IcedID aka #Bokbot Analysis with Ghidra.

By Dawid Golak

Published: 2019-06-25 · Archived: 2026-04-10 02:06:12 UTC

5 min read

Jun 25, 2019

A few days ago @brad published a post on the twitter about a resume-themed password-protected Word doc that was dropping IcedID (also known as #BokBot). The sample is available for download on the any.run https://app.any.run/tasks/13d6d9f9-7033-4ce7-9ad4-76591f15274c/ service for further analysis.

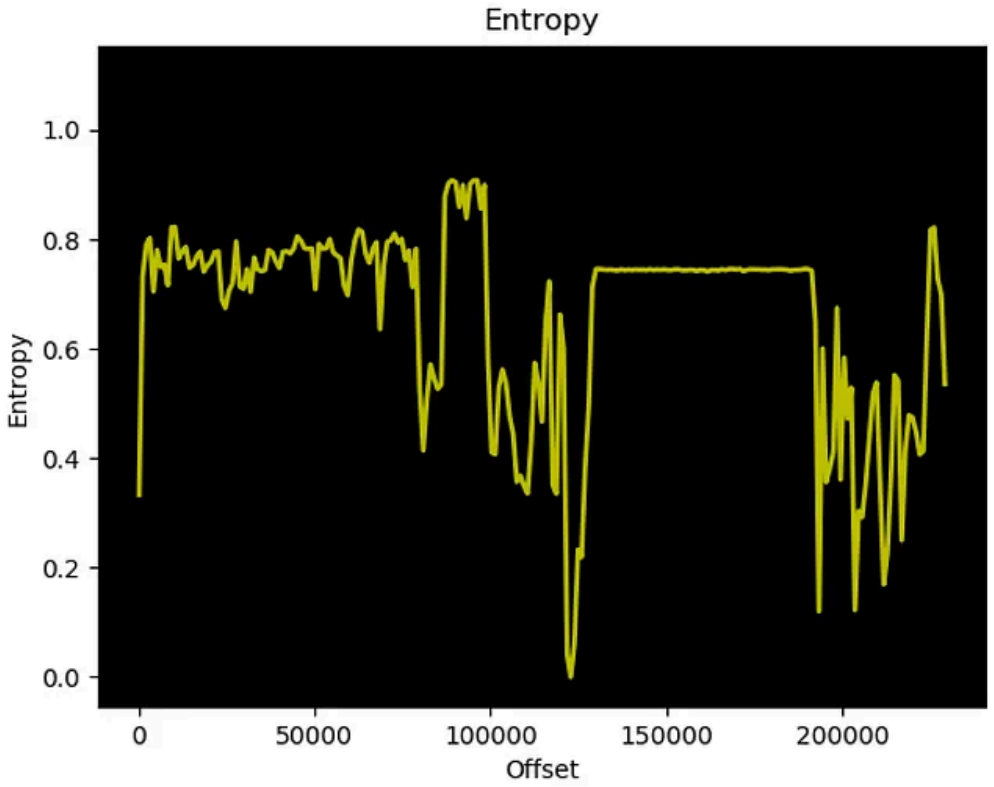
BTW. any.run is the awesome sandbox which can speed up the initial analysis of malicious files.

The IcedID sample was packed and contains an interesting startup mechanism

File analysis:
MD5:b05ea5fd73d25140cdb31f36789d9003
Filename:5.exe
sha256:d350d150f658a32c32984d7f879c6f3b3ddb6ba7918bfe22e19471a79a0cd490

At the first glance you can see, that the file is packed.

Press enter or click to view image in full size



After launching the sample in the debugger we can identify, that a standard packing mechanism is in use.

The first stage, or how to unpack IcedID?

If we want to unpack this file ourselves, then we can look at the OA Labs video tutorial, which is available there or follow the instruction below.

