A Place for Malware Geeks

# Research, Low-Level, Vulnerabilities, Exploitation

Sponsored by enSilo

# Moker, Part 1: dissecting a new APT under the microscope

Recently, we came across Moker, an advanced malware residing in a sensitive network of a customer. Since the malware did not try to access an external server, but rather tamper with the system inner workings, we decided to give this malware a second look.

Indeed, we found that there is more to this malware than meets the eye.

First, it was obvious that the malware authors placed many anti-research measures, beyond those of the so-called usual anti-debugging techniques.

Second, once we started analyzing the malware itself, we were also able to analyze its capabilities and the advanced techniques it uses to remain stealthy, hook itself into the operating system, and its sophisticated inter-process communication.

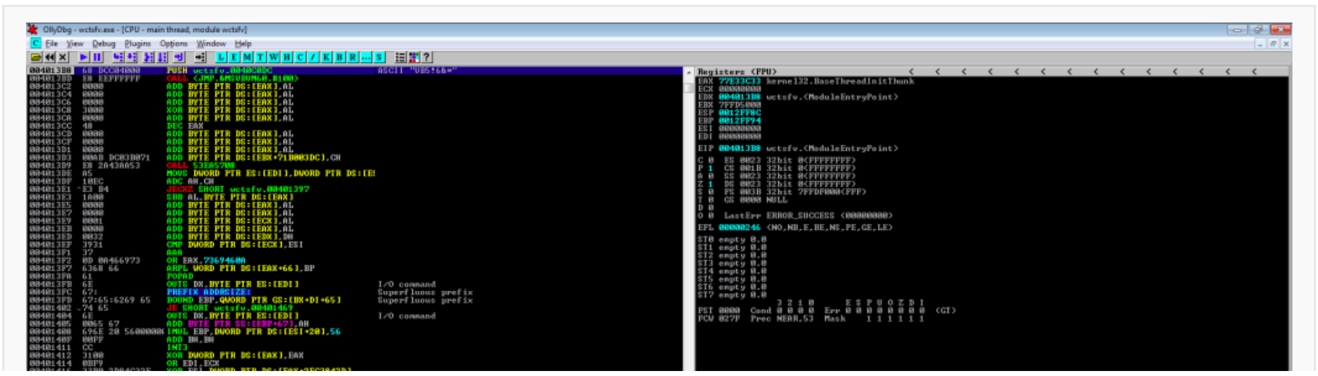
We decided to call this malware Moker as it's the file description that the malware author gave to the malware's executable file.

In this first of a 2-series blog we'll present the challenges that Moker placed in front of researchers to avoid detection and anti-dissection; and show how we overcame these measures one-by-one until we arrived at the stripped-down malware sample.

In an upcoming second blog entry, we'll present the comprehensive list and different advanced techniques that Moker utilized. Since we're still analyzing the malware, we'll present the technical details within the next few days.

## Challenge #1: initial anti-debugging and unpacking

We run the malware inside our virtual environment to first understand what type of executable we are looking at. In fact, we are able to recognize that the malware (or at least its packer), is a Visual 5 or 6 binary:


OllyDbg - wctsfv.exe - [CPU - main thread, module wctsfv]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S

```
004013B8   68 DCC04000      PUSH wctsfv.0040C0DC                    ASCII "VB5!6&*"
004013BD   E8 EEFFFFFF      CALL <JMP.&MSVBVM60.#100>
004013C2   0000             ADD BYTE PTR DS:[EAX],AL
004013C4   0000             ADD BYTE PTR DS:[EAX],AL
004013C6   0000             ADD BYTE PTR DS:[EAX],AL
004013C8   3000             XOR BYTE PTR DS:[EAX],AL
004013CA   0000             ADD BYTE PTR DS:[EAX],AL
004013CC   48               DEC EAX
004013CD   0000             ADD BYTE PTR DS:[EAX],AL
004013CF   0000             ADD BYTE PTR DS:[EAX],AL
004013D1   0000             ADD BYTE PTR DS:[EAX],AL
004013D3   00AB DC03B071    ADD BYTE PTR DS:[EBX+71B003DC],CH
004013D9   E8 2A43AA53      CALL 53EA5708
004013DE   A5               MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[E
004013DF   10EC             ADC AH,CH
004013E1  ^E3 B4            JECXZ SHORT wctsfv.00401397
004013E3   1A00             SBB AL,BYTE PTR DS:[EAX]
004013E5   0000             ADD BYTE PTR DS:[EAX],AL
004013E7   0000             ADD BYTE PTR DS:[EAX],AL
004013E9   0001             ADD BYTE PTR DS:[ECX],AL
004013EB   0000             ADD BYTE PTR DS:[EAX],AL
004013ED   0032             ADD BYTE PTR DS:[EDX],DH
004013EF   3931             CMP DWORD PTR DS:[ECX],ESI
004013F1   37               AAA
004013F2   0D 0A466973      OR EAX,7369460A
004013F7   6368 66          ARPL WORD PTR DS:[EAX+66],BP
004013FA   61               POPAD
004013FB   6E               OUTS DX,BYTE PTR ES:[EDI]               I/O command
004013FC   67:              PREFIX ADDRSIZE:                        Superfluous prefix
004013FD   67:65:6269 65    BOUND EBP,QWORD PTR GS:[BX+DI+65]       Superfluous prefix
00401402  ,74 65            JE SHORT wctsfv.00401469
00401404   6E               OUTS DX,BYTE PTR ES:[EDI]               I/O command
00401405   0065 67          ADD BYTE PTR SS:[EBP+67],AH
00401408   696E 20 5600000  IMUL EBP,DWORD PTR DS:[ESI+20],56
0040140F   00FF             ADD BH,BH
00401411   CC               INT3
00401412   3100             XOR DWORD PTR DS:[EAX],EAX
00401414   0BF9             OR EDI,ECX
```

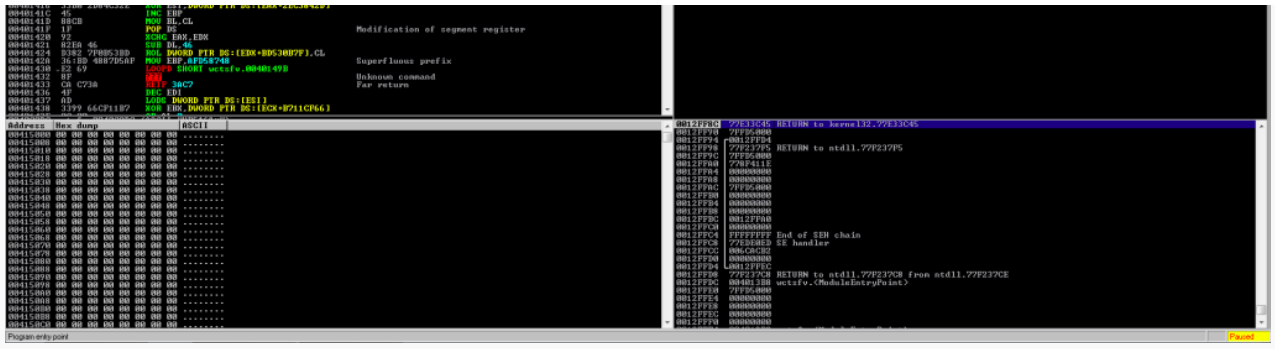
```
Registers (FPU)
EAX 77E33C33 kernel32.BaseThreadInitThunk
ECX 00000000
EDX 004013B8 wctsfv.<ModuleEntryPoint>
EBX 7FFD5000
ESP 0012FF8C
EBP 0012FF94
ESI 00000000
EDI 00000000

EIP 004013B8 wctsfv.<ModuleEntryPoint>

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```



```
0040141C  45               INC EBP
0040141D  8BCB             MOV BL,CL
0040141F  1F               POP DS                                  Modification of segment register
00401420  92               XCHG EAX,EDX
00401421  82EA 46          SUB DL,46
00401424  D382 7F0B53BD    ROL DWORD PTR DS:[EDX+BD530B7F],CL
0040142A  36:BD 4887D5AF   MOV EBP,AFD58748                        Superfluous prefix
00401430 .E2 69            LOOPD SHORT wctsfv.0040149B
00401432  8F               ???                                     Unknown command
00401433  CA C73A          RETF 3AC7                               Far return
00401436  4F               DEC EDI
00401437  AD               LODS DWORD PTR DS:[ESI]
00401438  3399 66CF11B7    XOR EBX,DWORD PTR DS:[ECX+B711CF66]
```

