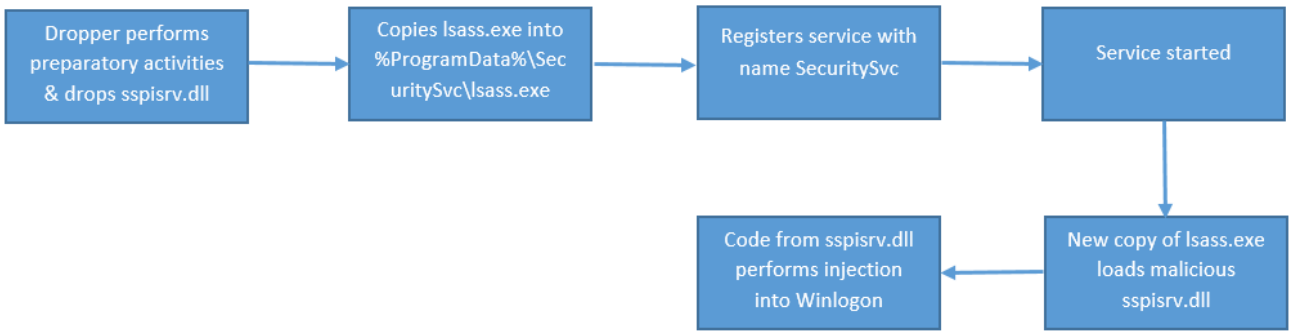
Finfisher rootkit analysis

artemonsecurity.blogspot.de/2017/01/finfisher-rootkit-analysis.html

My previous blog post was dedicated to very interesting malware that is called **Wingbird**. This malware has been used by **NEODYMIUM** cyber espionage group and contains rootkit to execute sensitive and important operations for attackers in a system. The first sample used rootkit for injection malicious code into Winlogon with removing ESET driver hooks in kernel SSDT, while second deploys rootkit for bypassing FS sandbox of several security products. Both droppers analyzed in 32-bit environment, while their behaviour in 64-bit Windows versions are interesting too and different from what we have seen in the 32-bit versions.

In 64-bit system, the dropper doesn't resort to the use of kernel mode rootkit (obviously, due to DSE restrictions) for injection malicious code & data into trusted Winlogon process. Instead this, it uses special trick for masking its malicious activity and for performing injection. The dropper uses copy of trusted LSASS process (executable file) and forces it to load malicious dll with standart name that is imported by LSASS.

