

Manual Unpacking IcedID Write-up

 kienmanowar.wordpress.com/2020/08/16/manual-unpacking-icedid-write-up/

August 16, 2020

Sample hash:

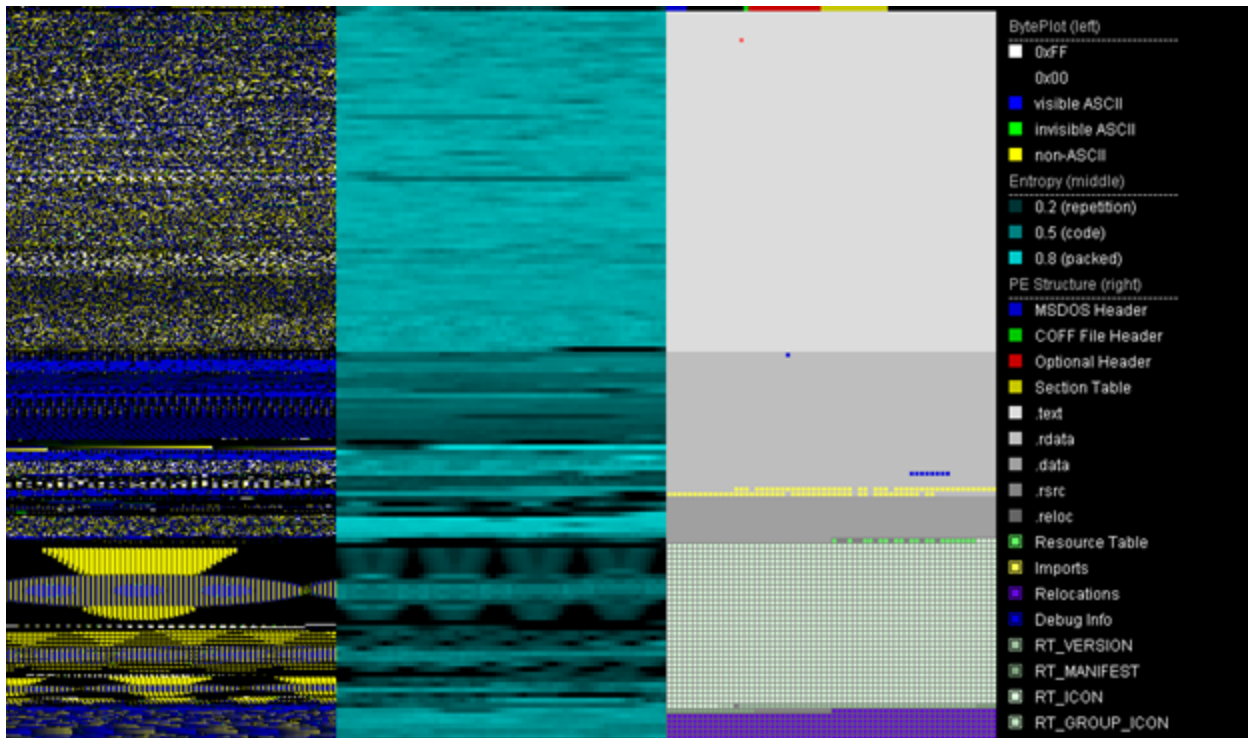
SHA256: 76cd290b236b11bd18d81e75e41682208e4c0a5701ce7834a9e289ea9e06eb7e

Tools:

- PE files static analysis: [PortExAnalyzer](#); [PE-bear](#)
- Debugger & plugin: [x64dbg](#) + [ScyllaHide Anti-Anti-Debug](#)
- Aplib decompress: [aplib-ripper](#)

1. Static Analysis

Thow the sample to **PortEx Analyzer**, tool will analyse file with a special focus on malformation. We get the results:



The section **.text** has high entropy, so may be the sample is packed:

```

Section Table
*****

```

	1. .text	2. .rdata	3. .data	4. .rsrc
Entropy	6.74	4.97	5.29	3.73
Pointer To Raw Data	0x400	0x10c00	0x17c00	0x19e00
Size Of Raw Data	0x10800	0x7000	0x2200	0x8400
Physical End	0x10c00	0x17c00	0x19e00	0x22200
Virtual Address	0x1000	0x12000	0x19000	0x34000
Virtual Size	0x106ab	0x6f4c	0x1a220	0x831c
-> actual virtual size	0x11000	0x7000	0x1b000	0x9000
Pointer To Relocations	0x0	0x0	0x0	0x0
Number Of Relocations	0x0	0x0	0x0	0x0
Pointer To Line Numbers	0x0	0x0	0x0	0x0
Number Of Line Numbers	0x0	0x0	0x0	0x0
Code	x			
Initialized Data		x	x	x
Execute	x			
Read	x	x	x	x
Write			x	

