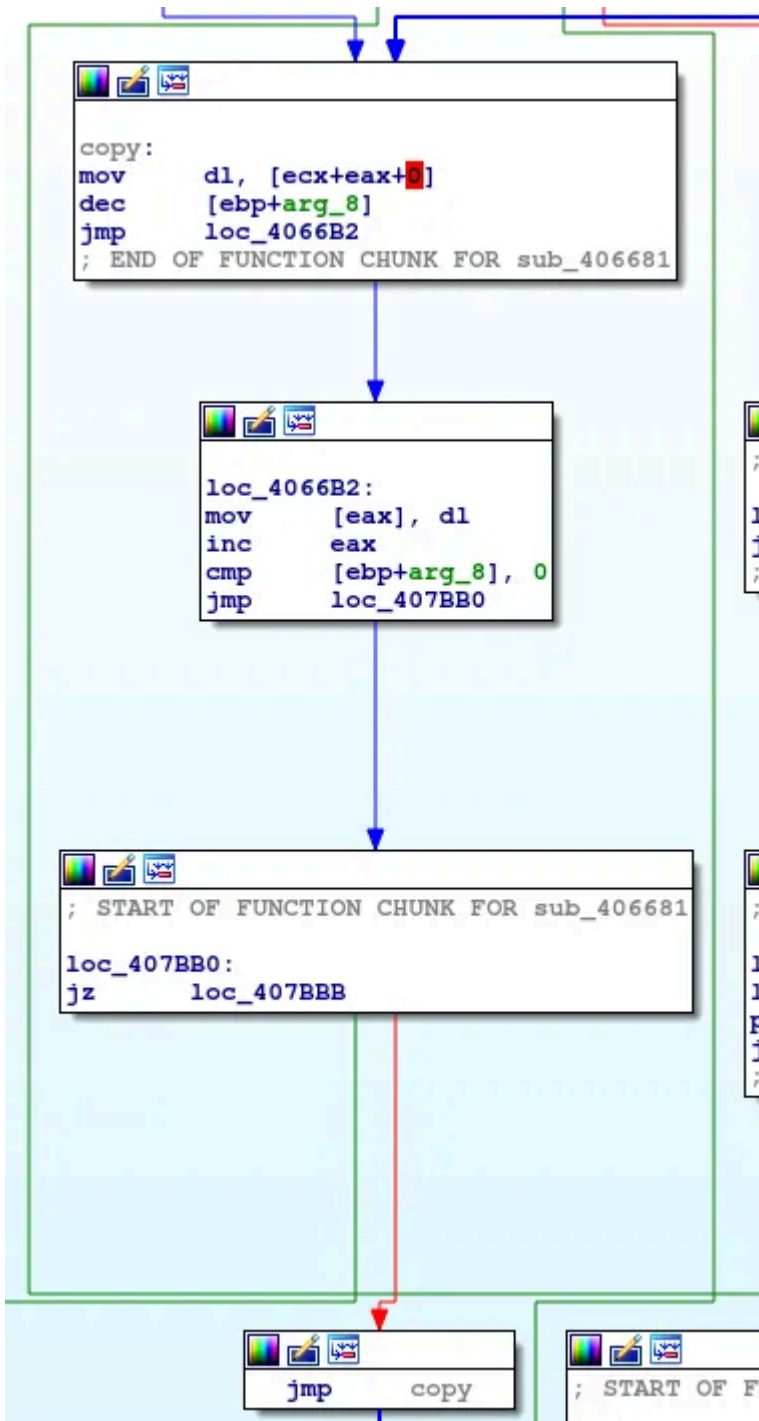
If we check the graph overview, this function is in a huge function with many jmp.

It seems this function is like a packer.

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

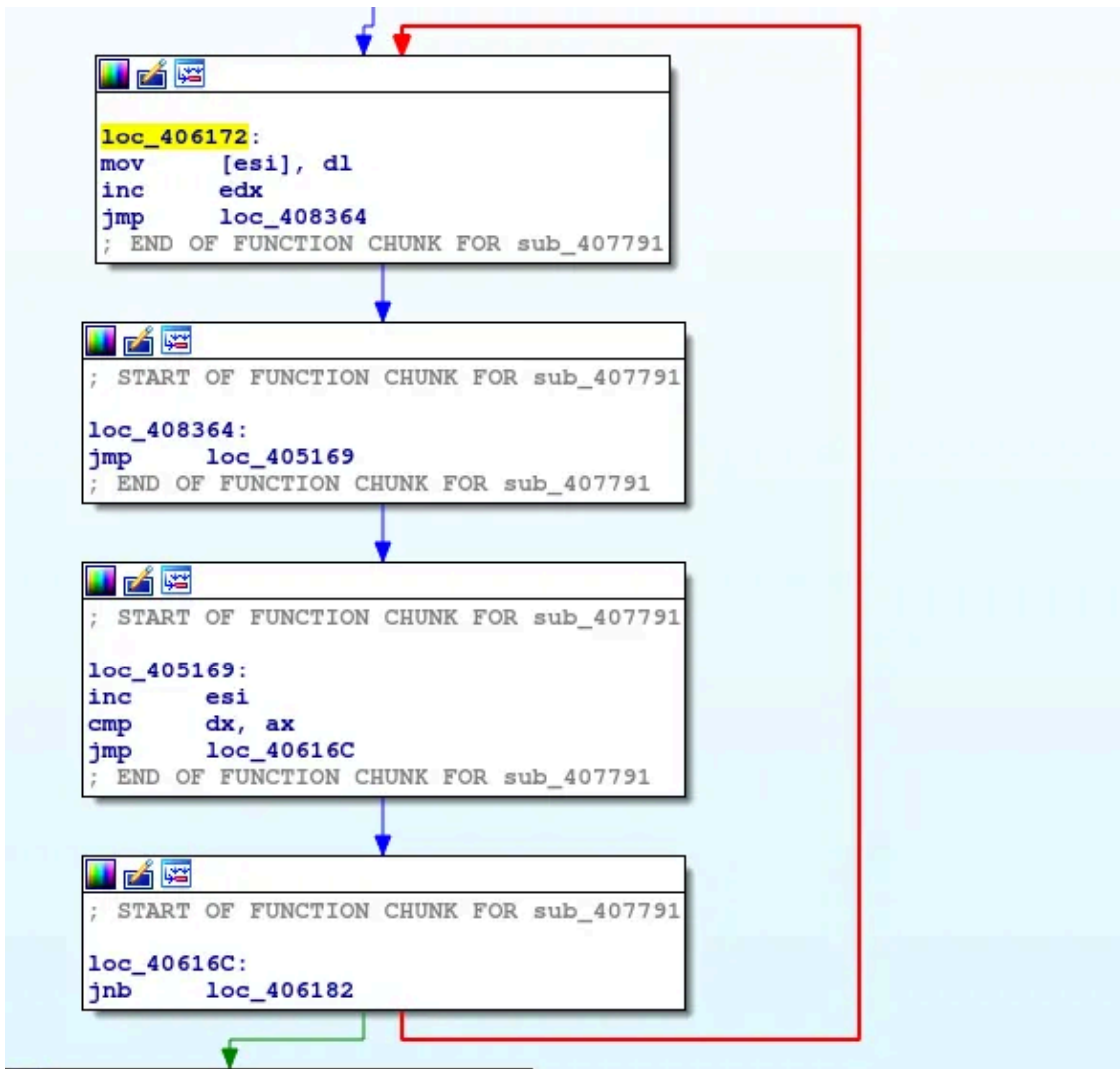
Press enter or click to view image in full size