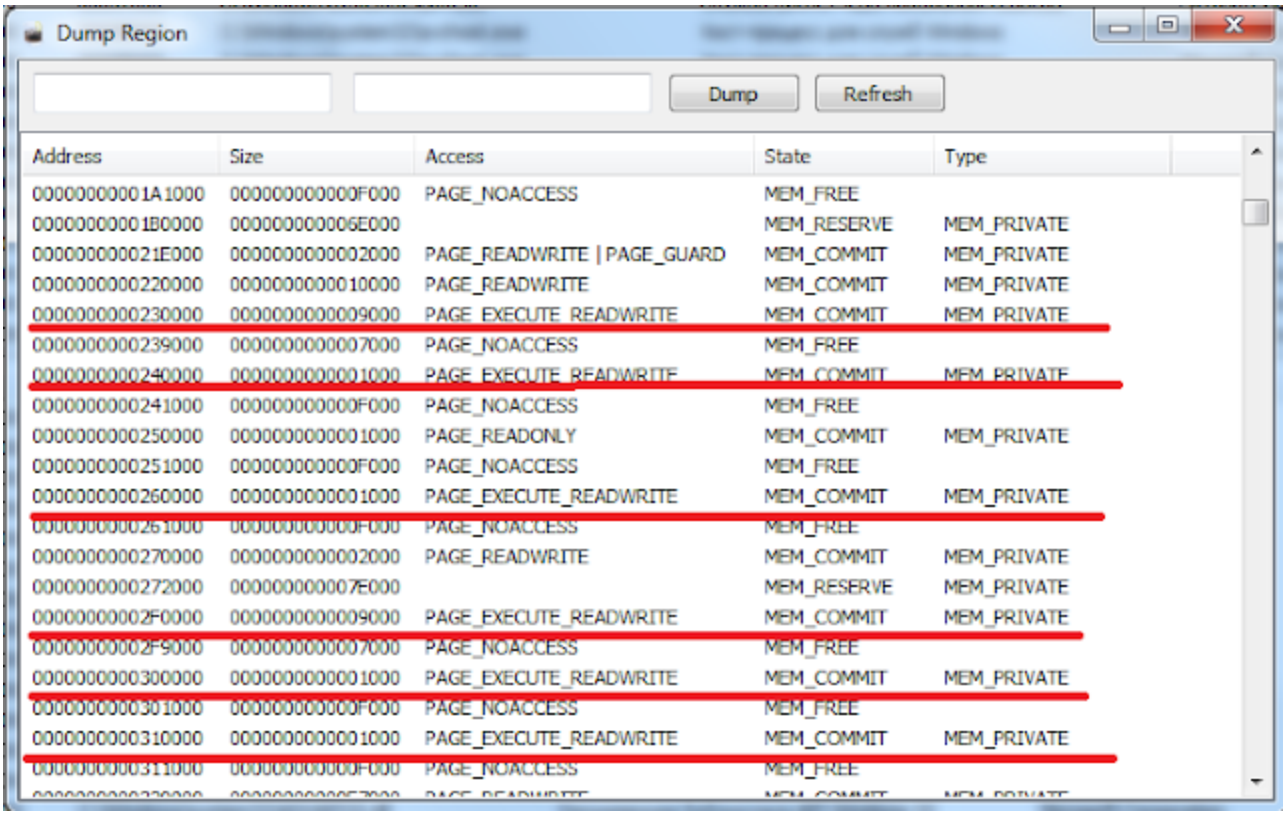


64-bit GMER anti-rootkit tool demonstrates injection anomalies into Winlogon and Svchost, where malicious code is located.

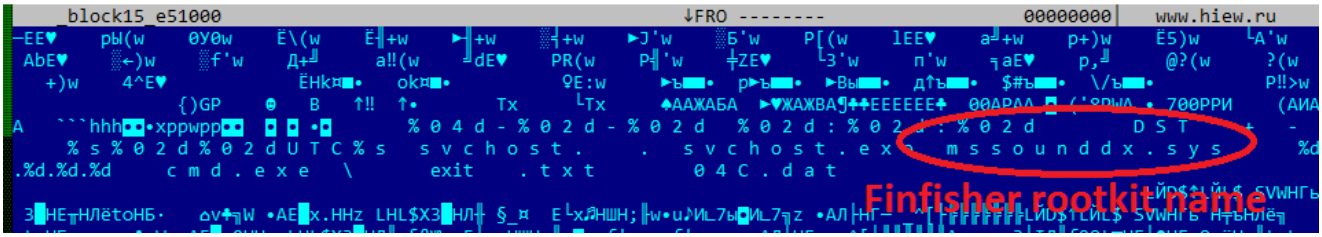
Rootkit/Malware | > > >

Type	Name	Value
Thread	C:\Windows\system32\winlogon.exe [508:884]	0000000000cf0000
Thread	C:\Windows\system32\winlogon.exe [508:1172]	0000000000ca18ec
Thread	C:\Windows\system32\winlogon.exe [508:2132]	0000000002f6ecb0
Thread	C:\Windows\system32\winlogon.exe [508:2060]	0000000002f6ff5c
Thread	C:\Windows\system32\winlogon.exe [508:2472]	0000000000ca2b30
Thread	C:\Windows\system32\winlogon.exe [508:2544]	0000000000ca2b30
Thread	C:\Windows\system32\winlogon.exe [508:2516]	0000000000ca2b30
Thread	C:\Windows\system32\winlogon.exe [508:2588]	0000000000ca2b30
Thread	C:\Windows\system32\winlogon.exe [508:2480]	0000000002f7a588
Thread	C:\Windows\system32\winlogon.exe [508:2856]	0000000002f79fd4
Thread	C:\Windows\system32\winlogon.exe [508:864]	0000000002f7b0ec
Thread	C:\Windows\system32\svchost.exe [628:792]	00000000003a0000