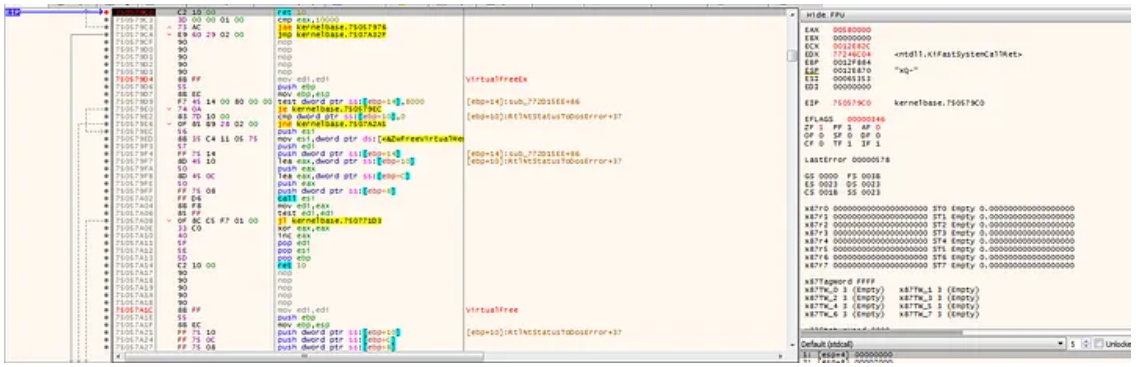
Briefly to unpack this sample:

1. Set breakpoint on VirtualAlloc method

   Press enter or click to view image in full size

2. Launch sample and wait for the debugger to stop on this method, and then look at the allocated memory address (the address is returned by VirtualAlloc $EAX).

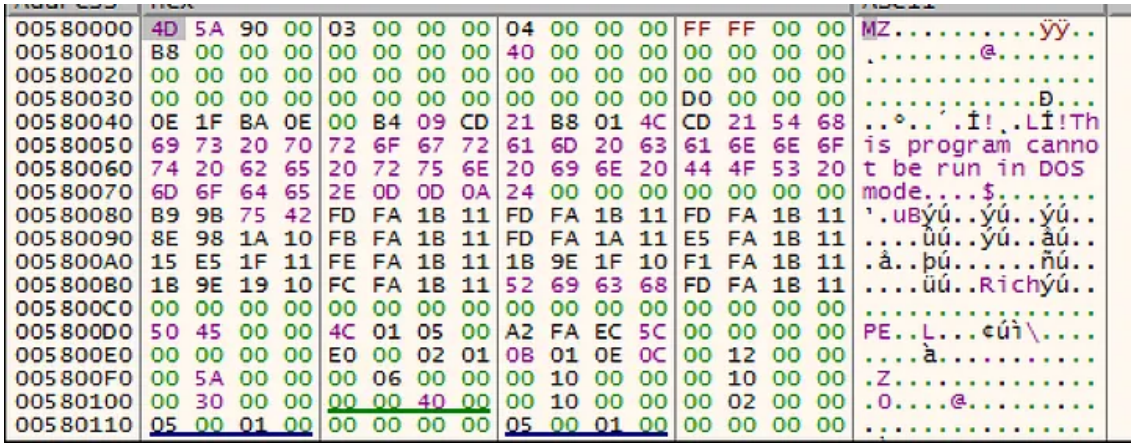
   Press enter or click to view image in full size



Once we have this address, set a breakpoint on the initial bytes of the address and wait until the program write data to this address.

3. As we can see at this point, the initial bytes of the allocated memory area begin with 4D 5A (MZ). It means that there is the second stage of the sample.

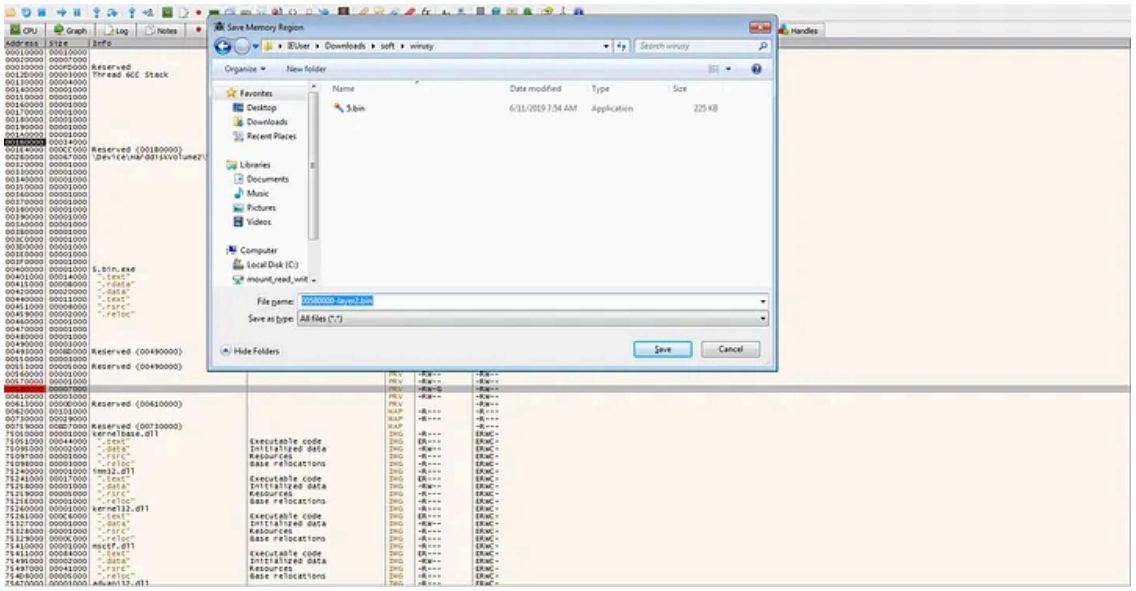
Press enter or click to view image in full size