```
Address   Hex dump                 ASCII
00415000  00 00 00 00 00 00 00 00  ........
00415008  00 00 00 00 00 00 00 00  ........
00415010  00 00 00 00 00 00 00 00  ........
00415018  00 00 00 00 00 00 00 00  ........
00415020  00 00 00 00 00 00 00 00  ........
00415028  00 00 00 00 00 00 00 00  ........
00415030  00 00 00 00 00 00 00 00  ........
00415038  00 00 00 00 00 00 00 00  ........
00415040  00 00 00 00 00 00 00 00  ........
00415048  00 00 00 00 00 00 00 00  ........
00415050  00 00 00 00 00 00 00 00  ........
00415058  00 00 00 00 00 00 00 00  ........
```

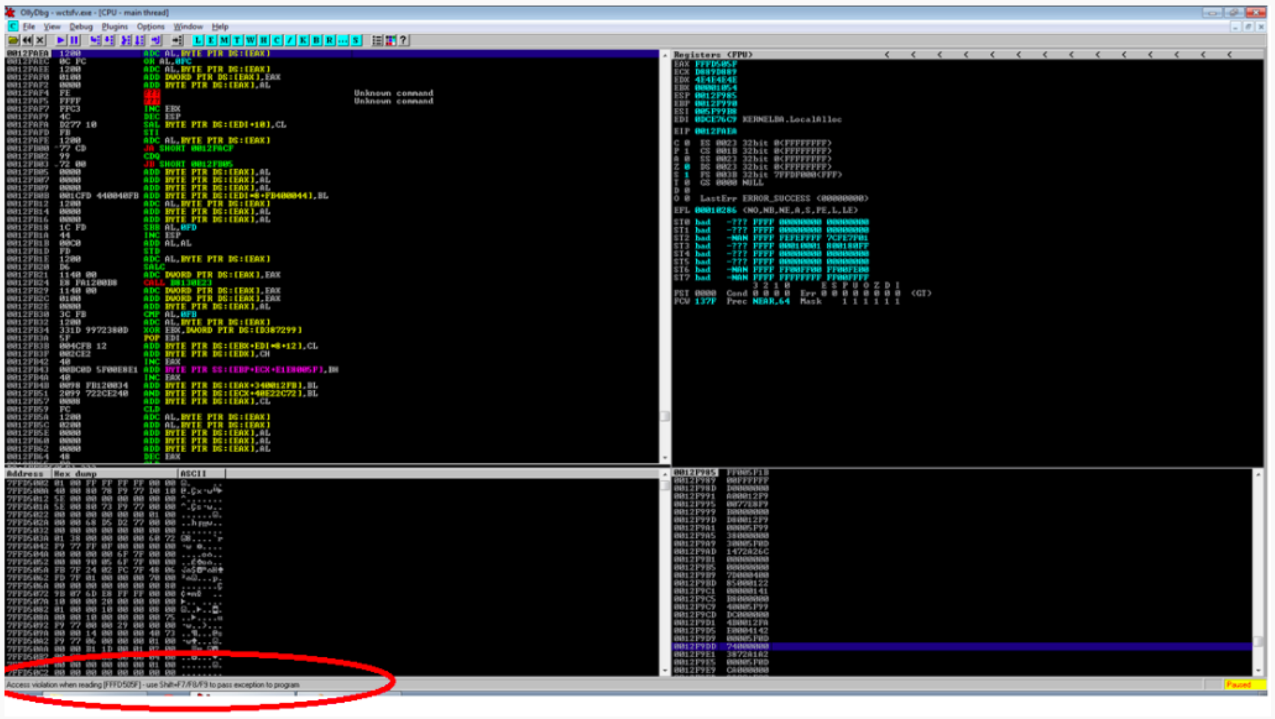
```
0012FF8C   77E33C45  RETURN to kernel32.77E33C45
0012FF90   7FFD5000
0012FF94  ┌0012FFD4
0012FF98  │77F237F5  RETURN to ntdll.77F237F5
0012FF9C  │7FFD5000
0012FFA0  │778F411E
0012FFA4  │00000000
0012FFA8  │00000000
0012FFAC  │7FFD5000
0012FFB0  │00000000
0012FFB4  │00000000
0012FFB8  │00000000
0012FFBC  │0012FFA0
0012FFC0  │00000000
0012FFC4  │FFFFFFFF  End of SEH chain
0012FFC8  │77EDE0ED  SE handler
0012FFCC  │006CACB2
0012FFD0  │00000000
0012FFD4  └0012FFEC
0012FFD8   77F237C8  RETURN to ntdll.77F237C8 from ntdll.77F237CE
0012FFDC   004013B8  wctsfv.<ModuleEntryPoint>
0012FFE0   7FFD5000
0012FFE4   00000000
0012FFE8   00000000
0012FFEC   00000000
0012FFF0   00000000
```

Program entry point

Paused

Figure 1: Recognizing that it's a Visual 5 or 6 Binary (first line, highlighted in blue)

Understanding the type of binary, we run the sample inside the appropriate debugger but very quickly the program crashes on Access Violation.



OllyDbg - wctsfv.exe - [CPU - main thread]

```
0012FAEA  1200             ADC AL,BYTE PTR DS:[EAX]
0012FAEC  0C FC            OR AL,0FC
0012FAEE  1200             ADC AL,BYTE PTR DS:[EAX]
0012FAF0  0100             ADD DWORD PTR DS:[EAX],EAX
0012FAF2  0000             ADD BYTE PTR DS:[EAX],AL
0012FAF4  FE               ???                                     Unknown command
0012FAF5  FFFF             ???                                     Unknown command
0012FAF7  FFC3             INC EBX
0012FAF9  4C               DEC ESP
0012FAFA  D277 10          SAL BYTE PTR DS:[EDI+10],CL
0012FAFD  FB               STI
0012FAFE  1200             ADC AL,BYTE PTR DS:[EAX]
0012FB00 ^77 CD            JA SHORT 0012FACF
0012FB02  99               CDQ
0012FB03 .72 00            JB SHORT 0012FB05
0012FB05  0000             ADD BYTE PTR DS:[EAX],AL
0012FB07  0000             ADD BYTE PTR DS:[EAX],AL
0012FB09  0000             ADD BYTE PTR DS:[EAX],AL
0012FB0B  001CFD 440040FB  ADD BYTE PTR DS:[EDI*8+FB400044],BL
0012FB12  1200             ADC AL,BYTE PTR DS:[EAX]
0012FB14  0000             ADD BYTE PTR DS:[EAX],AL
0012FB16  0000             ADD BYTE PTR DS:[EAX],AL
0012FB18  1C FD            SBB AL,0FD
0012FB1A  44               INC ESP
0012FB1B  00C0             ADD AL,AL
0012FB1D  FD               STD
0012FB1E  1200             ADC AL,BYTE PTR DS:[EAX]
0012FB20  D6               SALC
0012FB21  1140 00          ADC DWORD PTR DS:[EAX],EAX
0012FB24  E8 FA12000B      CALL 0B130E23
0012FB29  1140 00          ADC DWORD PTR DS:[EAX],EAX
0012FB2C  0100             ADD DWORD PTR DS:[EAX],EAX
0012FB2E  0000             ADD BYTE PTR DS:[EAX],AL
0012FB30  3C FB            CMP AL,0FB
0012FB32  1200             ADC AL,BYTE PTR DS:[EAX]
0012FB34  331D 9972380D    XOR EBX,DWORD PTR DS:[D3387299]
0012FB3A  5F               POP EDI
0012FB3B  004CFB 12        ADD BYTE PTR DS:[EBX+EDI*8+12],CL
0012FB3F  002CE2           ADD BYTE PTR DS:[EDX],CH
0012FB42  40               INC EAX
```

```
Registers (FPU)
EAX FFFD505F
ECX D0B9D0B9
EDX 4E4E4E4E
EBX 00001054
ESP 0012F985
EBP 0012F990
ESI 005F99B8
EDI 0DCE76C9 KERNELBA.LocalAlloc

EIP 0012FAEA

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 1  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010286 (NO,NB,NE,A,S,PE,L,LE)

ST0 bad    -??? FFFF 00000000 00000000
ST1 bad    -??? FFFF 00000000 00000000
ST2 bad    -NAN FFFF FEFEFFFF 7CFE7F01
ST3 bad    -??? FFFF 00010001 800180FF
ST4 bad    -??? FFFF 00000000 00000000
ST5 bad    -??? FFFF 00000000 00000000
ST6 bad    -NAN FFFF FF00FF00 FF00FE00
ST7 bad    -NAN FFFF FFFFFFFF FF00FFFF
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 137F  Prec NEAR,64  Mask    1 1 1 1 1 1
```