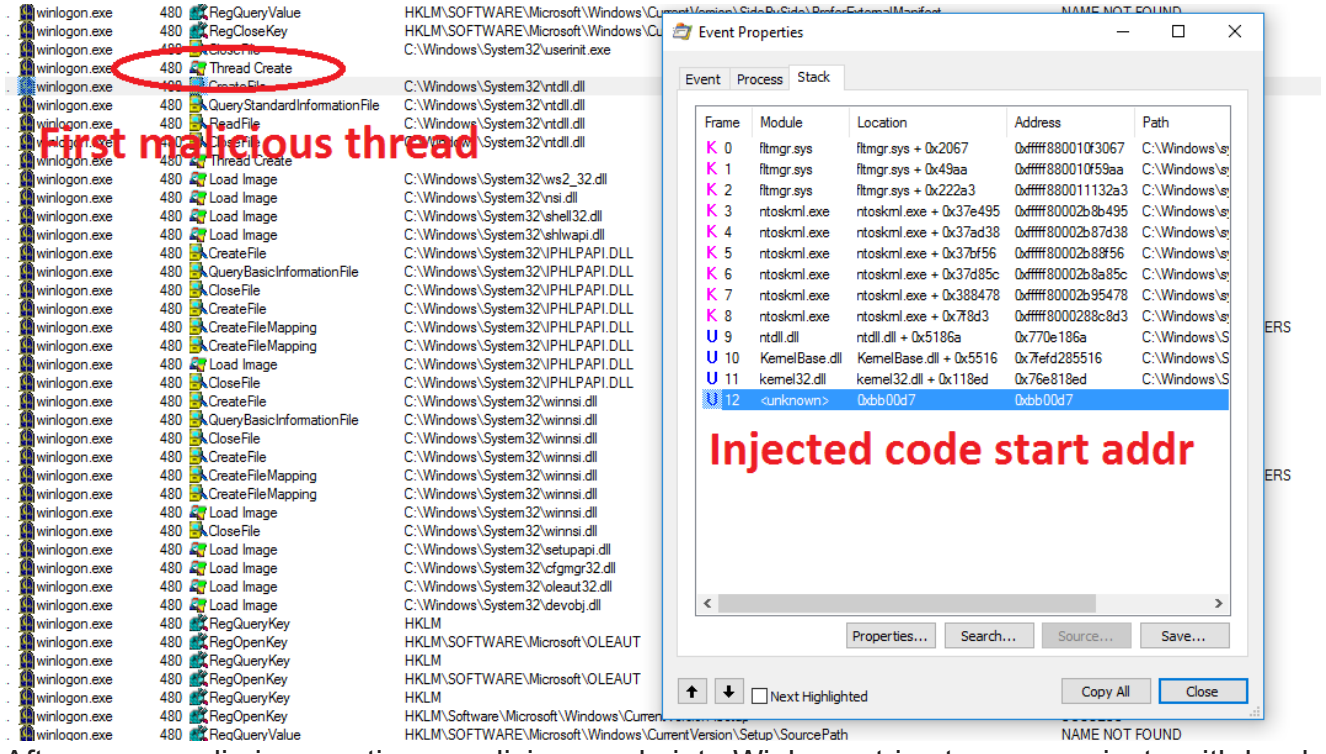
The presence of virtual memory regions into Winlogon with the protection attribute PAGE_EXECUTE_READWRITE is an indicator that the process was compromised.