00580000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....ÿÿ..
00580010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	........@.....
00580020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00580030	00	00	00	00	00	00	00	00	00	00	00	00	D0	00	00	00	.....Ð...
00580040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	..º..´.Í!¸.LÍ!Th
00580050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno
00580060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS
00580070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....$.....
00580080	B9	9B	75	42	FD	FA	1B	11	FD	FA	1B	11	FD	FA	1B	11	'.uBýú..ýú..ýú..
00580090	8E	98	1A	10	FB	FA	1B	11	FD	FA	1A	11	E5	FA	1B	11	....ûú..ýú..åú..
005800A0	15	E5	1F	11	FE	FA	1B	11	1B	9E	1F	10	F1	FA	1B	11	.å..þú.....ñú..
005800B0	1B	9E	19	10	FC	FA	1B	11	52	69	63	68	FD	FA	1B	11	....üú..Richýú..
005800C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
005800D0	50	45	00	00	4C	01	05	00	A2	FA	EC	5C	00	00	00	00	PE..L...¢úì\....
005800E0	00	00	00	00	E0	00	02	01	0B	01	0E	0C	00	12	00	00	....à.....
005800F0	00	5A	00	00	00	06	00	00	00	10	00	00	00	10	00	00	.Z.....
00580100	00	30	00	00	00	00	40	00	00	10	00	00	00	02	00	00	.0....@.....
00580110	05	00	01	00	00	00	00	00	05	00	01	00	00	00	00	00	.....

4. We save the memory to a file and and now we have a copy of the second stage to analyze.

Press enter or click to view image in full size



The second stage

After unpacking the file and then previewing the generated pseudocode in #Ghidra, we can see the following flow.

Press enter or click to view image in full size

```
2  undefined4 entry(int param_1,int param_2,char *param_3)
3
4  {
5    char cVar1;
6    byte bVar2;
7    LPSTR pCVar3;
8    LPSTR value_of_arg1;
9    int iVar4;
10   char *magic_string;
11   uint uVar5;
12   uint uVar6;
13
14   pCVar3 = GetCommandLineA();
15   value_of_arg1 = obtain_argv_q(pCVar3);
16   if (value_of_arg1 == (LPSTR)0x0) {
17     iVar4 = obtain_argv_param_p(pCVar3);
18     if ((iVar4 != 0) && (magic_string = FUN_004011be(), magic_string != (char *)0x0)) {
19       Sleep(1000);
20       create_self_process_with_additional_params(magic_string);
21     }
22   }
23   else {
24     create_svchost_process();
25   }
26   ExitProcess(0);
```

In line №15, the argument from parameter “-q=” is retrieved.

If this parameter is not present, the code beginning on line 17 is started.Further parameters are checked.

Why does he do it ?

The process under debugger, creates the a new process which is not debugged. It is one of the way to escape from the debugger.

Press enter or click to view image in full size

We have several options to debug the new process.

Ps. For instance I looked at what parameters it is run “CreateProcessA” and again execute the malware with the additional option under the debugger.
“C:\Users\admin\AppData\Local\Temp\5.exe” -q=[int]”

for example:
“C:\Users\admin\AppData\Local\Temp\5.exe” -q=412588568”

Get Dawid Golak’s stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

Once these argument checks have passed the next interested function to analyze is FUN_004011be().