```
7FFD503A  01 38 00 00 00 00 60 72
7FFD5042  F9 77 FF 0F 00 00 00 00
7FFD504A  00 00 00 00 6F 7F 00 00
7FFD5052  00 00 90 05 6F 7F 00 00
7FFD505A  FB 7F 24 02 FC 7F 48 06
7FFD5062  FD 7F 01 00 00 00 70 00
7FFD506A  00 00 00 00 00 00 00 80
7FFD5072  9B 07 6D E8 FF FF 00 00
7FFD507A  10 00 00 20 00 00 00 00
7FFD5082  01 00 00 10 00 00 08 00
7FFD508A  00 00 10 00 00 00 00 75
7FFD5092  F9 77 00 00 29 00 00 00
7FFD509A  00 00 14 00 00 00 40 73
7FFD50A2  F9 77 06 00 00 00 01 00
```

Access violation when reading [FFFD505F] - use Shift+F7/F8/F9 to pass exception to program

Paused

Figure 2: Access Violation in the stack memory space

The malware somehow knows that it is running inside the context of a debugger and jumps to some junk code.

The next step is to look for the appropriate debugger state flag – the PEB!BeingDebugged flag.

The corresponding byte is in the Process Environment Block (PEB). The byte is read by the API IsDebuggerPresent, and can be read manually by accessing:

***FS:[0x30]+2***

Since this byte has no other meaning apart from the debug status, we are able to safely reset it to zero.

Run again. Crash again. On a different address this time.

Our packer has another anti-debugging trick up its sleeve. This time it turns out to be a slightly less common method (though still in use): the use of PEB!NtGlobalFlags. Its corresponding DWORD can be read using an API, or by directly accessing this address:

***FS[0x30]+0x68***

Reset the DWORD to zero. Run. No crash this time. However, the process terminated.

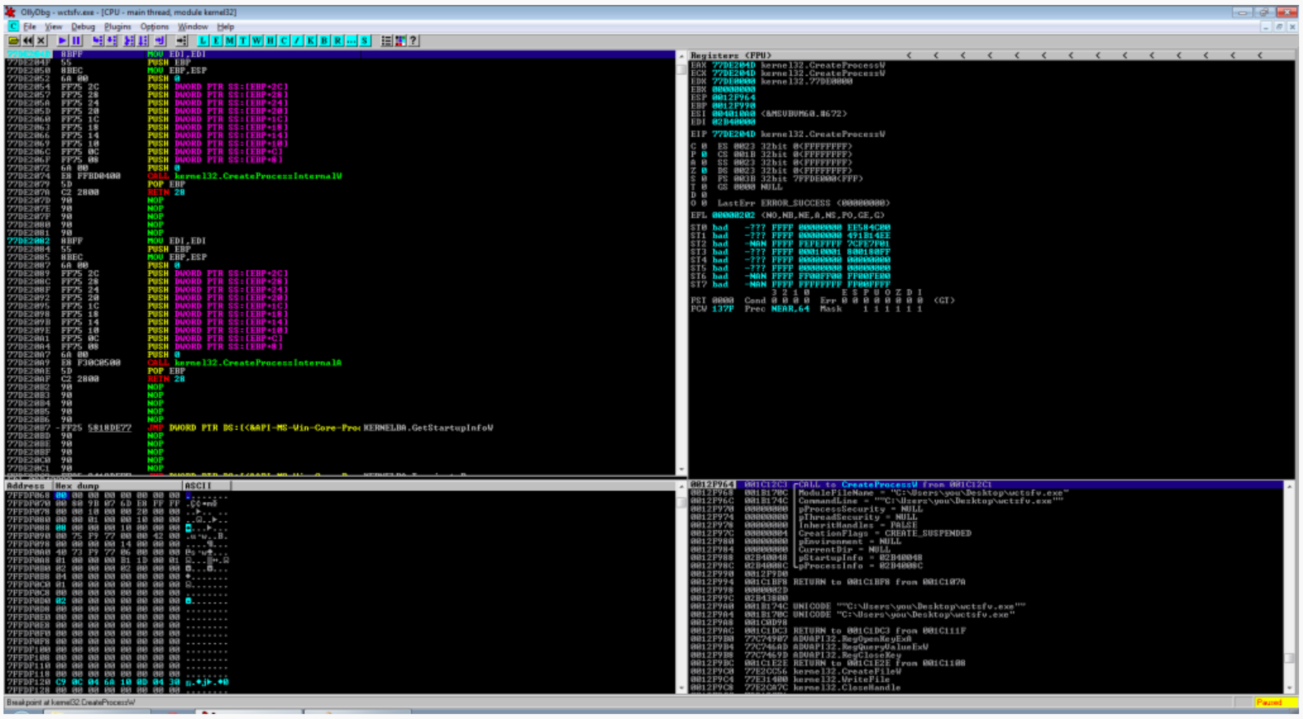
This fact alone means little. Anti-debugging can still be present and the process could have called ExitProcess or TerminateProcess before it does what it is designed to do.

In this specific case, however, anti-debugging is defeated.

## Challenge #2: Process Hollowing

Beginning to understand what the process is doing, let's set breakpoints at some interesting APIs, one of these being process creation.

Sure enough, a breakpoint at CreateProcessW is hit.


OllyDbg - wctsfv.exe - [CPU - main thread, module kernel32]
File View Debug Plugins Options Window Help
Breakpoint at kernel32.CreateProcessW
Paused

```
77DE204D   8BFF             MOV EDI,EDI
77DE204F   55               PUSH EBP
77DE2050   8BEC             MOV EBP,ESP
77DE2052   6A 00            PUSH 0
77DE2054   FF75 2C          PUSH DWORD PTR SS:[EBP+2C]
77DE2057   FF75 28          PUSH DWORD PTR SS:[EBP+28]
77DE205A   FF75 24          PUSH DWORD PTR SS:[EBP+24]
77DE205D   FF75 20          PUSH DWORD PTR SS:[EBP+20]
77DE2060   FF75 1C          PUSH DWORD PTR SS:[EBP+1C]
77DE2063   FF75 18          PUSH DWORD PTR SS:[EBP+18]
77DE2066   FF75 14          PUSH DWORD PTR SS:[EBP+14]
77DE2069   FF75 10          PUSH DWORD PTR SS:[EBP+10]
77DE206C   FF75 0C          PUSH DWORD PTR SS:[EBP+C]
77DE206F   FF75 08          PUSH DWORD PTR SS:[EBP+8]
77DE2072   6A 00            PUSH 0
77DE2074   E8 FFBD0400      CALL kernel32.CreateProcessInternalW
77DE2079   5D               POP EBP
77DE207A   C2 2800          RETN 28
77DE207D   90               NOP
77DE207E   90               NOP
77DE207F   90               NOP
77DE2080   90               NOP
77DE2081   90               NOP
77DE2082   8BFF             MOV EDI,EDI
77DE2084   55               PUSH EBP
77DE2085   8BEC             MOV EBP,ESP
77DE2087   6A 00            PUSH 0
77DE2089   FF75 2C          PUSH DWORD PTR SS:[EBP+2C]
77DE208C   FF75 28          PUSH DWORD PTR SS:[EBP+28]
77DE208F   FF75 24          PUSH DWORD PTR SS:[EBP+24]
77DE2092   FF75 20          PUSH DWORD PTR SS:[EBP+20]
77DE2095   FF75 1C          PUSH DWORD PTR SS:[EBP+1C]
77DE2098   FF75 18          PUSH DWORD PTR SS:[EBP+18]
77DE209B   FF75 14          PUSH DWORD PTR SS:[EBP+14]
77DE209E   FF75 10          PUSH DWORD PTR SS:[EBP+10]
77DE20A1   FF75 0C          PUSH DWORD PTR SS:[EBP+C]
77DE20A4   FF75 08          PUSH DWORD PTR SS:[EBP+8]
77DE20A7   6A 00            PUSH 0
77DE20A9   E8 F30C0500      CALL kernel32.CreateProcessInternalA
77DE20AE   5D               POP EBP
77DE20AF   C2 2800          RETN 28
77DE20B2   90               NOP
77DE20B3   90               NOP
77DE20B4   90               NOP
77DE20B5   90               NOP
77DE20B6   90               NOP
77DE20B7 - FF25 5818DE77    JMP DWORD PTR DS:[<&API-MS-Win-Core-Pro: KERNELBA.GetStartupInfoW
77DE20BD   90               NOP
77DE20BE   90               NOP
77DE20BF   90               NOP
77DE20C0   90               NOP
77DE20C1   90               NOP
```

```
Registers (FPU)
EAX 77DE204D kernel32.CreateProcessW
ECX 77DE204D kernel32.CreateProcessW
EDX 77DE0000 kernel32.77DE0000
EBX 00000000
ESP 0012F964
EBP 0012F990
ESI 004010A0 <&MSVBVM60.#672>
EDI 02D40000

EIP 77DE204D kernel32.CreateProcessW

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000202 (NO,NB,NE,A,NS,PO,GE,G)

ST0 bad    -??? FFFF 00000000 EE584C00
ST1 bad    -??? FFFF 00000000 491B14EE
ST2 bad    -NAN FFFF FEFEFFFF 7CFE7F01
ST3 bad    -??? FFFF 00010001 800180FF
ST4 bad    -??? FFFF 00000000 00000000
ST5 bad    -??? FFFF 00000000 00000000
ST6 bad    -NAN FFFF FF00FF00 FF00FE00
ST7 bad    -NAN FFFF FFFFFFFF FF00FFFF
                   3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 137F  Prec NEAR,64  Mask    1 1 1 1 1 1
```