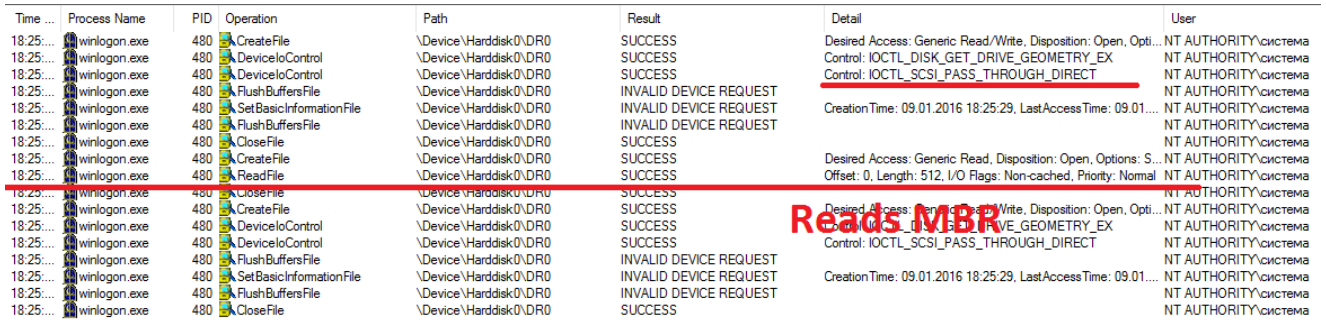
As I already noted in previous blog post, Wingbird malware shares similarities with another malware that is called **Finfisher**. For example, in malicious PE-file that was dumped from Winlogon memory region, we can see reference to name of Finfisher rootkit (**mssounddx.sys**).



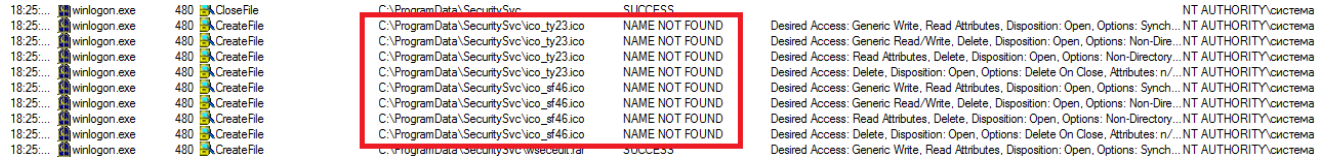
After lsass service started, it injects code into winlogon and with help of ProcMon boot logging we can identify first actions that come from malicious code.



After some preliminary actions, malicious code into Winlogon tries to communicate with hard disk on low level, it requests disk geometry info and sends SCSI control code for reading data. In 32-bit version it uses rootkit to perform this operation.



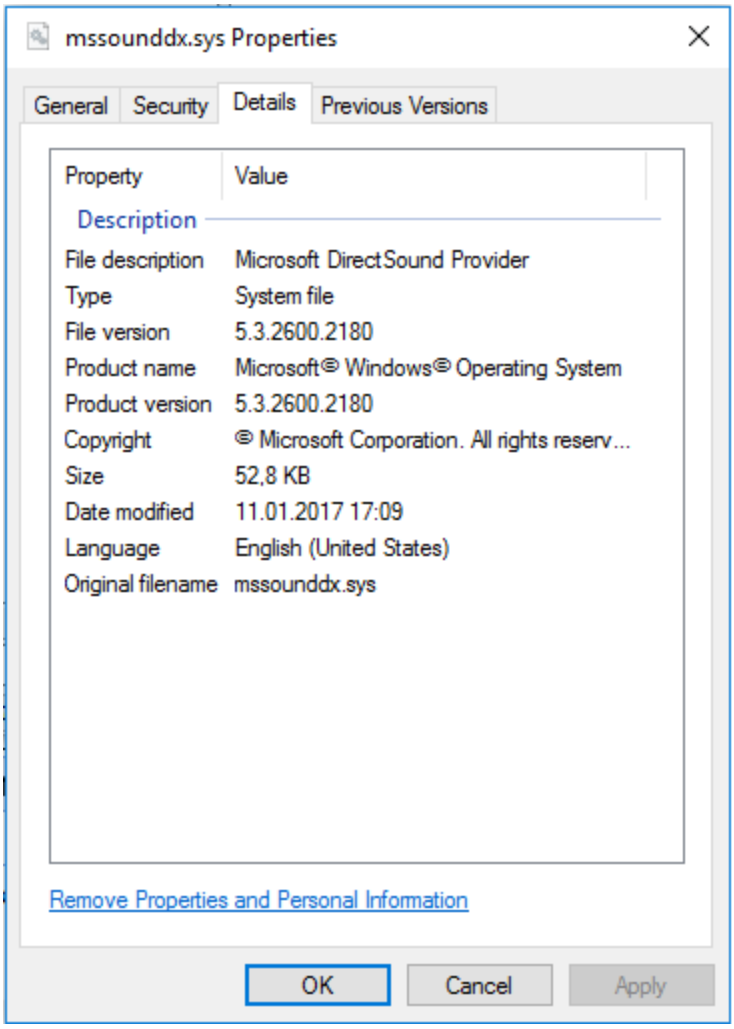
It also checks presence of Finfisher files. See details in Symantec blog post here.