Press enter or click to view image in full size

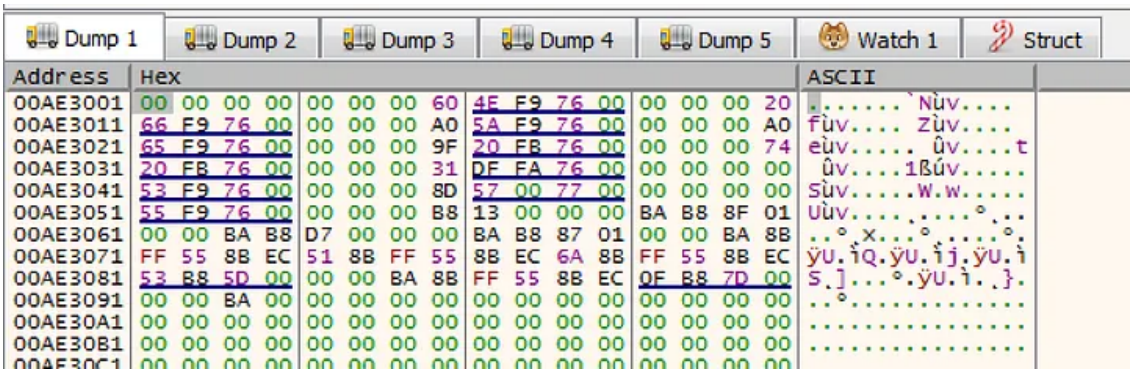
```
                    char * __stdcall FUN_004011be(void)
      char *          EAX:4          <RETURN>
                    FUN_004011be                                XREF[1]:     entry:00401026(c)
004011be 68 aa 08       PUSH        0x8aa
         00 00
004011c3 e8 39 0c       CALL        allocate_heap_memory                     undefined allocate_heap_memory(S...
         00 00
004011c8 8b d0          MOV         EDX,EAX
004011ca 59             POP         ECX
004011cb 85 d2          TEST        EDX,EDX
004011cd 75 01          JNZ         LAB_004011d0
004011cf c3             RET

                    LAB_004011d0                                XREF[1]:     004011cd(j)
004011d0 56             PUSH        ESI
004011d1 8b ca          MOV         ECX,EDX
004011d3 33 f6          XOR         ESI,ESI

                    LAB_004011d5                                XREF[1]:     00401204(j)
004011d5 0f b6 86       MOVZX       EAX,byte ptr [CHAR_??_00403000 + ESI]      = ??
         00 30 40 00
004011dc c1 e8 04       SHR         EAX,0x4
004011df 8a 80 70       MOV         AL,byte ptr [s_0123456789ABCDEF_00404070 + EAX]  = "0123456789ABCDEF"
         40 40 00
004011e5 88 01          MOV         byte ptr [ECX],AL
004011e7 8d 49 02       LEA         ECX,[ECX + 0x2]
004011ea 0f b6 86       MOVZX       EAX,byte ptr [CHAR_??_00403000 + ESI]      = ??
         00 30 40 00
004011f1 83 e0 0f       AND         EAX,0xf
004011f4 46             INC         ESI
004011f5 8a 80 70       MOV         AL,byte ptr [s_0123456789ABCDEF_00404070 + EAX]  = "0123456789ABCDEF"
         40 40 00
004011fb 88 41 ff       MOV         byte ptr [ECX + -0x1],AL
004011fe 81             ??          81h
004011ff fe             ??          FEh
00401200 54             PUSH        ESP
00401201 04 00          ADD         AL,0x0
00401203 00             ??          00h
00401204 72 cf          JC          LAB_004011d5
00401206 c6 01 00       MOV         byte ptr [ECX],0x0
00401209 8b c2          MOV         EAX,EDX
0040120b 5e             POP         ESI
0040120c c3             RET
```

This function contains a simple decryption algorithm, that begins at address 004011d5. Here there is a loop, which gets value from the address [00403000+ESI].

Press enter or click to view image in full size



Then the bitwise shift operation is performed to the right. Then the lower bits of $EAX register is downloaded. The value of the AL register indicates the element number from the second data set “0123456789ABCDEF”

In the next step, the value from the address [00403000 + ESI] is taken again and the “AND” operation is performed.

Press enter or click to view image in full size

```
004011f1 83 e0 0f        AND        EAX,0xf
004011f4 46              INC        ESI
004011f5 8a 80 70        MOV        AL,byte ptr [s_0123456789ABCDEF_00404070
         40 40 00
004011fb 88 41 ff        MOV        byte ptr [ECX + -0x1],AL
```

AND EAX,0xf, the retrieved is lower bits of $EAX register
The AL register value indicates the element number from the second data set **“0123456789ABCDEF”**