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

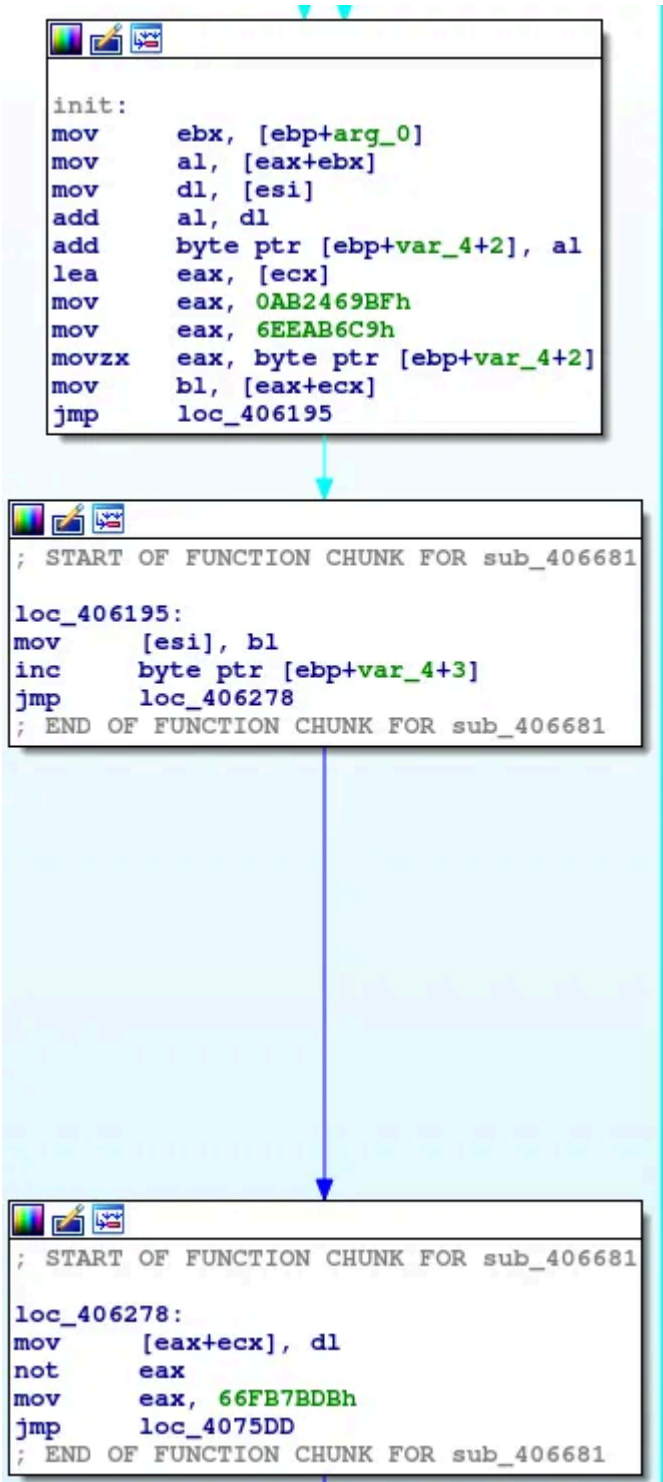
After reading the 40 bits, we have a loop of 256 steps, to store 01 to 256 on the stack.



init of RC4

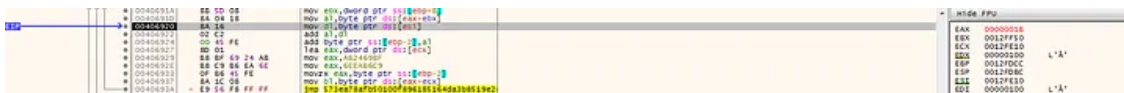
0012FE10	03020100
0012FE14	07060504
0012FE18	0B0A0908
0012FE1C	0F0E0D0C
0012FE20	13121110
0012FE24	17161514
0012FE28	1B1A1918
0012FE2C	1F1E1D1C
0012FE30	23222120
0012FE34	27262524
0012FE38	2B2A2928
0012FE3C	2F2E2D2C
0012FE40	33323130
0012FE44	37363534
0012FE48	3B3A3938
0012FE4C	3F3E3D3C
0012FE50	43424140
0012FE54	47464544

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



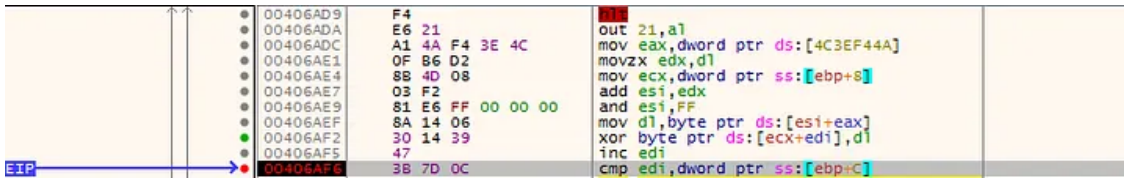
Init of RC4

Press enter or click to view image in full size



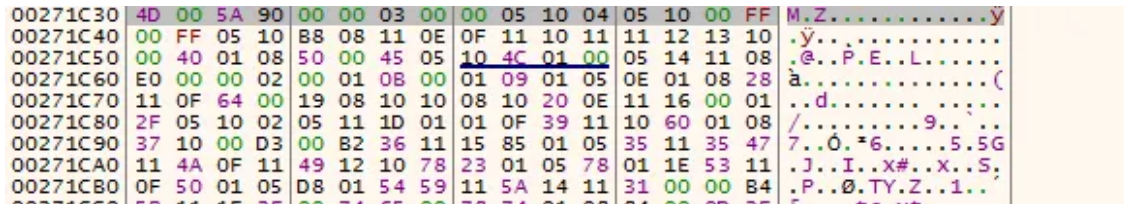
And a third function with arithmetic operations works on the overlay with the results of the two lasted functions