```
Address   Hex dump                 ASCII
7FFDF068  00 00 00 00 00 00 00 00  ........
7FFDF070  00 80 9B 07 6D E8 FF FF  ........
7FFDF078  00 00 10 00 00 20 00 00  ........
7FFDF080  00 00 01 00 00 10 00 00  ........
7FFDF088  00 00 00 00 10 00 00 00  ........
7FFDF090  00 75 F9 77 00 00 42 00  .u.w..B.
7FFDF098  00 00 00 00 14 00 00 00  ........
7FFDF0A0  40 73 F9 77 06 00 00 00  @s.w....
7FFDF0A8  01 00 00 00 B1 1D 00 01  ........
7FFDF0B0  02 00 00 00 02 00 00 00  ........
7FFDF0B8  04 00 00 00 00 00 00 00  ........
7FFDF0C0  01 00 00 00 00 00 00 00  ........
7FFDF0C8  00 00 00 00 00 00 00 00  ........
7FFDF0D0  02 00 00 00 00 00 00 00  ........
7FFDF0D8  00 00 00 00 00 00 00 00  ........
7FFDF0E0  00 00 00 00 00 00 00 00  ........
7FFDF0E8  00 00 00 00 00 00 00 00  ........
7FFDF0F0  00 00 00 00 00 00 00 00  ........
7FFDF0F8  00 00 00 00 00 00 00 00  ........
7FFDF100  00 00 00 00 00 00 00 00  ........
7FFDF108  00 00 00 00 00 00 00 00  ........
7FFDF110  00 00 00 00 00 00 00 00  ........
7FFDF118  00 00 00 00 00 00 00 00  ........
7FFDF120  C9 0C 04 6A 10 0D 04 30  ...j...0
7FFDF128  00 00 00 00 00 00 00 00  ........
```

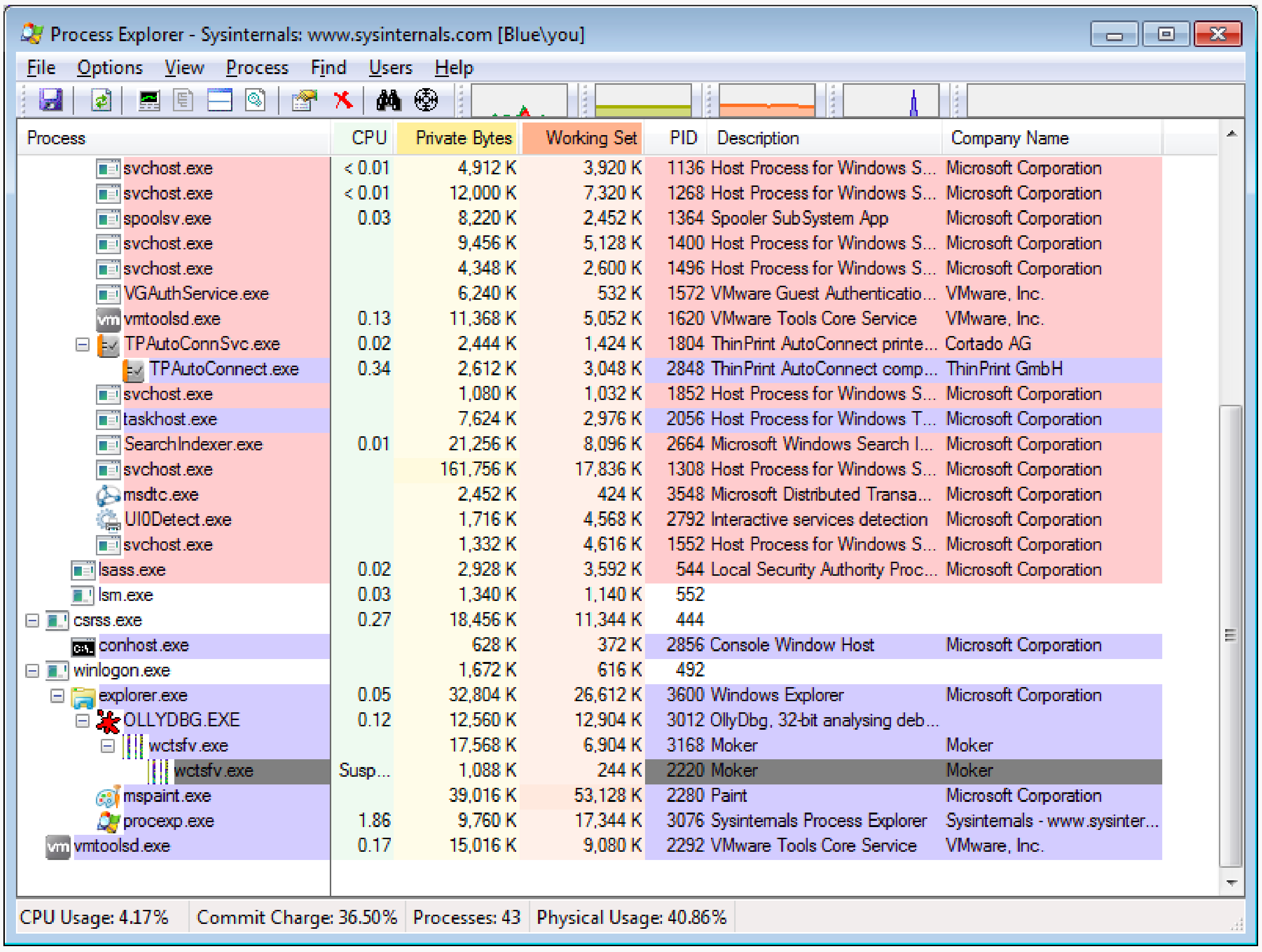
```
0012F964   001C12C3  ┌CALL to CreateProcessW from 001C12C1
0012F968   001B170C  │ModuleFileName = "C:\Users\you\Desktop\wctsfv.exe"
0012F96C   001B174C  │CommandLine = ""C:\Users\you\Desktop\wctsfv.exe""
0012F970   00000000  │pProcessSecurity = NULL
0012F974   00000000  │pThreadSecurity = NULL
0012F978   00000000  │InheritHandles = FALSE
0012F97C   00000004  │CreationFlags = CREATE_SUSPENDED
0012F980   00000000  │pEnvironment = NULL
0012F984   00000000  │CurrentDir = NULL
0012F988   02B40048  │pStartupInfo = 02B40048
0012F98C   02B4000C  └pProcessInfo = 02B4000C
0012F990   0012F9D0
0012F994   001C1BF8  RETURN to 001C1BF8 from 001C107A
0012F998   0000002D
0012F99C   02B43800
0012F9A0   001B174C  UNICODE ""C:\Users\you\Desktop\wctsfv.exe""
0012F9A4   001B170C  UNICODE "C:\Users\you\Desktop\wctsfv.exe"
0012F9A8   001C0D98
0012F9AC   001C1DC3  RETURN to 001C1DC3 from 001C111F
0012F9B0   77C74907  ADVAPI32.RegOpenKeyExA
0012F9B4   77C746AD  ADVAPI32.RegQueryValueExW
0012F9B8   77C7469D  ADVAPI32.RegCloseKey
0012F9BC   001C1E2E  RETURN to 001C1E2E from 001C1100
0012F9C0   77E2CC56  kernel32.CreateFileW
0012F9C4   77E31400  kernel32.WriteFile
0012F9C8   77E2CA7C  kernel32.CloseHandle
```

Figure 3: CreateProcessW is creating with the CREATE_SUSPEND flag

As indicated by the code above, the Moker sample tries to run its own executable without arguments, and with the CREATE_SUSPENDED flag. This means that when the call returns, a new process will be created in a suspended state. The process will remain suspended until killed or until one of its process' threads is resumed.

Continuing to run Moker, we see that on the created process at its base address (0x400000) the sample uses UnmapViewOfSection and MapViewOfSection. It then calls NtResumeThread, but let's wait with that one for now.

As we can see, at this stage we have a process running, but its code is not the original code in the file. In fact, the sample completely overwrites the new process' PE header, giving it a new entry point and a whole section of new code.


Process Explorer - Sysinternals: www.sysinternals.com [Blue\you]
File Options View Process Find Users Help

| Process | CPU | Private Bytes | Working Set | PID | Description | Company Name |
|---|---|---|---|---|---|---|
| svchost.exe | < 0.01 | 4,912 K | 3,920 K | 1136 | Host Process for Windows S... | Microsoft Corporation |
| svchost.exe | < 0.01 | 12,000 K | 7,320 K | 1268 | Host Process for Windows S... | Microsoft Corporation |
| spoolsv.exe | 0.03 | 8,220 K | 2,452 K | 1364 | Spooler SubSystem App | Microsoft Corporation |
| svchost.exe | | 9,456 K | 5,128 K | 1400 | Host Process for Windows S... | Microsoft Corporation |
| svchost.exe | | 4,348 K | 2,600 K | 1496 | Host Process for Windows S... | Microsoft Corporation |
| VGAuthService.exe | | 6,240 K | 532 K | 1572 | VMware Guest Authenticatio... | VMware, Inc. |
| vmtoolsd.exe | 0.13 | 11,368 K | 5,052 K | 1620 | VMware Tools Core Service | VMware, Inc. |