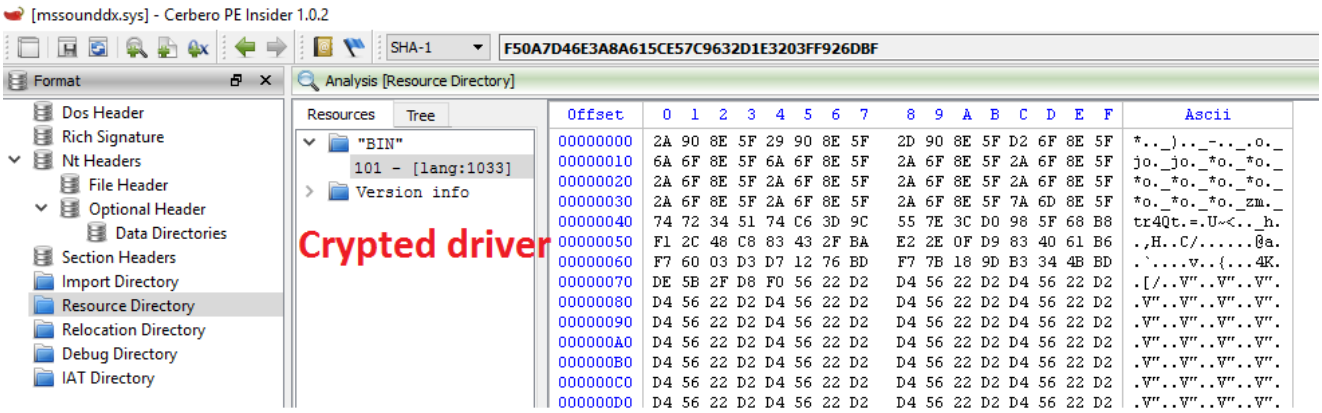
The following indicators show similarities between Wingbird and Finfisher.

Indicator (Wingbird)
Checks presence of ico_ty23.ico
Checks presence of ico_sf46.ico
Contains link to mssounddx.sys
Checks presence of *.dat files

I was able to get 32-bit version of mssounddx.sys rootkit. As you can see on screenshot below, authors masked its file as legitimate Microsoft driver.



Like Wingbird rootkit, Finfisher rootkit is protected from statical analysis. The code from *DriverEntry* and other functions in mssounddx.sys are representing a loader that decrypts content of BIN resource, where 2nd encrypted driver is located.



Rootkit code does following actions in DriverEntry.

1. It is looking for corresponding BIN resource into .rsrc section.
2. It allocates memory block from kernel pool and copies into it content of BIN resource with size 0xc180 (encrypted driver).
3. Decrypts data in allocated pool block.