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 comparison 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
```

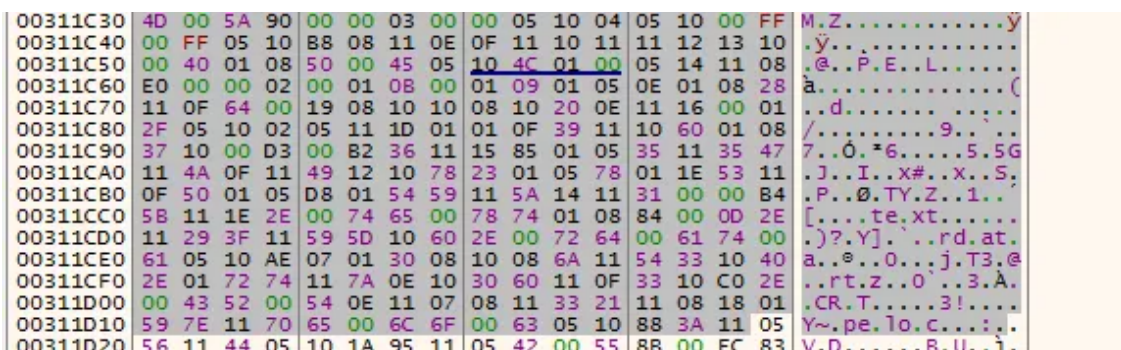
And the third function is:

```
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]) & 0xFF]) & 0xFF)
    return ''.join(output)
```

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

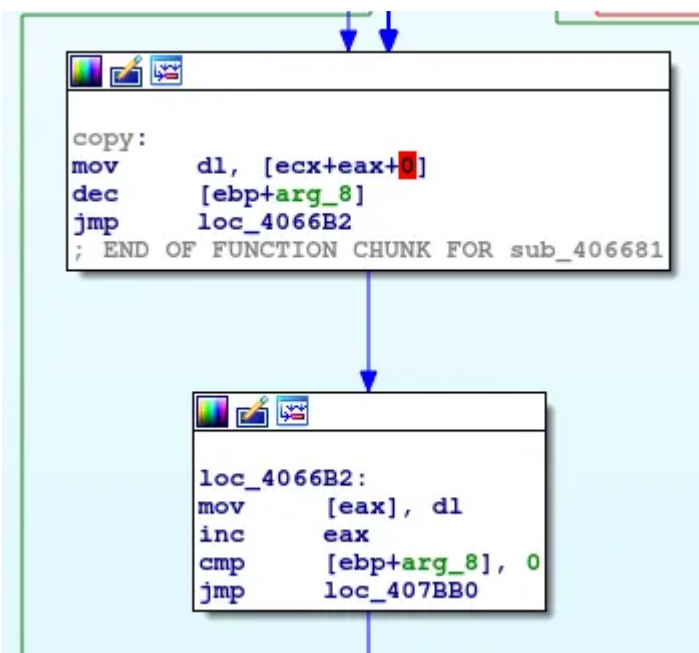
1B 73 B4 17 5E 5F 14 59 AF F7 BA AF DA 75 AB F5 19 4D 32 50 36 01 46 30 09 AB 9C 09 4D B2 74 01 9E C0 C0 9E FD B9 ED E5

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

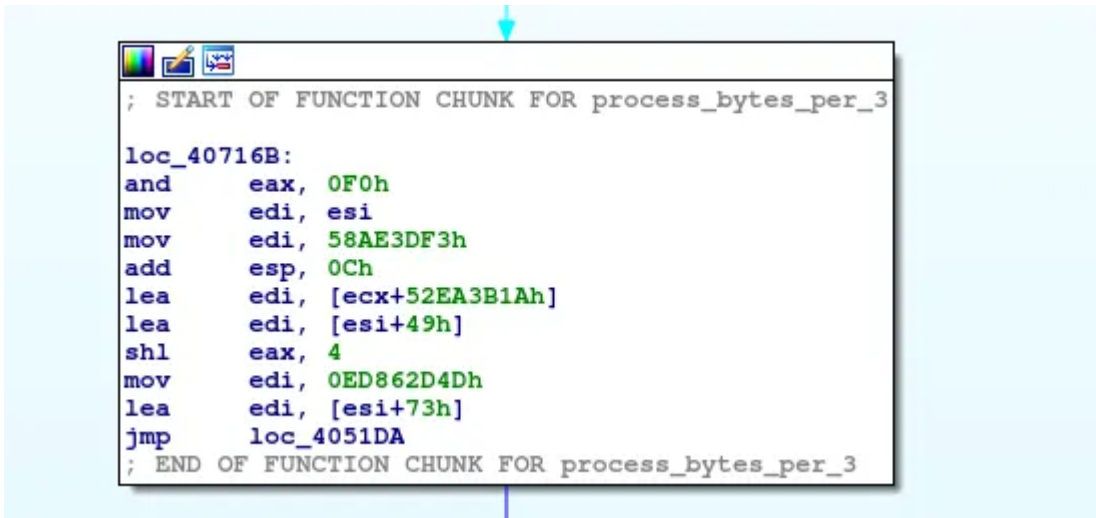


After that, the dropper increases the stack from 12F000 to 12C000 before launching the decompression function (seemly custom after many searches, but if it's not that, write a comment at the end of this post !)

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

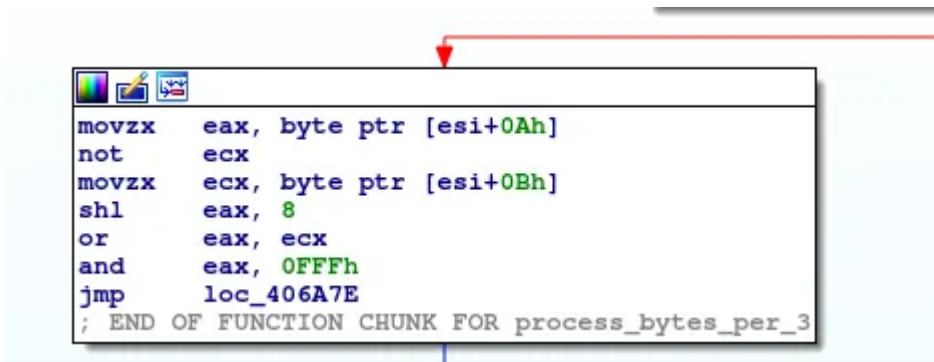


The algorithm used in the first AB,C and secondly D, EF in this function sub\_004055FC.