Press enter or click to view image in full size

```
00AE4070 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 0123456789ABCDEF
00AE4080 00 00 00 00 2D 71 3D 00 25 75 00 00 20 2D 71 3D ....-q=.%u.. -q=
00AE4090 00 00 00 00 00 00 00 00 94 44 5C 27 6C 4F 7C 99 .........D\'lO|.
00AE40A0 86 06 42 E0 3F B7 46 36 08 81 4C B3 59 27 1A 87 ..Bà?·F6..L³Y'..
00AE40B0 96 68 BE CA 91 17 2F 8F 55 B7 C7 F6 E0 72 B5 FA .h¾Ê../.U·Çöàrµú
00AE40C0 FD 72 6C 90 58 D6 3B 26 08 9D 4B B6 45 21 1D 96 ýrl.XÖ;&..K¶E!..
00AE40D0 D6 7F E2 C6 FD 72 20 E0 22 A5 DF F9 E4 66 B2 FC Ö.âÆýr à"¥ßùäf²ü
00AE40E0 E6 28 3A 99 55 D6 2A 54 69 F3 38 DB 2C 55 69 F3 æ(:.UÖ*Tió8Û,Uió
00AE40F0 B8 1B CC A5 91 07 42 E0 2F D6 AA 94 89 03 C0 9F ..Ì¥..Bà/Öª...À.
00AE4100 8E 06 42 E0 2F D6 2A 54 69 F3 38 DB 2C 55 69 F3 ..Bà/Ö*Tió8Û,Uió
00AE4110 B8 1B CC A5 91 07 42 E0 2F D6 AA 94 89 03 C0 9F ..Ì¥..Bà/Öª...À.
00AE4120 8E 06 42 E0 2F D6 2A 54 69 F3 38 DB 2C 55 69 F3 ..Bà/Ö*Tió8Û,Uió
00AE4130 B8 1B CC A5 91 07 42 E0 2F D6 AA 94 89 03 C0 9F ..Ì¥..Bà/Öª...À.
00AE4140 8E 06 42 E0 2F D6 2A 54 69 F3 38 DB 2C 55 69 F3 ..Bà/Ö*Tió8Û,Uió
00AE4150 B8 1B CC A5 91 07 42 E0 2F D6 AA 94 89 03 C0 9F ..Ì¥..Bà/Öª...À.
```

This algorithm has been replicated in python below.

```
key=file(‘data3.txt’).read()
tab=file(‘data2.txt’).read()
ind = 0
res = []
p=””
for i in key:
first_poz = ord(i)>>4
second_poz = ord(i)&0xf
res.append(tab[first_poz])
res.append(tab[second_poz])
ind+=1
if ind > 1107: # while (uVar5 < 0x454);
break
for i in res:
p=p+i
print(p)
```

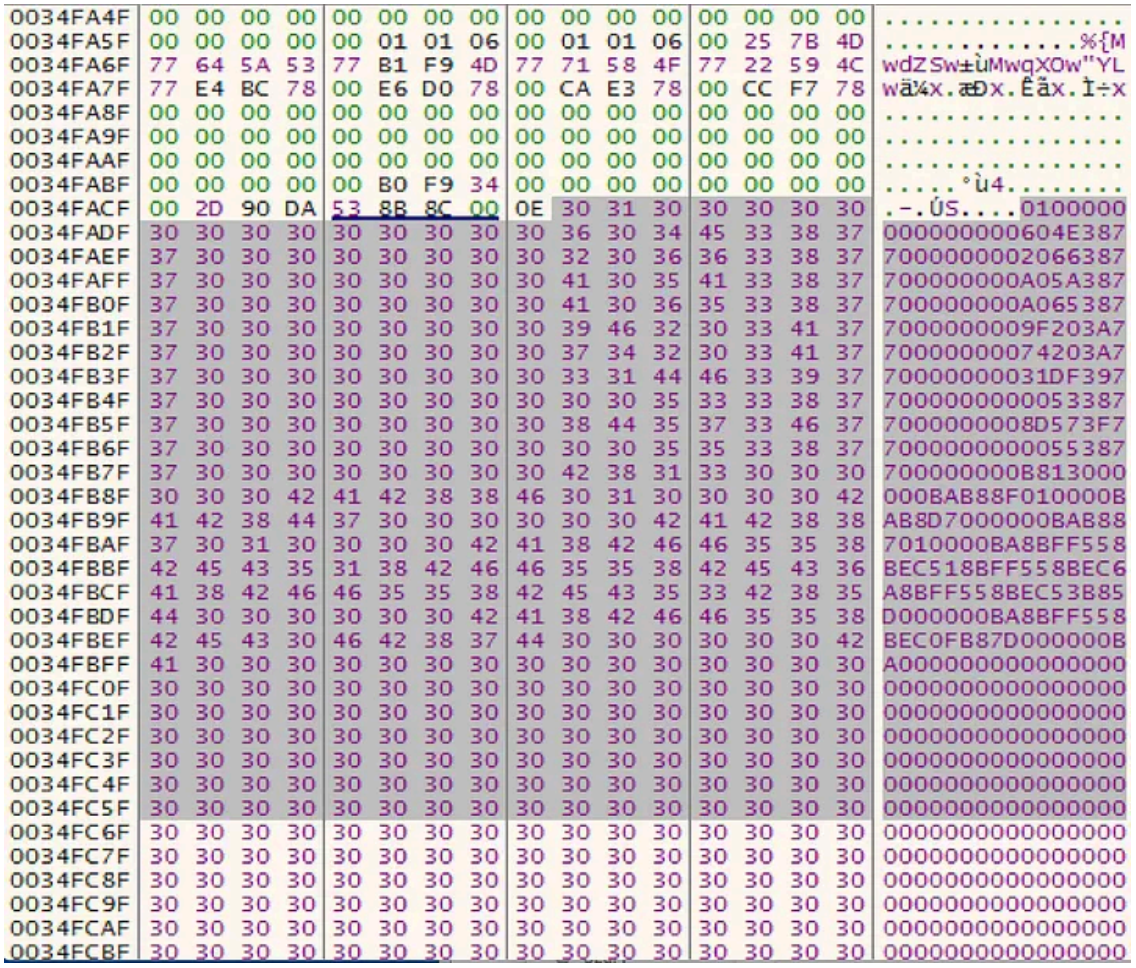
An example of the output is shown below.