| TPAutoConnSvc.exe | 0.02 | 2,444 K | 1,424 K | 1804 | ThinPrint AutoConnect printe... | Cortado AG |
| TPAutoConnect.exe | 0.34 | 2,612 K | 3,048 K | 2848 | ThinPrint AutoConnect comp... | ThinPrint GmbH |
| svchost.exe | | 1,080 K | 1,032 K | 1852 | Host Process for Windows S... | Microsoft Corporation |
| taskhost.exe | | 7,624 K | 2,976 K | 2056 | Host Process for Windows T... | Microsoft Corporation |
| SearchIndexer.exe | 0.01 | 21,256 K | 8,096 K | 2664 | Microsoft Windows Search I... | Microsoft Corporation |
| svchost.exe | | 161,756 K | 17,836 K | 1308 | Host Process for Windows S... | Microsoft Corporation |
| msdtc.exe | | 2,452 K | 424 K | 3548 | Microsoft Distributed Transa... | Microsoft Corporation |
| UI0Detect.exe | | 1,716 K | 4,568 K | 2792 | Interactive services detection | Microsoft Corporation |
| svchost.exe | | 1,332 K | 4,616 K | 1552 | Host Process for Windows S... | Microsoft Corporation |
| lsass.exe | 0.02 | 2,928 K | 3,592 K | 544 | Local Security Authority Proc... | Microsoft Corporation |
| lsm.exe | 0.03 | 1,340 K | 1,140 K | 552 | | |
| csrss.exe | 0.27 | 18,456 K | 11,344 K | 444 | | |
| conhost.exe | | 628 K | 372 K | 2856 | Console Window Host | Microsoft Corporation |
| winlogon.exe | | 1,672 K | 616 K | 492 | | |
| explorer.exe | 0.05 | 32,804 K | 26,612 K | 3600 | Windows Explorer | Microsoft Corporation |
| OLLYDBG.EXE | 0.12 | 12,560 K | 12,904 K | 3012 | OllyDbg, 32-bit analysing deb... | |
| wctsfv.exe | | 17,568 K | 6,904 K | 3168 | Moker | Moker |
| wctsfv.exe | Susp... | 1,088 K | 244 K | 2220 | Moker | Moker |
| mspaint.exe | | 39,016 K | 53,128 K | 2280 | Paint | Microsoft Corporation |
| procexp.exe | 1.86 | 9,760 K | 17,344 K | 3076 | Sysinternals Process Explorer | Sysinternals - www.sysinter... |
| vmtoolsd.exe | 0.17 | 15,016 K | 9,080 K | 2292 | VMware Tools Core Service | VMware, Inc. |

CPU Usage: 4.17% | Commit Charge: 36.50% | Processes: 43 | Physical Usage: 40.86%

Figure 4: Moker as a suspended process

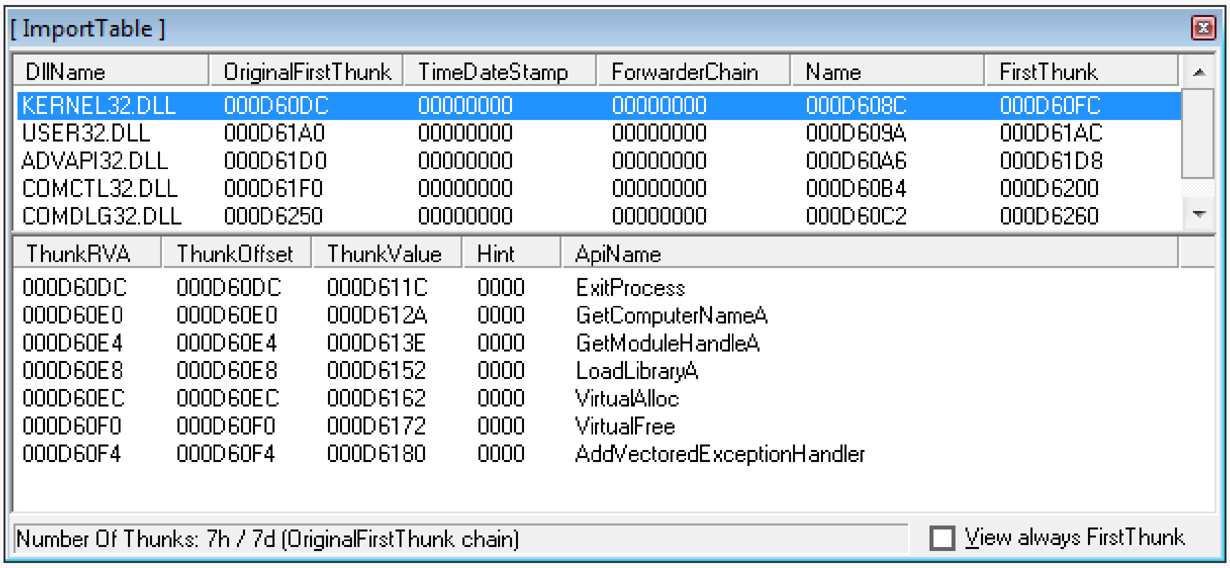
## Challenge #3: Planned exceptions, or anti-debugging revisited

Before resuming the process and letting it run, let's create a dump in order to analyze it.

Dump. Fix sections.

As a first, we need to look at the import table to get a general idea about our unpacking status

As a first, we need to look at the import table to get a general idea about our unpacking status.



[ ImportTable ]

| DllName | OriginalFirstThunk | TimeDateStamp | ForwarderChain | Name | FirstThunk |
|---|---|---|---|---|---|
| KERNEL32.DLL | 000D60DC | 00000000 | 00000000 | 000D608C | 000D60FC |
| USER32.DLL | 000D61A0 | 00000000 | 00000000 | 000D609A | 000D61AC |
| ADVAPI32.DLL | 000D61D0 | 00000000 | 00000000 | 000D60A6 | 000D61D8 |
| COMCTL32.DLL | 000D61F0 | 00000000 | 00000000 | 000D60B4 | 000D6200 |
| COMDLG32.DLL | 000D6250 | 00000000 | 00000000 | 000D60C2 | 000D6260 |

| ThunkRVA | ThunkOffset | ThunkValue | Hint | ApiName |
|---|---|---|---|---|
| 000D60DC | 000D60DC | 000D611C | 0000 | ExitProcess |
| 000D60E0 | 000D60E0 | 000D612A | 0000 | GetComputerNameA |
| 000D60E4 | 000D60E4 | 000D613E | 0000 | GetModuleHandleA |
| 000D60E8 | 000D60E8 | 000D6152 | 0000 | LoadLibraryA |
| 000D60EC | 000D60EC | 000D6162 | 0000 | VirtualAlloc |
| 000D60F0 | 000D60F0 | 000D6172 | 0000 | VirtualFree |
| 000D60F4 | 000D60F4 | 000D6180 | 0000 | AddVectoredExceptionHandler |

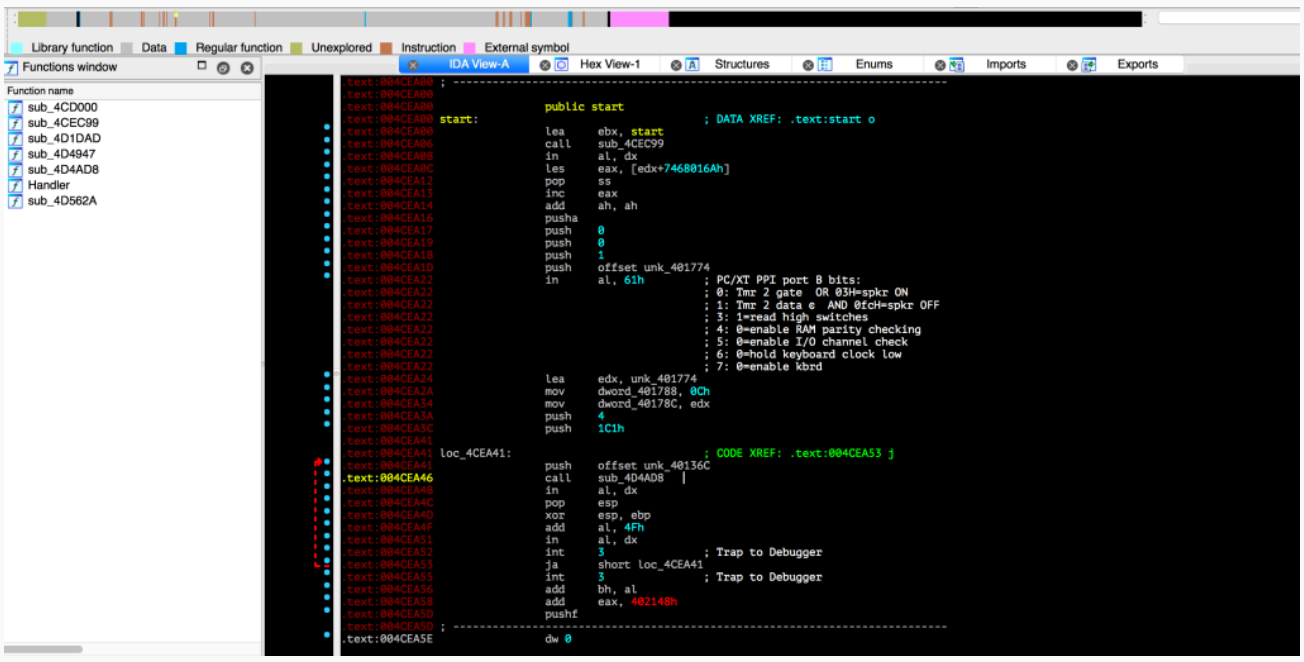
Number Of Thunks: 7h / 7d (OriginalFirstThunk chain)

☐ View always FirstThunk

Figure 5: Import table shows that no fixing needs to be made

Since the import table seems normal, we can open it with IDA.

When we open the dumped code in IDA, we see something strange. IDA finds the start address and parses the code, but the code cannot run very far as it attempts to run “**in *al,dx***“ at 4CEA08 – a privileged instruction. Since this is a user-mode program, it cannot natively run this instruction without generating an exception.



Function name
- sub_4CD000
- sub_4CEC99
- sub_4D1DAD
- sub_4D4947
- sub_4D4AD8
- Handler
- sub_4D562A