4. Prepares PE-file of encrypted driver for work: applies fixups, fills some internal variables (ptrs to import functions).
5. Passes control to *DriverEntry* of decrypted driver.

```
.text:F6C71886                 mov     esi, edx
.text:F6C71888                 xor     esi, 5F1ECA67h
.text:F6C7188E                 shr     eax, 2
.text:F6C71891                 push    edi
.text:F6C71892                 mov     [ecx], esi
.text:F6C71894                 lea     edi, [eax-1]
.text:F6C71897                 xor     esi, esi
.text:F6C71899                 test    edi, edi
.text:F6C7189B                 jbe     short loc_F6C718B2
.text:F6C7189D                 push    ebx
.text:F6C7189E
.text:F6C7189E loc_F6C7189E:                           ; CODE XREF: fnDecryptDriver+37↓j
.text:F6C7189E                 mov     eax, [ecx+esi*4+4]
.text:F6C718A2                 mov     ebx, eax
.text:F6C718A4                 xor     ebx, edx
.text:F6C718A6                 mov     [ecx+esi*4+4], ebx
.text:F6C718AA                 inc     esi
.text:F6C718AB                 cmp     esi, edi
.text:F6C718AD                 mov     edx, eax
.text:F6C718AF                 jb      short loc_F6C7189E
.text:F6C718B1                 pop     ebx
.text:F6C718B2
.text:F6C718B2 loc_F6C718B2:                           ; CODE XREF: fnDecryptDriver+23↑j
.text:F6C718B2                 pop     edi
.text:F6C718B3                 pop     esi
.text:F6C718B4                 pop     ebp
.text:F6C718B5                 retn    8
.text:F6C718B5 fnDecryptDriver endp
```

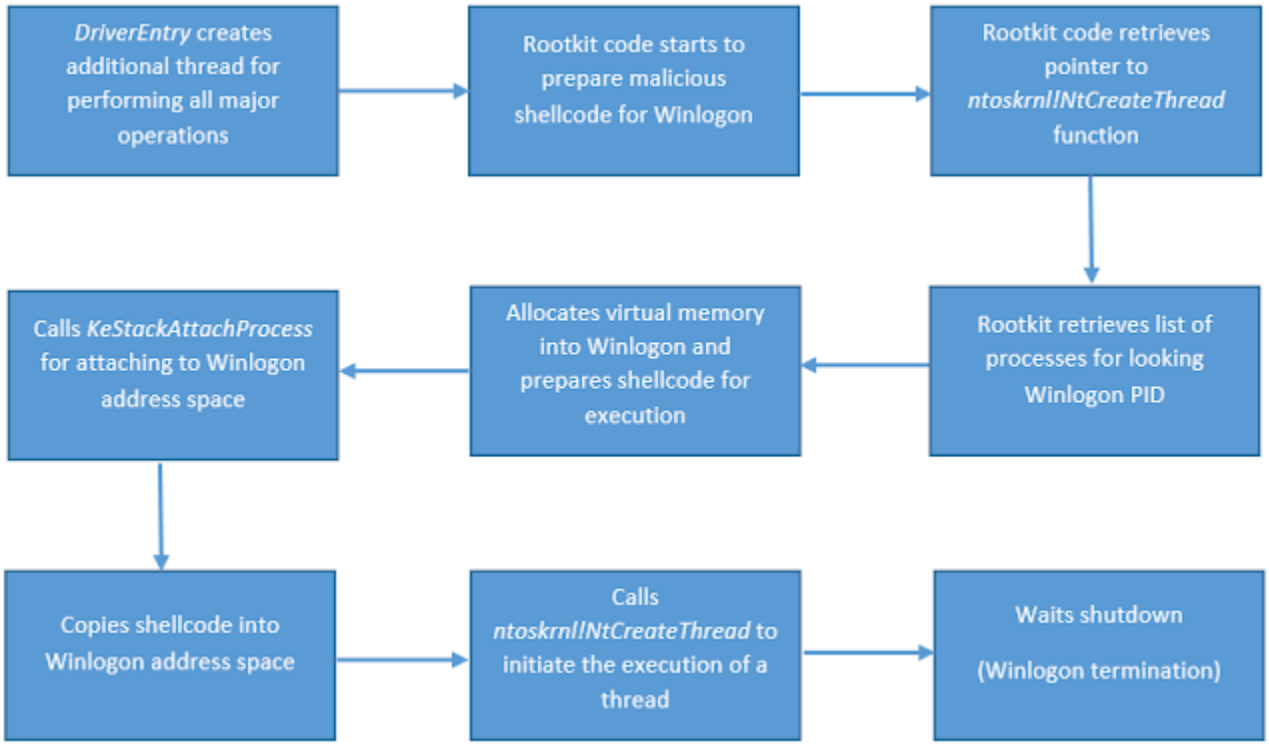
Second driver uses following kernel functions.

```
 1  ExFreePoolWithTag
 2  _stricmp
 3  ZwQuerySystemInformation
 4  ExAllocatePoolWithTag
 5  wcsicmp
 6  ObfDereferenceObject
 7  KeWaitForSingleObject
 8  ZwClose
 9  ZwFreeVirtualMemory
10  KeUnstackDetachProcess
11  PsLookupProcessByProcessId
12  ZwAllocateVirtualMemory
13  ZwOpenProcess
14  KeDelayExecutionThread
15  PsGetVersion
16  MmGetSystemRoutineAddress
17  RtlInitUnicodeString
18  memset
19  memcpy
20  KeServiceDescriptorTable
21  PsCreateSystemThread
22  wcsncpy
23  ZwQueryValueKey
24  ZwOpenKey
25  KeTickCount
```

