

- REVERSE ENGINEERING -

Zbot Malware Unpacking Process

MD5 8a0c95be8a40ae5419f7d97bb3e91b2b
SHA-1 3fb703474bc750c5e99da9ad5426128a8936a118
File Type Win32 EXE
Packer ASProtect v1.23 RC1

While start, typically we perform basic static analysis. For example strings, imported functions, sections so on.

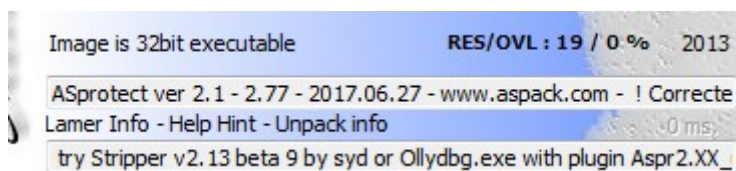
Analysing of strings don't provide human readable anything us. Lets analysis PE sections. I wonder, can we find sections as .text, .data so on? When we analysed sections with CFF Explorer, have some unnamed sections and .data and .rsc sections but we have not found .text section which contains program's codes.

	00004000	00036000	00003000	00022C00	00000000	00000000	0000
	0000B000	0003A000	00002E00	00025C00	00000000	00000000	0000
	0000B000	00045000	00002E00	00028A00	00000000	00000000	0000
	0000B000	00050000	00002E00	0002B800	00000000	00000000	0000
.rsrc	00015000	0005B000	00013000	0002E600	00000000	00000000	0000
	00019000	00070000	00000000	00041600	00000000	00000000	0000
.data	0005B000	00089000	00023E00	00041600	00000000	00000000	0000
.adata	00001000	000E4000	00000000	00065400	00000000	00000000	0000

Also we want to detect imported functions but what the hell? Only one function imported from used every library. This technique is commonly used by packers.

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
000421D1	N/A	00042108	0004210C	00042110	00042114	00042118
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
kernel32.dll	3	00000000	00000000	00000000	00089A50	00089A3C
user32.dll	1	00000000	00000000	00000000	00089B94	00089C1B
user32.dll	1	00000000	00000000	00000000	00089B9F	00089C23
advapi32.dll	1	00000000	00000000	00000000	00089BAA	00089C2B
oleaut32.dll	1	00000000	00000000	00000000	00089BB7	00089C33
advapi32.dll	1	00000000	00000000	00000000	00089BC4	00089C3B
version.dll	1	00000000	00000000	00000000	00089BD1	00089C43
gdi32.dll	1	00000000	00000000	00000000	00089BDD	00089C4B
oleaut32.dll	1	00000000	00000000	00000000	00089BE7	00089C53
comctl32.dll	1	00000000	00000000	00000000	00089BF4	00089C5B

Because this status is a little suspected, we have used "exeinfope" to detect whether use any packer.



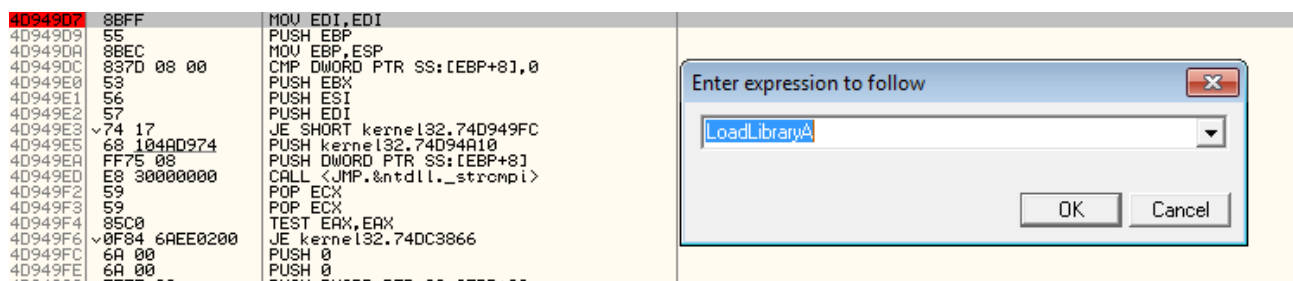
Alright! We have detected which used packer.