```
.text:004CEA00 ; ---------------------------------------------------------------------------
.text:004CEA00
.text:004CEA00                 public start
.text:004CEA00 start:                                  ; DATA XREF: .text:start o
.text:004CEA00                 lea     ebx, start
.text:004CEA06                 call    sub_4CEC99
.text:004CEA08                 in      al, dx
.text:004CEA0C                 les     eax, [edx+7468016Ah]
.text:004CEA12                 pop     ss
.text:004CEA13                 inc     eax
.text:004CEA14                 add     ah, ah
.text:004CEA16                 pusha
.text:004CEA17                 push    0
.text:004CEA19                 push    0
.text:004CEA1B                 push    1
.text:004CEA1D                 push    offset unk_401774
.text:004CEA22                 in      al, 61h         ; PC/XT PPI port B bits:
.text:004CEA22                                         ; 0: Tmr 2 gate  OR 03H=spkr ON
.text:004CEA22                                         ; 1: Tmr 2 data ¢  AND 0fcH=spkr OFF
.text:004CEA22                                         ; 3: 1=read high switches
.text:004CEA22                                         ; 4: 0=enable RAM parity checking
.text:004CEA22                                         ; 5: 0=enable I/O channel check
.text:004CEA22                                         ; 6: 0=hold keyboard clock low
.text:004CEA22                                         ; 7: 0=enable kbrd
.text:004CEA24                 lea     edx, unk_401774
.text:004CEA2A                 mov     dword_401788, 0Ch
.text:004CEA34                 mov     dword_40178C, edx
.text:004CEA3A                 push    4
.text:004CEA3C                 push    1C1h
.text:004CEA41
.text:004CEA41 loc_4CEA41:                             ; CODE XREF: .text:004CEA53 j
.text:004CEA41                 push    offset unk_40136C
.text:004CEA46                 call    sub_4D4AD8
.text:004CEA4B                 in      al, dx
.text:004CEA4C                 pop     esp
.text:004CEA4D                 xor     esp, ebp
.text:004CEA4F                 add     al, 4Fh
.text:004CEA51                 in      al, dx
.text:004CEA52                 int     3               ; Trap to Debugger
.text:004CEA53                 ja      short loc_4CEA41
.text:004CEA55                 int     3               ; Trap to Debugger
.text:004CEA56                 add     bh, al
.text:004CEA58                 add     eax, 402148h
.text:004CEA5D                 pushf
.text:004CEA5D ; ---------------------------------------------------------------------------
.text:004CEA5E                 dw 0
```

Figure 6: IDA Dump shows that the code attempts to run a privileged op-code

The only thing we can run is the call prior to “**in *al,dx***” – sub_4CEC99. What’s in that call? A small function that registers a Vectored Exception Handler. IDA has already identified that and named a “Handler” function on the list.

At this stage, when trying to step through the program with a debugger, the program attempts to execute the privileged instruction and an exception is raised. When we return from the exception Handler, the program calls NtContinue and the debugger loses track of the program.

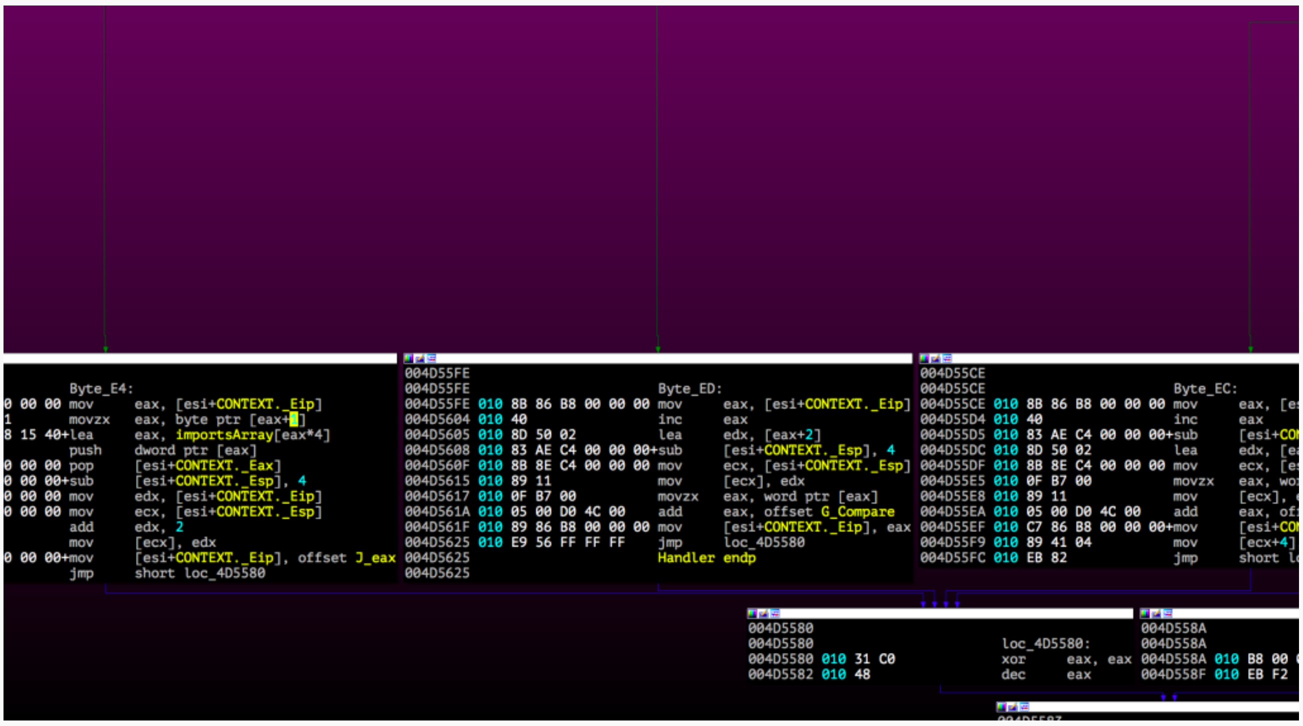
This is an interesting anti-debugging technique that our new in-memory module uses.

It was time to investigate that Handler.

We found out that the Handler reads the byte at the address where the exception occuredand then enters a state machine.

This set of rules reads one or more bytes further, depending on the first byte's value, then alters the CONTEXT fields of the thread in order to simulate a call instruction.

A return address will also be inserted to the stack – to the precise location where our special opcode ends.