```
; START OF FUNCTION CHUNK FOR process_bytes_per_3  
  
loc_40716B:  
and    eax, 0F0h  
mov    edi, esi  
mov    edi, 58AE3DF3h  
add    esp, 0Ch  
lea    edi, [ecx+52EA3B1Ah]  
lea    edi, [esi+49h]  
shl    eax, 4  
mov    edi, 0ED862D4Dh  
lea    edi, [esi+73h]  
jmp    loc_4051DA  
; END OF FUNCTION CHUNK FOR process_bytes_per_3
```

AB,C



```
movzx  eax, byte ptr [esi+0Ah]  
not    ecx  
movzx  ecx, byte ptr [esi+0Bh]  
shl    eax, 8  
or     eax, ecx  
and    eax, 0FFFh  
jmp    loc_406A7E  
; END OF FUNCTION CHUNK FOR process_bytes_per_3
```

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

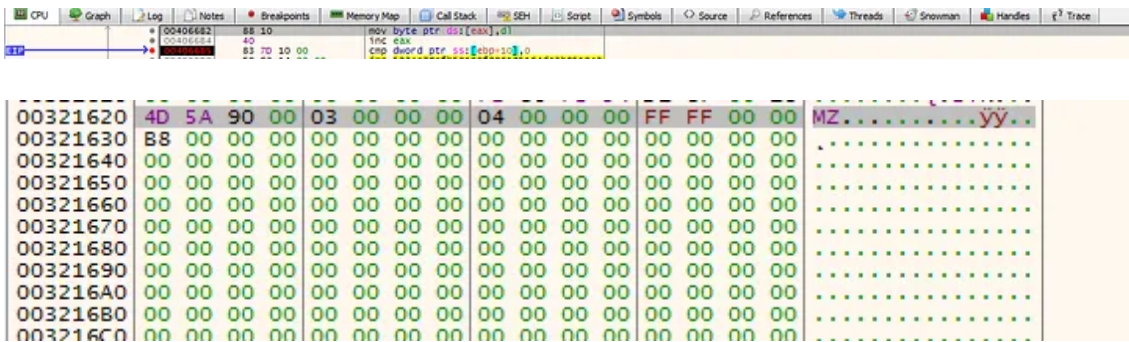
90 00 00 -> 90 00

03 00 00 -> 03 00

51 10 04 -> 00 00 04

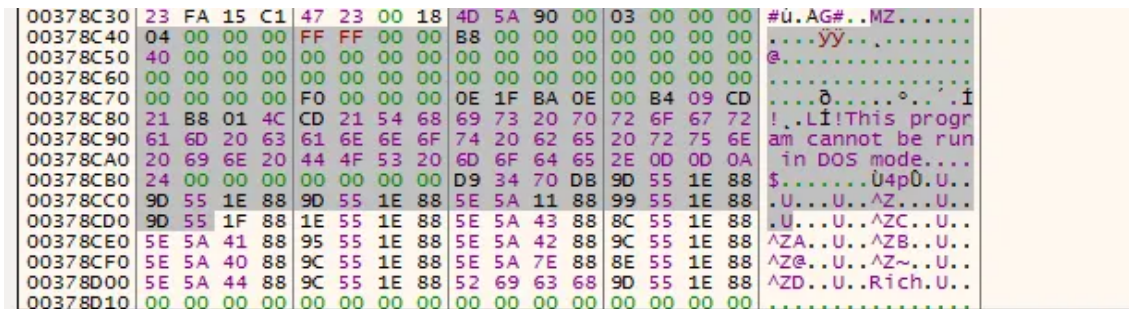
The result is stored in a buffer. The address of the buffer is stored by EAX.

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 copied to 378C38.

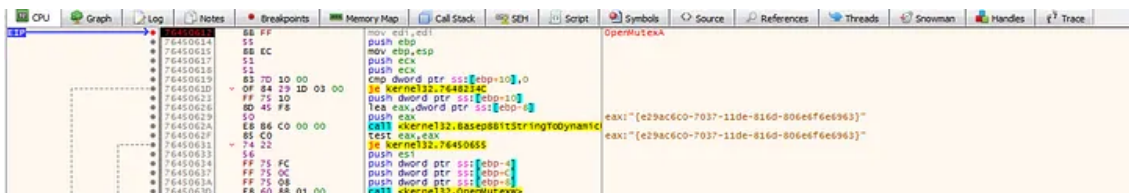


drop is copied after a Virtualloc at 20000 and drop is executed in 00021964.

## 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
```

```
; Attributes: bp-based frame
checkmutex proc near
HwProfileInfo= tagHW_PROFILE_INFOA ptr -7Ch
push    ebp
mov     ebp, esp
sub     esp, 7Ch
push    esi
lea    eax, [ebp+HwProfileInfo]
xor    esi, esi
push    eax                ; lpHwProfileInfo
inc    esi
call   ds:GetCurrentHwProfileA
test   eax, eax
jz     short loc_216CC
```

```
lea    eax, [ebp+HwProfileInfo.szHwProfileGuid]
push   eax                ; lpName
push   0                  ; bInheritHandle
push   100000h            ; dwDesiredAccess
call   ds:OpenMutexA
test   eax, eax
jz     short loc_216CC
```

```
xor    esi, esi
```

```
loc_216CC:
mov    eax, esi
pop    esi
leave
retn
checkmutex endp
```