Aspack packer has been focusing on security. Therefore Aspack is difficult to unpacking. Also automated unpacking tools may be useless.

Lets explain how to unpacking?

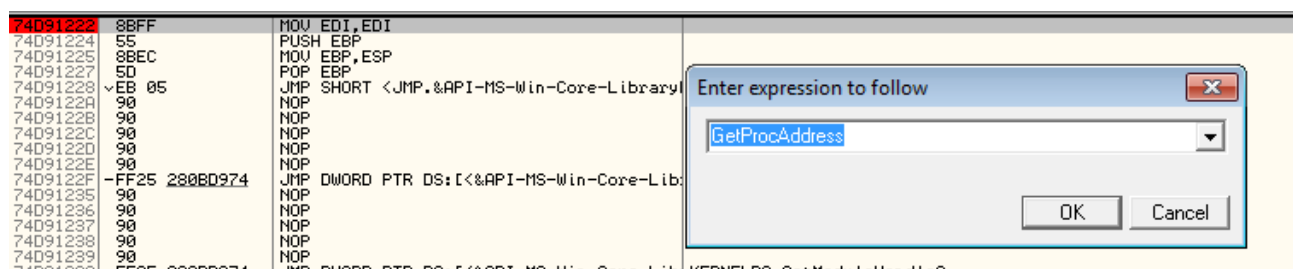
Before "unpacking stub" transfers execution flow to OEP, it loads libraries and functions using two API functions : LoadLibrary and GetProcAddress. Each packed executables includes these API functions certainly. If we find last imported library and function before the execution transfers to "OEP", we are so near malware's main process.

To do this, first in OllyDbg we set breakpoint at both LoadLibraryA and LoadLibraryW functions pressing "Ctrl + G" key combination.



Perform same process for LoadLibraryw and run pressing F9 key until program running. Here so important. That's so important because after each library loaded program will be pause. You need to note last loaded library before program state is running.

Restart the program then you repeat above process and after last library have loaded, in additional set a breakpoint on GetProcAddress and progress until find last imported function. Keep in mind, note last imported function.



Then you can progress step over(F8) on program's code until find possible program start point.

0041BC1E	85	DB 85	
0041BC1F	C0	DB C0	
0041BC20	75	DB 75	CHAR 'u'
0041BC21	02	DB 02	
0041BC22	B3	DB B3	
0041BC23	01FF	ADD EDI,EDI	
0041BC25	75	DB 75	CHAR 'u'
0041BC26	F8	DB F8	
0041BC27	FF	DB FF	
0041BC28	15	DB 15	
0041BC29	84	DB 84	
0041BC2A	12	DB 12	
0041BC2B	40	DB 40	CHAR '@'
0041BC2C	00	DB 00	
0041BC2D	8A	DB 8A	
0041BC2E	C3	DB C3	
0041BC2F	5B	DB 5B	CHAR '['
0041BC30	C9	DB C9	
0041BC31	C2	DB C2	
0041BC32	04	DB 04	
0041BC33	00	DB 00	
0041BC34	55	DB 55	CHAR 'U'
0041BC35	8B	DB 8B	
0041BC36	EC	DB EC	
0041BC37	51	DB 51	CHAR 'Q'
0041BC38	56	DB 56	CHAR 'V'
0041BC39	8B	DB 8B	
0041BC3A	35	DB 35	CHAR 'S'
0041BC3B	30	DB 30	CHAR '0'
0041BC3C	10	DB 10	
0041BC3D	40	DB 40	CHAR '@'
0041BC3E	00	DB 00	

We reached this point and on cursor point press right click and Analysis -> Analyse Code.