```
        Byte_E4:
0 00 00 mov     eax, [esi+CONTEXT._Eip]
1       movzx   eax, byte ptr [eax+1]
8 15 40+lea     eax, importsArray[eax*4]
        push    dword ptr [eax]
0 00 00 pop     [esi+CONTEXT._Eax]
0 00 00+sub     [esi+CONTEXT._Esp], 4
0 00 00 mov     edx, [esi+CONTEXT._Eip]
0 00 00 mov     ecx, [esi+CONTEXT._Esp]
        add     edx, 2
        mov     [ecx], edx
0 00 00+mov     [esi+CONTEXT._Eip], offset J_eax
        jmp     short loc_4D5580
```

```
004D55FE
004D55FE                               Byte_ED:
004D55FE 010 8B 86 B8 00 00 00 mov     eax, [esi+CONTEXT._Eip]
004D5604 010 40                inc     eax
004D5605 010 8D 50 02          lea     edx, [eax+2]
004D5608 010 83 AE C4 00 00 00+sub     [esi+CONTEXT._Esp], 4
004D560F 010 8B 8E C4 00 00 00 mov     ecx, [esi+CONTEXT._Esp]
004D5615 010 89 11             mov     [ecx], edx
004D5617 010 0F B7 00          movzx   eax, word ptr [eax]
004D561A 010 05 00 D0 4C 00    add     eax, offset G_Compare
004D561F 010 89 86 B8 00 00 00 mov     [esi+CONTEXT._Eip], eax
004D5625 010 E9 56 FF FF FF    jmp     loc_4D5580
004D5625                       Handler endp
004D5625
```

```
004D55CE
004D55CE                               Byte_EC:
004D55CE 010 8B 86 B8 00 00 00 mov     eax, [es
004D55D4 010 40                inc     eax
004D55D5 010 83 AE C4 00 00 00+sub     [esi+CON
004D55DC 010 8D 50 02          lea     edx, [ea
004D55DF 010 8B 8E C4 00 00 00 mov     ecx, [es
004D55E5 010 0F B7 00          movzx   eax, wo
004D55E8 010 89 11             mov     [ecx], e
004D55EA 010 05 00 D0 4C 00    add     eax, of
004D55EF 010 C7 86 B8 00 00 00+mov     [esi+CON
004D55F9 010 89 41 04          mov     [ecx+4]
004D55FC 010 EB 82             jmp     short l
```

```
004D5580
004D5580                     loc_4D5580:
004D5580 010 31 C0           xor     eax, eax
004D5582 010 48              dec     eax
```

```
004D558A
004D558A
004D558A 010 B8 00
004D558F 010 EB F2
```

Figure 7: Handler changes the CONTEXT fields

The rules are:

- E4 <byte> – Call a Windows API from the array of API addresses by index <byte>.
- ED <word> – Call a local function by offset Base+<word>
- EC <word> – Call a specific function with an offset Base+<word> as a parameter.

This allows us to find system calls and local functions. This is enough to figure out the intentions of this binary but the analysis cannot be complete without knowing what the EC “call” does.

## Challenge #4: Decrypted code

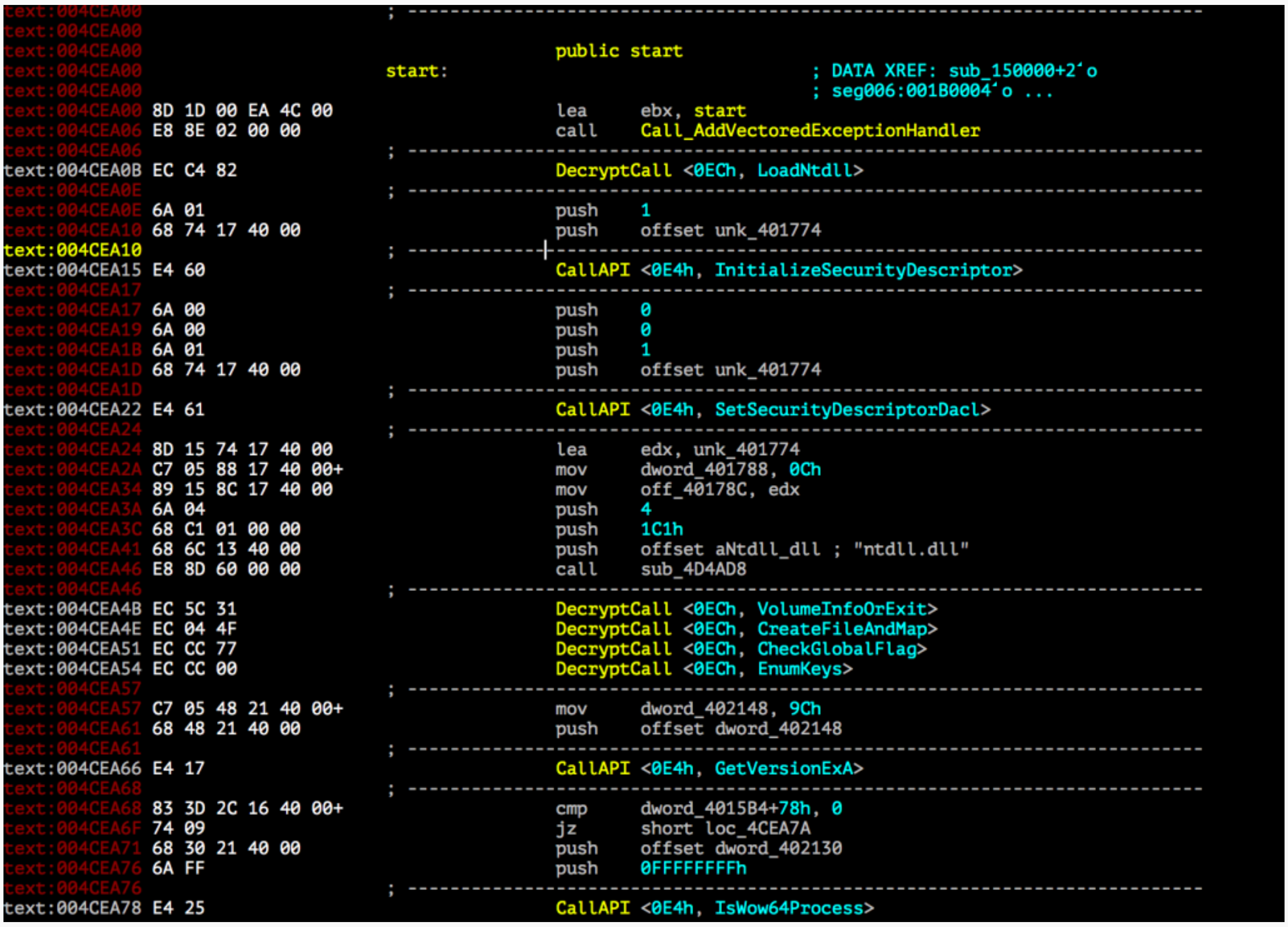
The EC opcode throws an exception to the Handler, as expected. The Handler then changes _eip in the CONTEXT struct of the thread to make the program jump to a function with an offset as a parameter.

The data at this offset does not look useful at its raw form.

Now that we know where the function is, there is no problem running it with a debugger and letting it decrypt the data blob into code and jump to it. However, the algorithm was simple enough to write a decoder script for it.

Finally, we were able to decode the whole program.

We use IDA’s structs and enums to make the exception-handler-calls more readable:



```
text:004CEA00                   ; ---------------------------------------------------------------------------
text:004CEA00
text:004CEA00                                   public start
text:004CEA00                   start:                                  ; DATA XREF: sub_150000+2↑o
text:004CEA00                                                           ; seg006:001B0004↑o ...
text:004CEA00 8D 1D 00 EA 4C 00                 lea     ebx, start
text:004CEA06 E8 8E 02 00 00                    call    Call_AddVectoredExceptionHandler
text:004CEA06                   ; ---------------------------------------------------------------------------
text:004CEA0B EC C4 82                          DecryptCall <0ECh, LoadNtdll>
text:004CEA0E                   ; ---------------------------------------------------------------------------
text:004CEA0E 6A 01                             push    1
text:004CEA10 68 74 17 40 00                    push    offset unk_401774
text:004CEA10                   ; ---------------------------------------------------------------------------
text:004CEA15 E4 60                             CallAPI <0E4h, InitializeSecurityDescriptor>
text:004CEA17                   ; ---------------------------------------------------------------------------
text:004CEA17 6A 00                             push    0
text:004CEA19 6A 00                             push    0
text:004CEA1B 6A 01                             push    1
text:004CEA1D 68 74 17 40 00                    push    offset unk_401774
text:004CEA1D                   ; ---------------------------------------------------------------------------
text:004CEA22 E4 61                             CallAPI <0E4h, SetSecurityDescriptorDacl>
text:004CEA24                   ; ---------------------------------------------------------------------------
text:004CEA24 8D 15 74 17 40 00                 lea     edx, unk_401774
text:004CEA2A C7 05 88 17 40 00+                mov     dword_401788, 0Ch
text:004CEA34 89 15 8C 17 40 00                 mov     off_40178C, edx
text:004CEA3A 6A 04                             push    4
text:004CEA3C 68 C1 01 00 00                    push    1C1h
text:004CEA41 68 6C 13 40 00                    push    offset aNtdll_dll ; "ntdll.dll"
text:004CEA46 E8 8D 60 00 00                    call    sub_4D4AD8
text:004CEA46                   ; ---------------------------------------------------------------------------
text:004CEA4B EC 5C 31                          DecryptCall <0ECh, VolumeInfoOrExit>
text:004CEA4E EC 04 4F                          DecryptCall <0ECh, CreateFileAndMap>
text:004CEA51 EC CC 77                          DecryptCall <0ECh, CheckGlobalFlag>
text:004CEA54 EC CC 00                          DecryptCall <0ECh, EnumKeys>
text:004CEA57                   ; ---------------------------------------------------------------------------
text:004CEA57 C7 05 48 21 40 00+                mov     dword_402148, 9Ch
text:004CEA61 68 48 21 40 00                    push    offset dword_402148
text:004CEA61                   ; ---------------------------------------------------------------------------
text:004CEA66 E4 17                             CallAPI <0E4h, GetVersionExA>
text:004CEA68                   ; ---------------------------------------------------------------------------
text:004CEA68 83 3D 2C 16 40 00+                cmp     dword_4015B4+78h, 0
text:004CEA6F 74 09                             jz      short loc_4CEA7A
text:004CEA71 68 30 21 40 00                    push    offset dword_402130
text:004CEA76 6A FF                             push    0FFFFFFFFh
text:004CEA76                   ; ---------------------------------------------------------------------------
text:004CEA78 E4 25                             CallAPI <0E4h, IsWow64Process>
```

Figure 8: pseudo-code for exception handler calls

## Challenge #5: Process injection

As usual, the malware searches through processes, with known APIs and injects itself into

Explorer.exe as well as to the following three typical Windows' background processes:

- Svchost.exe
- lsass.exe
- csrss.exe

The injected thread tries to load a library using LoadLibraryW with a file inside the user's temp folder, the name is similar to this:

C:\Users\user\AppData\Local\Temp\4FE53.

Notice the "." at the end. Naming a file as anything that ends with a dot causes it to be hidden in Windows Explorer.  This is done to prevent users from finding this file by listing its folder's contents.

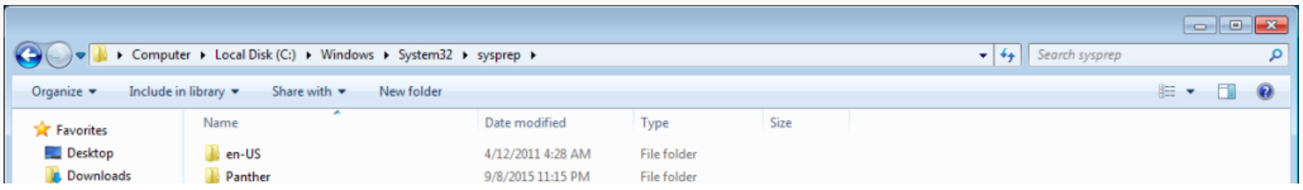
## Challenge #6: Elevation of privileges

To recall, in Windows, some system operations are only available to programs that request elevation. Requesting elevation requires Administrator rights and also pops up the User Account Control (UAC) mechanism that asks the user for permissions to elevate (a.k.a "Allow this program to make changes to this computer?").

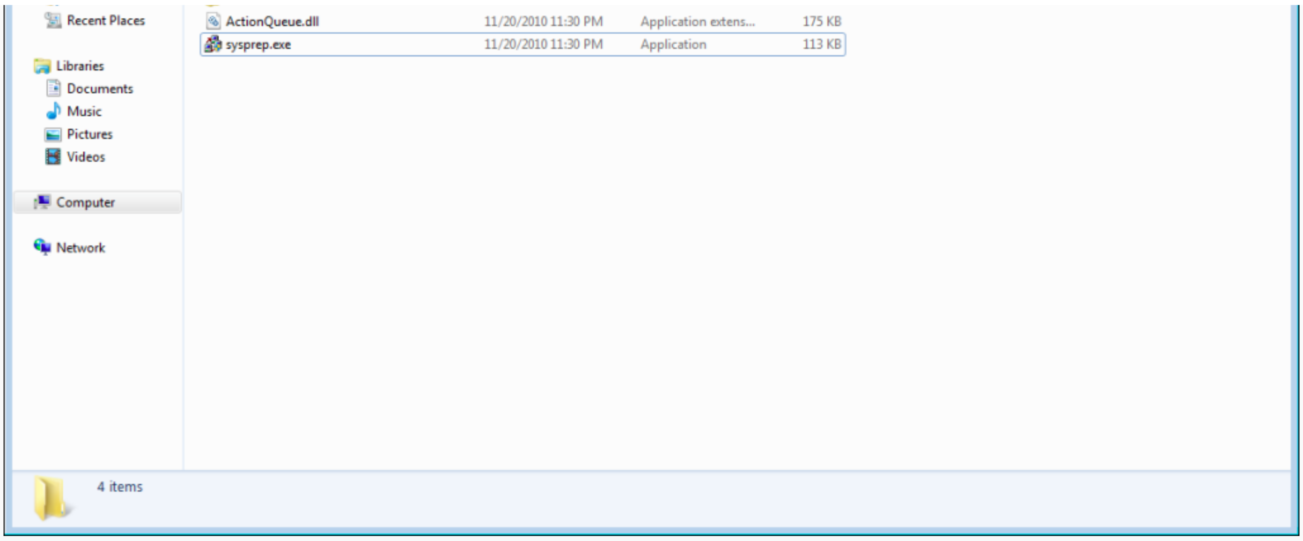
How was Moker able to ensure that it gains system-privileges, without requiring the user's consent?

1. **Leeches on to an elevated program.** In order to enable proper functioning of the Windows system, there are exceptional programs that are always granted elevation permission – without a user prompt. One of these exceptional processes that Moker found is the Microsoft System Preparation Tool that resides in C:\Windows\system32\sysprep.
2. **Elevates the privilege of a DLL.** There is a Windows design vulnerability which enables loading unauthorized DLLs by authorized applications. This vulnerability is found in the way Windows loads DLLs upon request. When an application loads a DLL by its name, Windows first looks through the application's current folder, and then proceeds to search at system directories (and other paths). In order to avoid a predicted chaos using this technique, some DLLs are always loaded from the system directory regardless of any other same named DLLs found in its own path. Monitoring sysprep shows it loading one DLL which does not follow this restriction: ActionQueue.dll.

Putting it all together, Moker writes a file named "ActionQueue.dll" into the "sysprep" directory and then runs sysprep so that the DLL is able to run with elevated privileges.



| Name | Date modified | Type | Size |
|---|---|---|---|
| en-US | 4/12/2011 4:28 AM | File folder | |
| Panther | 9/8/2015 11:15 PM | File folder | |



ActionQueue.dll 11/20/2010 11:30 PM Application extens... 175 KB
sysprep.exe 11/20/2010 11:30 PM Application 113 KB

Figure 9: In the sysprep directory, Moker runs sysrep whereas its executable ActionQueue.dll receives elevated privileges

## Challenge #7: Downloading the payload

We have skipped another part of the story where Moker loaded itself into the sysprep process and injects itself into other processes. That actually happened earlier (challenge #5).

By analyzing the running code, we realize that we actually have a downloader in our hands.

The downloader itself uses one of its injected threads to communicate with some unknown server. Interestingly, at the time of analysis, the address for the server was empty. This could suggest that:

1. The executable was changed since it was first installed
2. The payload is loaded in different ways, not only via the network
3. There is another layer of protection protecting specifically that part of information

At this point let's try to emulate a server response, just to see its program flow.

It uses InternetReadFile, reads the response to a buffer, and saves it to the registry as-is.

This leads us to understand that the payload is in the registry.

Setting a breakpoint at RegQueryValueEx takes us the next step.

At this stage the program reads the registry value, manipulates it (decrypts it), validates that it contains a valid PE header ("MZ", "PE"), and then loads it as a library.

Extracting the registry value from the infected machine and feeding it to the decryption code results in a valid DLL with relatively readable code.

Moker is finally installed, without any external connections.

## Coming Up: analyzing Moker’s capabilities

As this stage, we are ready to look under the hood of Moker.

A first look reveals that Moker features RAT capabilities, a few different techniques to obfuscate strings, a rather sophisticated inter-process communication and synchronization between its components (including the downloader from previous parts).

A distinct feature is its ability to start the Remote Desktop service and allow external access to the machine using Remote Desktop clients (like Windows client or Linux’ ‘rdesktop’).

Since we’re still analyzing the malware, we’ll present the technical details within the next few days.  In the meanwhile, you can find a high-level overview of the capabilities here:

http://blog.ensilo.com/moker-a-new-apt-discovered-within-a-sensitive-network

Like 6 Tweet 22 G+1 0 Pin it

Posted October 6, 2015 by Yotam Gottesman

Category: Malware

Tags: apt, malware, moker, rat

← Injection on Steroids: Code-less Code Injections and 0-Day Techniques