Press enter or click to view image in full size

```
bash-3.2$ python ./decrypt.py
0100000000000000604E3877000000002066387700000000A0
5A3877000000A06538770000000009F203A77000000007420
3A770000000031DF397700000000005338770000000008D573F
770000000005538770000000B813000000BAB88F010000BA
B8D7000000BAB887010000BA8BFF558BEC518BFF558BEC6A8B
FF558BEC53B85D000000BA8BFF558BEC0FB87D000000BA0000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000043
003A005C00550073006500720073005C0049004500550073 00
650072005C0044006F0077006E006C006F006100640073005C
0073006F00660074005C007700690072007500730079005C00
330035003000300030003002D006C006100790065007200 2D
0032002E00620069006E0000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000009444
5C276C4F7C99860642E03FB7463608814CB359271A87966BBE
CA91172F8F55B7C7F6E072B5FAFD726C9058D63B26089D4BB6
45211D96D67FE2C6FD7220E022A5DFF9E466B2FCE6283A9955
D62A5469F338DB2C5569F3B81BCCA5910742E02FD6AA948903
C09F8E0642E02FD62A5469F338DB2C5569F3B81BCCA5910742
E02FD6AA948903C09F8E0642E02FD62A5469F338DB2C5569F3
B81BCCA5910742E02FD6AA948903C09F8E0642E02FD62A5469
F338DB2C5569F3B81BCCA5910742E02FD6AA948903C09F8E06
42E02FD62A5469F338DB2C5569F3B81BCCA5910742E02FD6AA
948903C09F8E0642E02FD62A5469F338DB2C5569F3B81BCCA5
910742E000000000
```

Using the debugger we can verify the output from our script.