This sample is PE32 with **ASLR enabled** (can quickly disable this feature by using `setdllcharacteristics`):

Magic Number: PE32, normal executable file

Entry Point is in section 1 with name .text

DLL Characteristics * DLL can be relocated at load time.

* Image is NX compatible.

* Terminal Server aware.

Subsystem: The Windows graphical user interface (GUI) subsystem

This sample reveals information about the pdb path:

Debug Information

Time Date Stamp: Tue Feb 10 17:51:45 ICT 2015

Type: Visual C++ debug information

description	value	file offset
Characteristics	0x0	0x10d70
Time/Date Stamp	0x54d9e2c1	0x10d74
Major Version	0x0	0x10d78
Minor Version	0x0	0x10d7a
Type	0x2	0x10d7c
Size of Data	0x7e	0x10d80
Address of Raw Data	0x17ef8	0x10d84
Pointer to Raw Data	0x16af8	0x10d88

Codeview

Age: 1

GUID: 028afe6d-fecb-4ffb-862f-b9d8251b493f

File: c:\Sizeanger\CreatePick\mixpractice\Sciencescience\KeyContain\farterm\Tiresubtract\CenterSkinMass.pdb

Some anomalies were identified by **PortEx**:

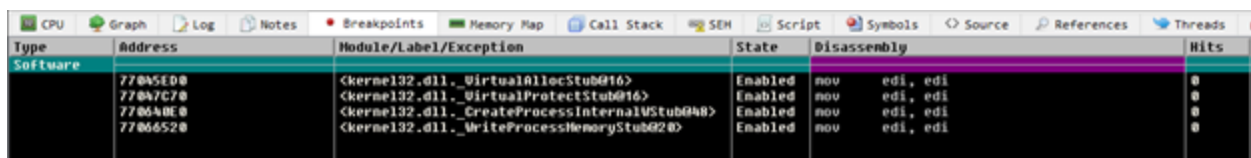
Anomalies

- * Import function typical for code injection: VirtualProtectEx may set PAGE_EXECUTE flag for memory region
- * Import function typical for code injection: CreateThread is used to open and execute a thread in the victim process

2. Dynamic Analysis

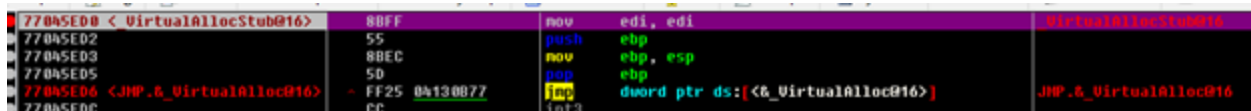
Load specimen to **x64dbg**, for unpacking process, we set breakpoints at some common APIs:

- VirtualAlloc
- VirtualProtect
- CreateProcessInternalW
- WriteProcessMemory



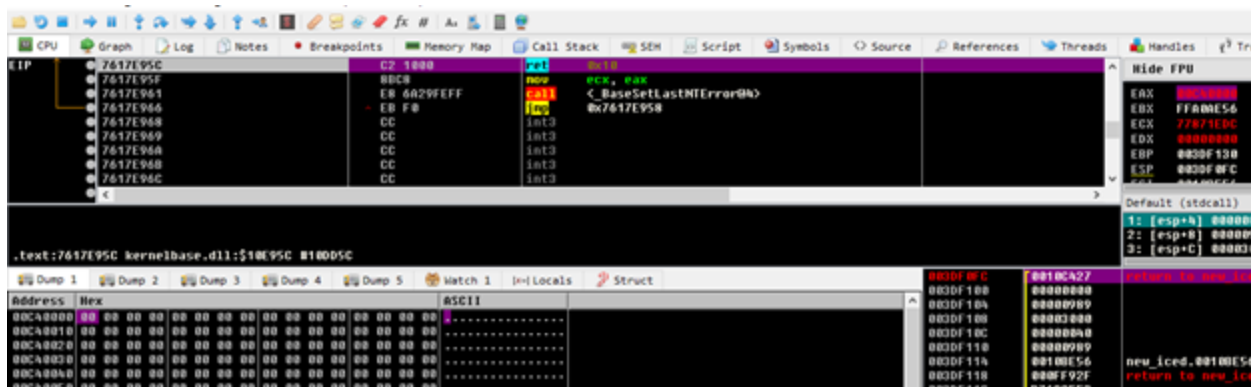
Type	Address	Module/Label/Exception	State	Disassembly	Hits
Software	770A5E08	<kernel32.dll._VirtualAllocStub@16>	Enabled	mov edi, edi	0
Software	770A7C78	<kernel32.dll._VirtualProtectStub@16>	Enabled	mov edi, edi	0
Software	770A4BE8	<kernel32.dll._CreateProcessInternalWStub@16>	Enabled	mov edi, edi	0
Software	77066528	<kernel32.dll._WriteProcessMemoryStub@20>	Enabled	mov edi, edi	0

After placing the breakpoints like above picture, press **F9** to execute. First hit at **VirtualAlloc** :



Address	Disassembly	Comment
770A5E08	mov edi, edi	VirtualAllocStub@16
770A5E02	push ebp	
770A5E03	mov esp, ebp	
770A5E05	pop ebp	
770A5E06	jmp &_VirtualAlloc@16	JMP.&_VirtualAlloc@16

Execute till Return (**Ctrl+F9**) and Follow in dump the allocated memory (return in **EAX** register):



Address	Hex	ASCII
00C40000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C40010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C40020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C40030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C40040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C40050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Continue run with **F9** , hit the second call to **VirtualAlloc** and observe changes in the allocated memory. We see new bytes value was written to this location and it is likely a shellcode:

```

ESI  00C40000  90      nop
      00C40001  E8 00000000  call 0xC40006
      00C40006  5B      nop     ebx
      00C40007  8D43 31    lea   eax, dword ptr ds:[ebx+0x31]
      00C4000A  BF 04F6027B  mov   edi, 0x7B02F604
      00C4000F  B9 4C090000  mov   ecx, 0x94C
      00C40014  89FA    mov   edx, edi
      00C40016  31DB    xor   ebx, ebx
      00C40018  89CE    mov   esi, ecx

```

```

00C40000
Dump 1  Dump 2  Dump 3  Dump 4  Dump 5  Watch 1  [x] Locals  Struct
Address Hex ASCII
00C40000 90 E8 00 00 00 00 5B 8D 43 31 BF 04 F6 02 7B B9  à...[.C1.ø.{'
00C40010 4C 09 00 00 89 FA 31 DB 89 CE 83 E6 03 75 0A 89  L...ú10.1.æ.u..
00C40020 FB 66 01 DA C1 CA 03 89 D7 30 10 40 C1 CA 08 E2  ÕF.ÚÁÈ...x0.0ÁÈ.â
00C40030 E7 E9 BC 04 00 00 90 68 5C 34 2E 89 09 00 00 B8  çé%...h\4.....
00C40040 1A 00 00 00 1A 00 00 D3 0C 00 00 50 7C 01 00 1C  :.:ú.m.....
00C40050 00 00 00 3A DA 1C 6D 0E 18 0E 05 03 02 08 00 06  :.:ú.m.....
00C40060 03 03 00 00 00 04 14 07 08 0F 05 07 00 00 00 F8  :.:ú.m.....
00C40070 97 01 00 00 00 00 00 07 00 00 00 00 00 0F 00 3C  :.:ú.m.....
00C40080 D1 38 00 FA 8B 34 00 42 31 0E 00 8E 18 07 00 AE  NB.ú.4.B1.....

```

Once again, **Ctrl+F9** and Follow in dump the new allocated memory:

```

CPU  Graph  Log  Notes  Breakpoints  Memory Map  Call Stack  SEH  Script  Symbols  Source  References  Threads  Handles  Trace  WFD
7617E95C  CT 3000  mov  ecx, eax
7617E95F  80CB  call  C:\BaseSetLastError()
7617E961  EB 6A29EFF  jmp  0x7617E958
7617E966  EB FB  jmp  0x7617E968
7617E968  CC  jmp  0x7617E968
7617E969  CC  jmp  0x7617E968
7617E96A  CC  jmp  0x7617E968
7617E96B  CC  jmp  0x7617E968
7617E96C  CC  jmp  0x7617E968
7617E96D  CC  jmp  0x7617E968
7617E96E  CC  jmp  0x7617E968
7617E96F  CC  jmp  0x7617E968
7617E970  CC  jmp  0x7617E968

```

```

Address Hex ASCII
00C50000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00C50010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00C50020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00C50030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00C50040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....