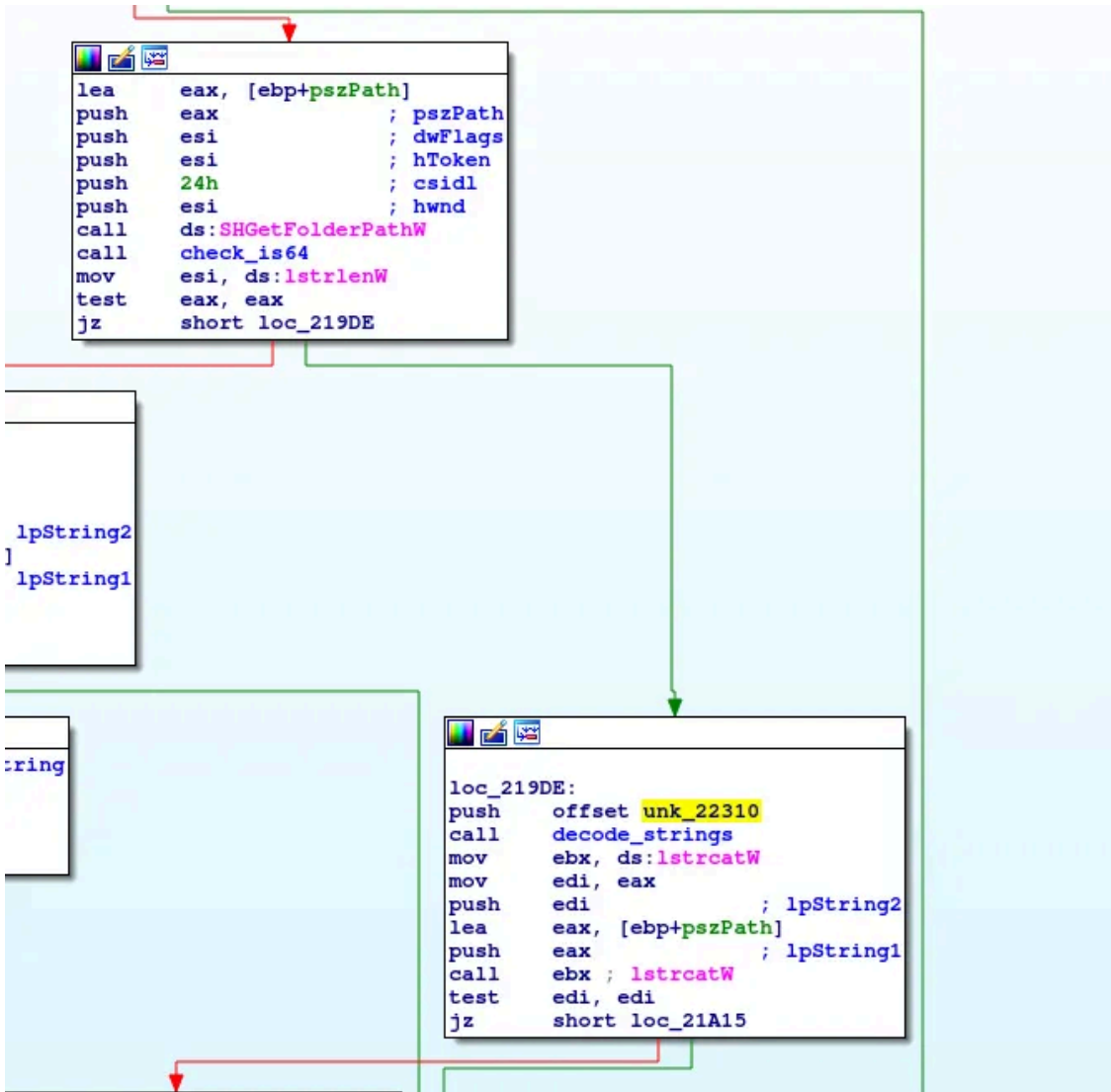
If it's ok, drop checks if it's executed on a 64bits systems to correctly set the parameters of decoding strings

```
; Attributes: bp-based frame
sub_21027 proc near
var_4= dword ptr -4
push    ebp
mov     ebp, esp
push    ecx
and     [ebp+var_4], 0
push    offset ProcName ; "IsWow64Process"
push    offset ModuleName ; "kernel32"
call    ds:GetModuleHandleA
push    eax ; hModule
call    ds:GetProcAddress
mov     dword_23000, eax
test    eax, eax
jz     short loc_21060
```

```
lea    eax, [ebp+var_4]
push    eax
call    ds:GetCurrentProcess
push    eax
call    dword_23000
```

```
loc_21060:
mov     eax, [ebp+var_4]
leave
retn
sub_21027 endp
```

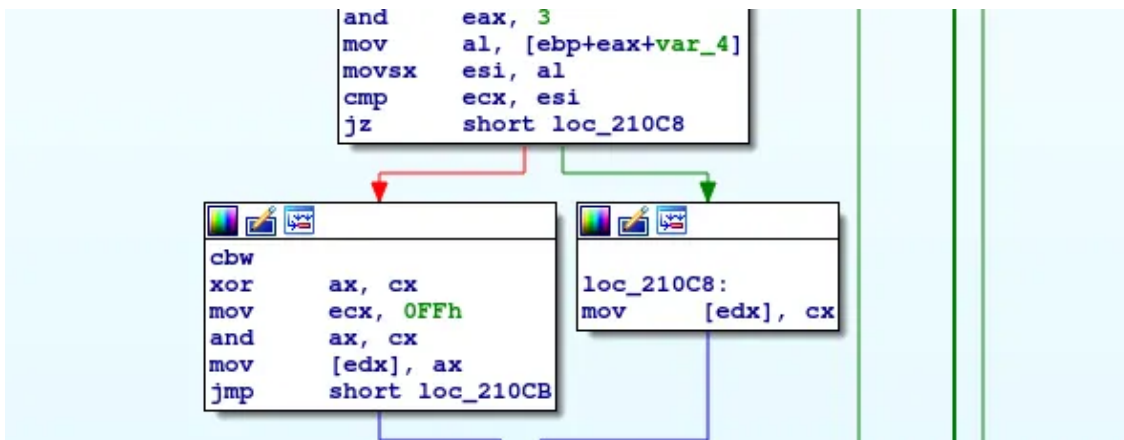
Drop decodes the first string depending on if it's 64bits or not.