Code injection.

```
.text:00010902                  push    eax
.text:00010903                  push    [ebp+var_2C]
.text:00010906                  call    ds:PsLookupProcessByProcessId
.text:0001090C                  lea     eax, [ebp+var_84]
.text:00010912                  push    eax
.text:00010913                  push    [ebp+var_20]
.text:00010916                  call    [ebp+pKeStackAttachProcess]
.text:00010919                  push    [ebp+var_34]
.text:0001091C                  push    [ebp+var_3C]
.text:0001091F                  push    [ebp+var_30]
.text:00010922                  call    memcpy
.text:00010927                  push    edi
.text:00010928                  push    [ebp+var_40]
.text:0001092B                  push    esi
.text:0001092C                  call    memcpy
.text:00010931                  add     esp, 18h
.text:00010934                  lea     eax, [ebp+var_84]
.text:0001093A                  push    eax
.text:0001093B                  call    ds:KeUnstackDetachProcess
.text:00010941                  push    2CCh
```

Next picture demonstrates logic of 2nd driver execution.



Start of shellcode looks like.

```
.data:00011080
.data:00011080                  loc_11080:                              ; DATA XREF: sub_10490+C↑o
.data:00011080 E8 E2 0A 00 00                call    sub_11B67
.data:00011085 E8 00 00 00 00                call    $+5
.data:0001108A
.data:0001108A                  loc_1108A:                              ; DATA XREF: .data:0001108B↓o
.data:0001108A 5D                            pop     ebp
.data:0001108B 81 ED 8A 10 01+               sub     ebp, offset loc_1108A
.data:00011091 89 85 65 16 01+               mov     ss:dword_11665[ebp], eax
.data:00011097 89 9D 55 16 01+               mov     ss:dword_11655[ebp], ebx
.data:0001109D 89 B5 5D 16 01+               mov     ss:dword_1165D[ebp], esi
.data:000110A3 8D B5 B5 16 01+               lea     esi, dword_116B5[ebp]
.data:000110A9 8D BD 71 16 01+               lea     edi, dword_11671[ebp]
.data:000110AF 6A 00                         push    0
.data:000110B1 68 FF 1F 7C C9                push    0C97C1FFFh
.data:000110B6 FF B5 65 16 01+               push    ss:dword_11665[ebp]
.data:000110BC E8 3C 0A 00 00                call    sub_11AFD
```

Conclusion

As you can see from the analysis, we haven't seen something new in Finfisher rootkit. Like other drivers that are used by attackers, it is intended only for one purpose - for injection malicious code into Winlogon process. Nevertheless, authors use some anti-analysis tricks, including, driver encryption and obfuscation some data that driver keeps in kernel memory.