Press enter or click to view image in full size



0034FA4F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0034FA5F	00	00	00	00	00	01	01	06	00	01	01	06	00	25	7B	4D	..............%{M
0034FA6F	77	64	5A	53	77	B1	F9	4D	77	71	58	4F	77	22	59	4C	wdZSw±ùMwqXOw"YL
0034FA7F	77	E4	BC	78	00	E6	D0	78	00	CA	E3	78	00	CC	F7	78	wä¼x.æÐx.Êãx.Ì÷x
0034FA8F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0034FA9F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0034FAAF	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0034FABF	00	00	00	00	00	B0	F9	34	00	00	00	00	00	00	00	00	.....°ù4........
0034FACF	00	2D	90	DA	53	8B	8C	00	0E	30	31	30	30	30	30	30	.-.ÚS....0100000
0034FADF	30	30	30	30	30	30	30	30	30	36	30	34	45	33	38	37	000000000604E387
0034FAEF	37	30	30	30	30	30	30	30	30	32	30	36	36	33	38	37	7000000002066387
0034FAFF	37	30	30	30	30	30	30	30	30	41	30	35	41	33	38	37	700000000A05A387
0034FB0F	37	30	30	30	30	30	30	30	30	41	30	36	35	33	38	37	700000000A065387
0034FB1F	37	30	30	30	30	30	30	30	30	39	46	32	30	33	41	37	7000000009F203A7
0034FB2F	37	30	30	30	30	30	30	30	30	37	34	32	30	33	41	37	7000000074203A7
0034FB3F	37	30	30	30	30	30	30	30	30	33	31	44	46	33	39	37	70000000031DF397
0034FB4F	37	30	30	30	30	30	30	30	30	30	30	35	33	33	38	37	7000000000053387
0034FB5F	37	30	30	30	30	30	30	30	30	38	44	35	37	33	46	37	7000000008D573F7
0034FB6F	37	30	30	30	30	30	30	30	30	30	30	35	35	33	38	37	7000000000055387
0034FB7F	37	30	30	30	30	30	30	30	30	42	38	31	33	30	30	30	700000000B813000
0034FB8F	30	30	30	42	41	42	38	38	46	30	31	30	30	30	30	42	000BAB88F010000B
0034FB9F	41	42	38	44	37	30	30	30	30	30	30	42	41	42	38	38	AB8D7000000BAB88
0034FBAF	37	30	31	30	30	30	30	42	41	38	42	46	46	35	35	38	7010000BA8BFF558
0034FBBF	42	45	43	35	31	38	42	46	46	35	35	38	42	45	43	36	BEC518BFF558BEC6
0034FBCF	41	38	42	46	46	35	35	38	42	45	43	35	33	42	38	35	A8BFF558BEC53B85
0034FBDF	44	30	30	30	30	30	30	42	41	38	42	46	46	35	35	38	D000000BA8BFF558
0034FBEF	42	45	43	30	46	42	38	37	44	30	30	30	30	30	30	42	BEC0FB87D000000B
0034FBFF	41	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	A000000000000000
0034FC0F	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	0000000000000000
0034FC1F	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	0000000000000000
0034FC2F	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	0000000000000000
0034FC3F	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	0000000000000000
0034FC4F	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	0000000000000000
0034FC5F	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	0000000000000000
0034FC6F	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	0000000000000000
0034FC7F	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	0000000000000000
0034FC8F	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	0000000000000000
0034FC9F	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	0000000000000000
0034FCAF	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	0000000000000000
0034FCBF	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	0000000000000000

Afterwards, the **“q”** parameter is generated.

Press enter or click to view image in full size

```
Decompile: obtain_argv_q - (350000-layer-2.bin)
1
2  LPSTR __cdecl obtain_argv_q(LPCSTR param_1)
3
4  {
5    LPSTR argv_1;
6    int iVar1;
7    LPSTR lpBuffer;
8    CHAR value_of_argv1 [32];
9
10   argv_1 = StrStrA(param_1,"-q=");
11   if (argv_1 != (LPSTR)0x0) {
12     iVar1 = StrToIntA(argv_1 + 3);
13     wsprintfA(value_of_argv1,"%u",iVar1);
14     lpBuffer = (LPSTR)allocate_heap_memory(0x8e8);
15     argv_1 = lpBuffer;
16     if (lpBuffer != (LPSTR)0x0) {
17       argv_1 = (LPSTR)GetEnvironmentVariableA(value_of_argv1,lpBuffer,0x8e8);
18       if (argv_1 != (LPSTR)0x0) {
19         argv_1 = (LPSTR)FUN_0040104c((int)lpBuffer,0x454,&CHAR_??_00403000);
20       }
21       heap_free_memory(lpBuffer);
22     }
23   }
24   return argv_1;
25 }
26
```

The next step is to run the process again with the parameters that were checked in the above conditions (-q=[int]). In the second start of the process, after passing the conditions, we come to the function.

create_self_process_with_additional_params() (org. FUN_0040124a()), which launches a new process of svchost.exe

Press enter or click to view image in full size

```
                    make_svchost.exe_string                         XREF[1]:
0040122d 8b 44 24 04     MOV        EAX,dword ptr [ESP + param_1]
                    svchost.exe
00401231 c7 00 5c        ds         C7,"\0\\svc",C7,"@",04,"host",C7,"@\b.exe"
         73 76 63
         c7 40 04 ...
00401245 c6 40 0c 00     MOV        byte ptr [EAX + 0xc],0x0
00401249 c3              RET
```

Anti-debugging, or GetNativeSystemInfo.

Before the creation of the svchost process the malware uses an anti-debugging method. Malware uses a known method to this end called “GetNativeSystemInfo” (User32.dll).