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"%IS\\%IS\\%IS.dbf"

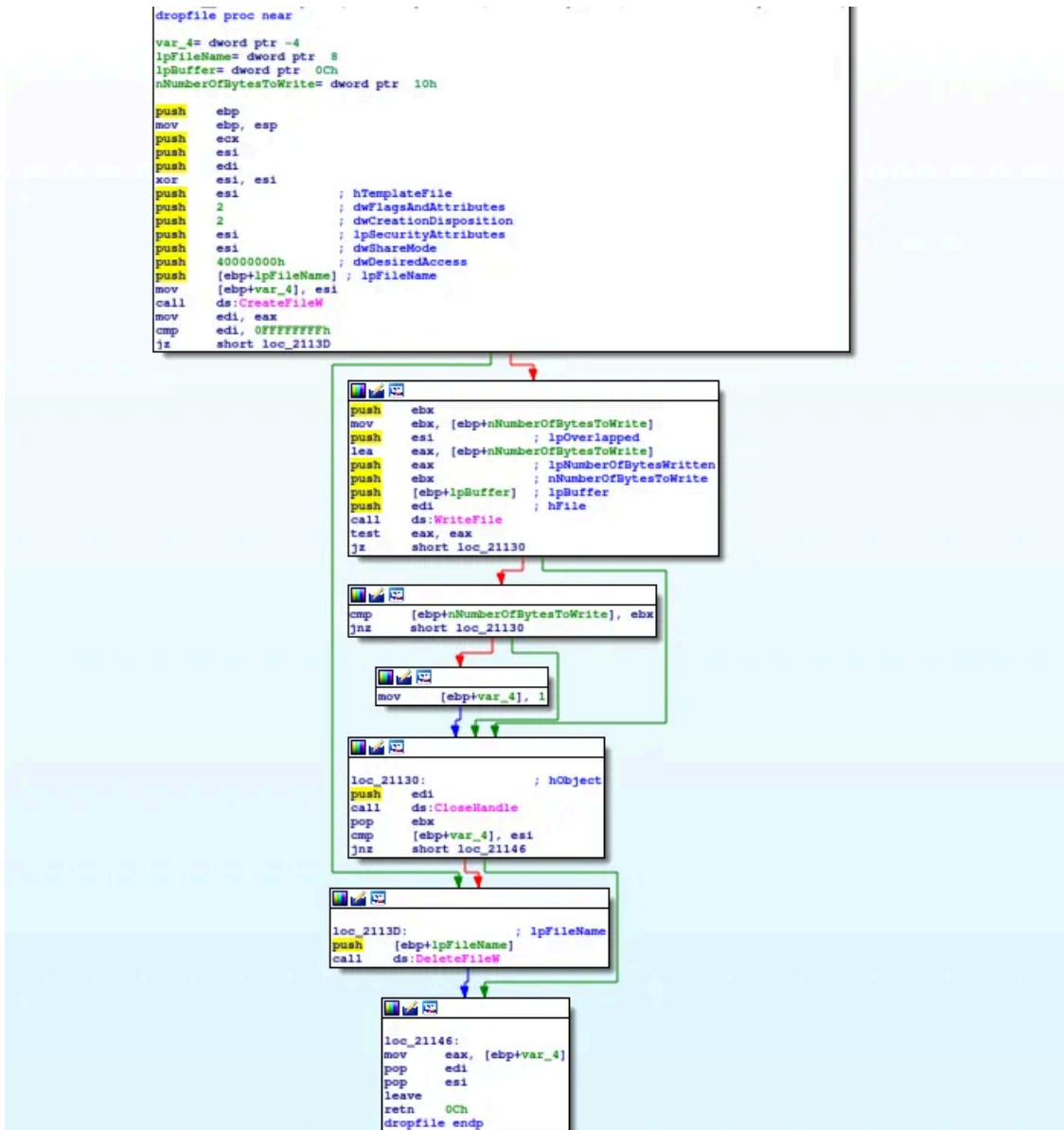
It chooses the special folder: %APPDATA%/Roaming/Microsoft/

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

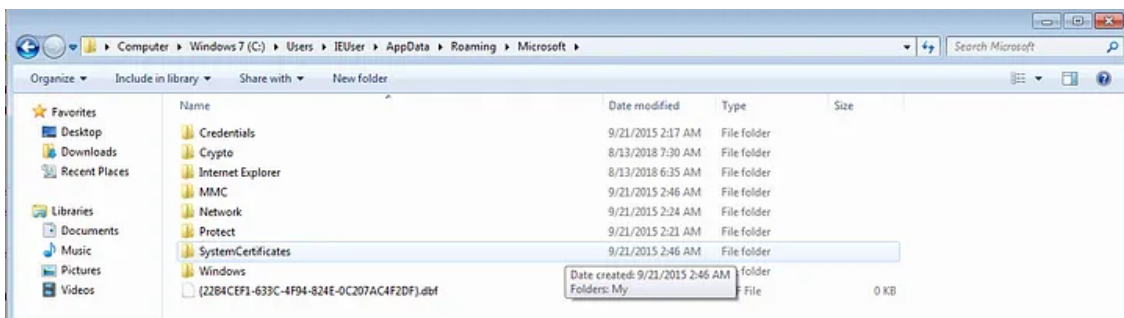
And the path of file is: "C:\\Users\\IEUser\\AppData\\Roaming\\Microsoft\\{22B4CEF1-633C-4F94-824E-0C207AC4F2DF}.dbf"

The name of file changes at each execution.