```

Let's continue execute and hit the third call to **VirtualAlloc**, some bytes were written to the new allocated memory. They do not look like shellcode but could be some data that malicious code uses:

```

EDI  00C50000  23FC    and   edi, esp
      00C50002  3806    cmp   byte ptr ds:[esi], al
      00C50004  34 01    xor   al, 0x1
      00C50006  0107    add   dword ptr ds:[edi], eax
      00C50008  EF      out   dx, eax
      00C50009  0067 05    add   byte ptr ds:[edi+0x5], ah
      00C5000C  52      push  edx
      00C5000D  63A7 06113271  arpl  word ptr ds:[edi+0x71321106], sp
      00C50013  0100    add   dword ptr ds:[eax], eax
      00C50015  0000    add   byte ptr ds:[eax], al
      00C50017  0065 14    add   byte ptr ss:[ebp+0x14], ah
      00C5001A  6990 00000000 0000  imul  edx, dword ptr ds:[eax], 0x0
      00C50024  0000    add   byte ptr ds:[eax], al
      00C50026  0000    add   byte ptr ds:[eax], al
      00C50028  0000    add   byte ptr ds:[eax], al

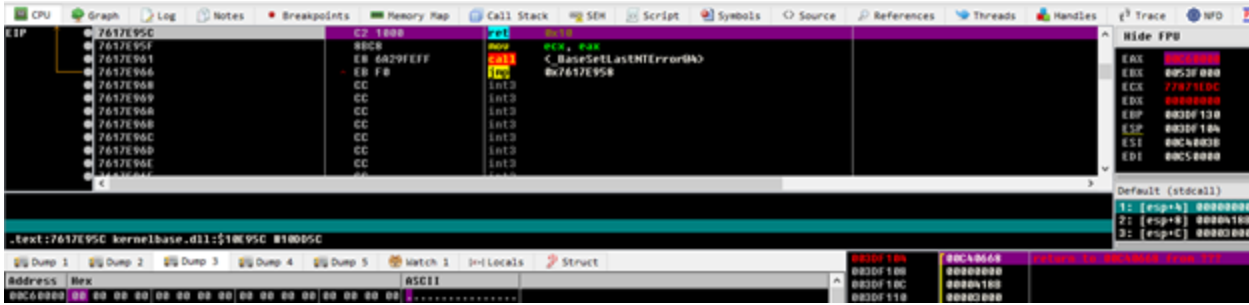
```

```

edi=00C50000
esp=003DF104
00C50000
Dump 1  Dump 2  Dump 3  Dump 4  Dump 5  Watch 1  [x] Locals  Struct
Address Hex ASCII
00C50000 23 FC 38 06 34 01 01 07 EF 00 67 05 52 63 A7 06  ü8.4...ÿ.g.Rç$.
00C50010 11 32 71 01 00 00 00 00 65 14 69 90 00 00 00 00  .2q.....e.i.....
00C50020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00C50030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00C50040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....

```

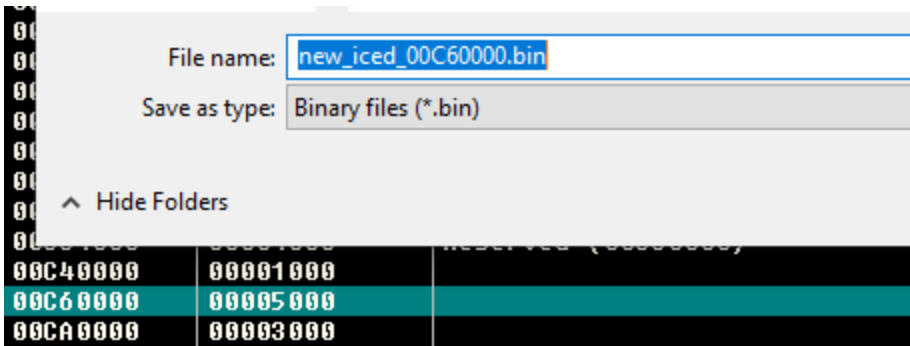
Continuing to execute the call to the **VirtualAlloc** function, we have a newly allocated memory:



Press **F9** , we break at **VirtualProtect** . The newly allocated device has been filled with bytes. I spotted a PE file that has been compressed using *aPlib* because the PE magic bytes **MZ** become **M8Z** .