Press enter or click to view image in full size

Decompile: FUN_00401706 - (350000-layer-2.bin)

```
1
2  uint FUN_00401706(void)
3
4  {
5    short local_28 [18];
6
7    FUN_00401e58((undefined *)local_28,0x24);
8    GetNativeSystemInfo((LPSYSTEM_INFO)local_28);
9    return (uint)(local_28[0] == 9);
10 }
11
```

The method is called in a function FUN_00401706(), which is called in FUN_004015a9()

Press enter or click to view image in full size

Decompile: FUN_004015a9 - (350000-layer-2.bin)

```
1
2  /* WARNING: Globals starting with '_' overlap smaller symbols at the same address */
3
4  void FUN_004015a9(void)
5
6  {
7    uint uVar1;
8    int *piVar2;
9    uint uVar3;
10   uint uVar4;
11   uint uVar5;
12   uint uVar6;
13   uint uVar7;
14   uint uVar8;
15   uint uVar9;
16   uint uVar10;
17   uint uVar11;
18   int iVar12;
19   longlong lVar13;
20
21   _DAT_00403098 = 0;
22   uVar1 = FUN_00401706();
23   if (uVar1 != 0) {
24     lVar13 = empty_fun();
25     iVar12 = (int)((ulonglong)lVar13 >> 0x20);
26     uVar1 = (uint)lVar13;
27     if ((lVar13 != 0) && (piVar2 = (int *)allocate_heap_memory2(uVar1,iVar12), piVar2 != (int *)0x0)
28        ) {
29       uVar3 = FUN_0040142d(piVar2,uVar1,iVar12,0xed46dfd2,(int *)&DAT_004030e0,(int)&DAT_00403120);
30       uVar4 = FUN_0040142d(piVar2,uVar1,iVar12,0x434c7242,(int *)&DAT_004030e8,(int)&DAT_00403126);
31       uVar5 = FUN_0040142d(piVar2,uVar1,iVar12,0xd510f438,(int *)&DAT_004030a8,(int)&DAT_004030f6);
32       uVar6 = FUN_0040142d(piVar2,uVar1,iVar12,0xbe06b948,(int *)&DAT_004030b8,(int)&DAT_00403102);
33       uVar7 = FUN_0040142d(piVar2,uVar1,iVar12,0x3d3fb58a,(int *)&DAT_004030d0,(int)&DAT_00403114);
34       uVar8 = FUN_0040142d(piVar2,uVar1,iVar12,0xc33bb247,(int *)&DAT_004030a0,(int)&DAT_004030f0);
35       uVar9 = FUN_0040142d(piVar2,uVar1,iVar12,0xe4f137ca,(int *)&DAT_004030d8,(int)&DAT_0040311a);
```

According to the documentation MSDN we can obtain SYSTEM_INFO structure

Press enter or click to view image in full size

SYSTEM_INFO structure

Contains information about the current computer system. This includes the architecture and type of the processor, the number of processors in the system, the page size, and other such information.

Syntax

```
typedef struct _SYSTEM_INFO {
  union {
    DWORD  dwOemId;
    struct {
      WORD wProcessorArchitecture;
      WORD wReserved;
    };
  };
  DWORD     dwPageSize;
  LPVOID    lpMinimumApplicationAddress;
  LPVOID    lpMaximumApplicationAddress;
  DWORD_PTR dwActiveProcessorMask;
  DWORD     dwNumberOfProcessors;
  DWORD     dwProcessorType;
  DWORD     dwAllocationGranularity;
  WORD      wProcessorLevel;
  WORD      wProcessorRevision;
} SYSTEM_INFO;
```

The code shows that the returned value is compared with 0x9 which means the comparison of the processor architecture.

PROCESSOR_ARCHITECTURE_AMD64x64 (AMD or Intel) = 9

Press enter or click to view image in full size

wProcessorArchitecture

The processor architecture of the installed operating system. This member can be one of the following values.

Value	Meaning
PROCESSOR_ARCHITECTURE_AMD64 9	x64 (AMD or Intel)
PROCESSOR_ARCHITECTURE_ARM 5	ARM
PROCESSOR_ARCHITECTURE_ARM64 12	ARM64
PROCESSOR_ARCHITECTURE_IA64 6	Intel Itanium-based
PROCESSOR_ARCHITECTURE_INTEL 0	x86
PROCESSOR_ARCHITECTURE_UNKNOWN 0xffff	Unknown architecture.

Once the malware executes the svchost process it injects a final stage of itself into the process. I will cover this technique and continue out analysis in Part Two.