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



Press enter or click to view image in full size

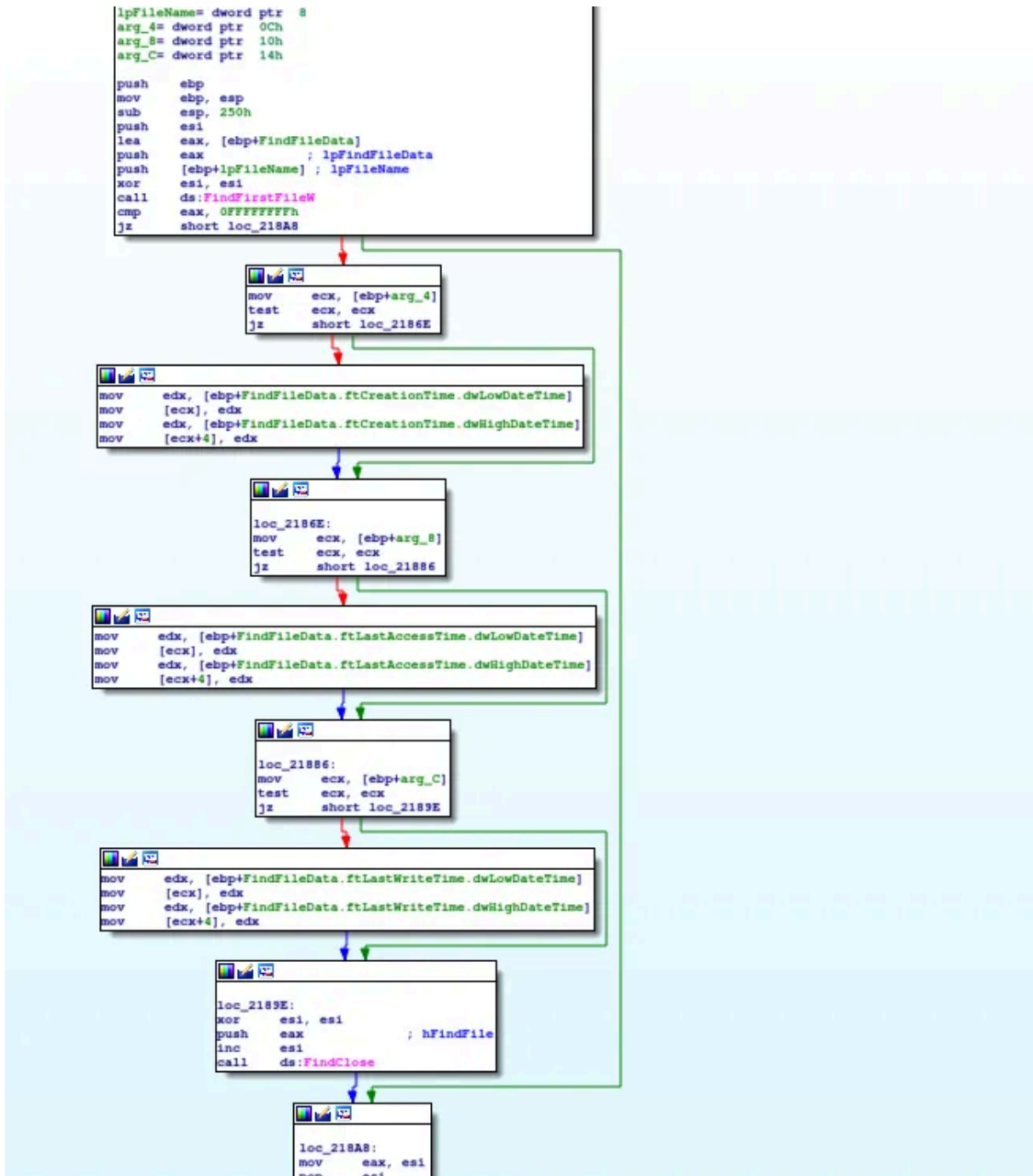


drop decodes a string to have the path: c:\windows\system32\msvcrt.dll



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

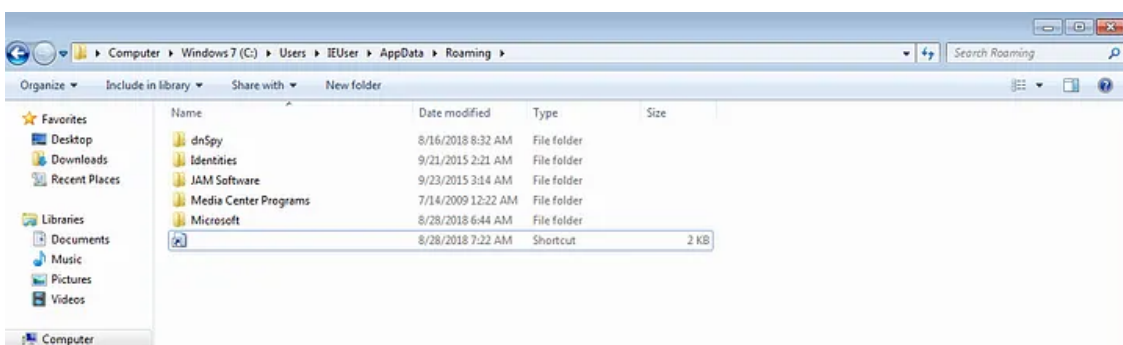
Press enter or click to view image in full size



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

After that create the persistence 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 shortcut 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"
```

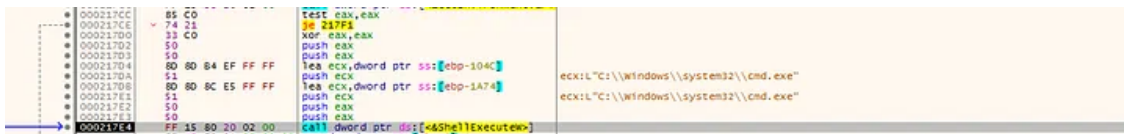
```
\\C:\\Users\\IEUser\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\.lnk"
```

```
push [ebp+arg_0]
mov esi, ds:wsprintfW
lea eax, [ebp+var_82C]
push eax
mov edi, offset aLsLs ; "%1S\\%1S"
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 ; "%1S \"%1S\" \"%1S\""
push eax ; LPWSTR
call esi ; wsprintfW
add esp, 34h
lea eax, [ebp+String]
push eax ; lpString
call ebx ; strlenW
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