Address	Hex	ASCII
00C60000	4D 38 5A 90	M8Z .8.f...qj.â.
00C60010	01 40 C2 15	.â.â.â.â.â.â.â.â.
00C60020	21 B8 01 4C	?.LÀ.This .prog
00C60030	67 61 6D 87	gam.cGn.0tçbe iu
00C60040	5F 98 69 06	_.i.D0~S.mode...
00C60050	0A 24 4C 44	.\$LD...0.íúqxx.¾
00C60060	0A 98 B7 D6	..-üâ.¼ `î.+¾üC
00C60070	C8 3C B4 22	È<`"i.Rich(!.PPE
00C60080	80 4C 01 A0	.L. æSt+.]..à...
00C60090	0B 23 0E 0C	.#...v.¾.3=...+
00C600A0	09 20 E6 A0	. æ .@..à.âA.¡ç@
00C600B0	15 88 1F 40	...@.âS,...ú...-
00C600C0	21 49 78 2D	?Ix-é.x.+U..Z.
00C600D0	C1 2E 74 65	â.texî"2.'..N.BC
00C600E0	C0 60 2E 72	â`.rdart.h.@ie..

Follow this section in the **Memory Map** and dump it to file:



3. Decompress dumped file

From the command line, simply need to pass dumped file to *aprip.py* . The tool will do its job and each extracted file will be written to a file “**dump0.bin**”, “**dump1.bin**”, ...

```

Cmder
Downloads

-----

APLIB RIPPER 1.2

-----

Ripping PE files, this may take some time...
- Ripped PE writing to file: dump0.bin

```

Check **dump0.bin (21dd005162c62af26f3f59e2ebcb345c)** with PE-bear:
 AddressOfEntryPoint = 0x0000163D

The screenshot shows the PE-bear disassembler interface. On the left, the 'Sections' pane lists .text (EP = A3D), .rdata, .data, and .reloc. The main window displays assembly code for the .text section:

Address	Hex	Disasm	Hint
163D	E9B7FF	CALL 0x4014F9	
1642	6A00	PUSH 0	
1644	FF16C0204000	CALL DWORD PTR [0x40200C] (KERNEL32.dll).ExitProcess	
164A	CC	INT3	
164B	83EC1C	SUB ESP, 0x1C	

Below the disassembler is a table of sections:

Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.
> .text	400	A00	1000	932	60000020	0	0	0
> .rdata	E00	600	2000	468	40000040	0	0	0
> .data	1400	400	3000	250	C0000040	0	0	0
> .reloc	1800	200	4000	8C	42000040	0	0	0

Valid IATs:

The screenshot shows the 'Valid IATs' view in PE-bear. It displays a table of valid IATs:

Offset	Name	Func. Count	Bound?	OriginalFirstThunk	TimeDateStamp	Forwarder	NameRVA
F0C	ADVAPI32.dll	1	FALSE	2184	0	0	2228
F20	SHELL32.dll	1	FALSE	21D8	0	0	224A
F34	KERNEL32.dll	18	FALSE	218C	0	0	2356
F48	WINHTTP.dll	10	FALSE	21EC	0	0	2438
F5C	USER32.dll	2	FALSE	21E0	0	0	245C

Below this is the 'KERNEL32.dll [18 entries]' table:

Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
2008	IstrcpyA	-	231A	231A	-	62D
200C	ExitProcess	-	2326	2326	-	15C
2010	CreateDirectoryA	-	2334	2334	-	B4
2014	IstrcatA	-	2306	2306	-	624
2018	Sleep	-	2312	2312	-	575
201C	IstrlenA	-	22FA	22FA	-	633
2020	ReadFile	-	2256	2256	-	46C
2024	HeapFree	-	2262	2262	-	345
2028	WriteFile	-	226F	226F	-	60A

End!

m4n0w4r