00021648	8B 35 7C 20 02 00	mov esi, dword ptr ds:[45hGetSpectralFo]	esi:wsprintfw
0002164E	6A 01	push 1	[ebp-4]:"/c move"
00021700	89 45 FC	mov dword ptr ss:[ebp-4], eax	
00021703	6A 1A	push 1A	
00021705	2D 85 A4 E8 FF FF	lea eax, dword ptr ss:[ebp-145C]	
00021706	33 FF	xor edi, edi	
00021707	50	push eax	
00021708	57	push edi	eax:"C:\\Users\\IEUser\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Star
0002170F	89 7D F8	mov dword ptr ss:[ebp-8], edi	
00021712	FF D6	call esi	esi:wsprintfw
00021714	8B 1D 6C 20 02 00	mov ebx, dword ptr ds:[45hstrlenW]	ebx:strlenw
0002171A	85 C0	test eax, eax	eax:"C:\\Users\\IEUser\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Star
0002171C	0F 84 CF 00 00 00	je 217F1	
00021722	6A 01	push 1	
00021724	6A 07	push 7	
00021726	8D 85 04 F7 FF FF	lea eax, dword ptr ss:[ebp-82C]	eax:"C:\\Users\\IEUser\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Star
0002172D	57	push eax	
0002172E	57	push edi	
00021730	FF D6	call esi	esi:wsprintfw
00021732	85 C9	test ebx, eax	ebx:strlenw
00021738	0F 84 89 00 00 00	je 217F1	[ebp-8]:"/c move"
0002173B	FF 75 08	push dword ptr ss:[ebp-8]	esi:wsprintfw
00021738	8B 15 94 20 02 00	mov esi, dword ptr ds:[45hwsprintfW]	esi:wsprintfw
00021738	8D 85 04 F7 FF FF	lea eax, dword ptr ss:[ebp-82C]	eax:"C:\\Users\\IEUser\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Star
00021747	50	push eax	2220:"%1S\\%1S"
00021748	BF 2C 22 02 00	mov edi, 2220	
0002174D	2D 85 E4 F8 FF FF	lea eax, dword ptr ss:[ebp-43C]	
00021753	57	push edi	
00021754	57	push eax	eax:"C:\\Users\\IEUser\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Star
00021755	FF D6	call esi	esi:wsprintfw
00021757	FF 75 08	push dword ptr ss:[ebp-8]	[ebp-8]:"/c move"
00021760	8D 85 A4 E8 FF FF	lea eax, dword ptr ss:[ebp-145C]	
00021761	50	push eax	eax:"C:\\Users\\IEUser\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Star
00021762	8D 85 C4 F3 FF FF	lea eax, dword ptr ss:[ebp-C3C]	
00021767	57	push edi	
00021768	57	push eax	eax:"C:\\Users\\IEUser\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Star
00021769	FF D6	call esi	esi:wsprintfw
0002176B	8D 85 E4 F8 FF FF	lea eax, dword ptr ss:[ebp-43C]	
00021772	57	push edi	eax:"C:\\Users\\IEUser\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Star
00021772	8D 85 C4 F3 FF FF	lea eax, dword ptr ss:[ebp-C3C]	
00021778	57	push eax	eax:"C:\\Users\\IEUser\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Star
00021779	FF 75 FC	push dword ptr ss:[ebp-4]	[/c move]
0002177C	8D 85 04 F7 FF FF	lea eax, dword ptr ss:[ebp-104C]	
00021782	66 0C 20 02 00	push 2220	2220:"%1S\\%1S"
00021787	50	push eax	eax:"C:\\Users\\IEUser\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Star
00021788	FF D6	call esi	esi:wsprintfw

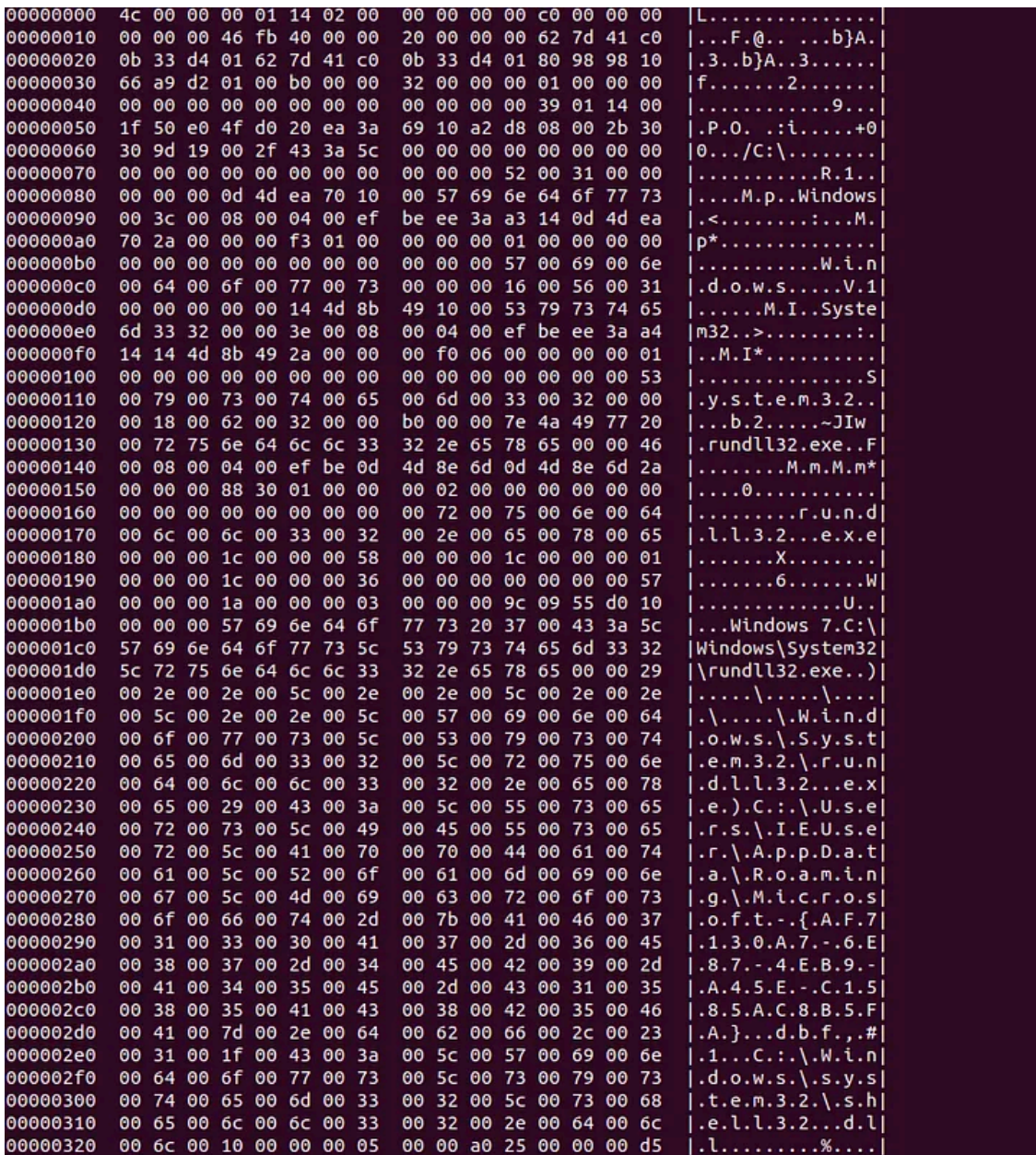
Press enter or click to view image in full size



## 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 persistence 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 !

---

Source: <https://medium.com/@Sebdraaven/when-a-malware-is-more-complex-than-the-paper-5822fc7ff257>