0041BC1E	. 85C0	TEST EAX,EAX	
0041BC20	.>75 02	JNZ SHORT zeus.0041BC24	
0041BC22	. B3 01	MOV BL,1	
0041BC24	> FF75 F8	PUSH DWORD PTR SS:[EBP-8]	hMemory
0041BC27	> FF15 84124000	CALL DWORD PTR DS:[401284]	LocalFree
0041BC2D	> 8AC3	MOV AL,BL	
0041BC2F	. 5B	POP EBX	
0041BC30	. C9	LEAVE	
0041BC31	. C2 0400	RETN 4	
0041BC34	. 55	PUSH EBP	
0041BC35	. 8BEC	MOV EBP,ESP	
0041BC37	. 51	PUSH ECX	
0041BC38	. 56	PUSH ESI	
0041BC39	. 8B35 30104000	MOV ESI,DWORD PTR DS:[401030]	ADVAPI32.GetTokenInformation
0041BC3F	. 57	PUSH EDI	
0041BC40	. 8045 FC	LEA EAX,DWORD PTR SS:[EBP-4]	
0041BC43	. 50	PUSH EAX	
0041BC44	. 6A 00	PUSH 0	pRetLen
0041BC46	. 6A 00	PUSH 0	BufSize = 0
0041BC48	. 6A 01	PUSH 1	Buffer = NULL
0041BC4A	. FF75 08	PUSH DWORD PTR SS:[EBP+8]	InfoClass = TokenUser
0041BC4D	. FFD6	CALL ESI	hToken
0041BC4F	. 85C0	TEST EAX,EAX	GetTokenInformation
0041BC51	.>75 36	JNZ SHORT zeus.0041BC89	
0041BC53	. FF15 40114000	CALL DWORD PTR DS:[401140]	GetLastError
0041BC59	. 83F8 7A	CMP EAX,7A	
0041BC5C	.>75 2B	JNZ SHORT zeus.0041BC89	
0041BC5E	. 8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	
0041BC61	. E8 CAC4FFFF	CALL zeus.00418130	

Because 41BC1E address do not present program's start point we progress with step by step.

004088DB	. 55	PUSH EBP	
004088DC	. 8BEC	MOV EBP,ESP	
004088DE	. 83EC 0C	SUB ESP,0C	
004088E1	. 53	PUSH EBX	
004088E2	. 56	PUSH ESI	
004088E3	. 33F6	XOR ESI,ESI	
004088E5	. 56	PUSH ESI	
004088E6	. 32DB	XOR BL,BL	Arg1 => 00000000
004088E8	. E8 B6F2FFFF	CALL zeus.00407BA3	zeus.00407BA3
004088ED	. 84C0	TEST AL,AL	
004088EF	.>0F84 B6000000	JE zeus.004089AB	
004088F5	. 68 07800000	PUSH 8007	ErrorCode = SEM_FAILCRITICALERRORS;SEM_NOOPFAULTEORR
004088FA	. C645 F4 00	MOV BYTE PTR SS:[EBP-C],0	
004088FE	. C645 F4 01	MOV BYTE PTR SS:[EBP-8],1	
00408902	. FF15 D4124000	CALL DWORD PTR DS:[4012D4]	SetErrorMode
00408908	. 8D45 FC	LEA EAX,DWORD PTR SS:[EBP-4]	
0040890B	. 50	PUSH EAX	
0040890C	. FF15 D8124000	CALL DWORD PTR DS:[4012D8]	pArgv
00408912	. 50	PUSH EAX	GetCommandLineW
00408913	. FF15 18134000	CALL DWORD PTR DS:[401318]	CmdLine
00408919	. 3BC6	CMP EAX,ESI	CommandLineToArgvW
0040891B	.>74 68	JE SHORT zeus.00408985	
0040891D	. 33D2	XOR EDX,EDX	
0040891F	. 3975 FC	CMP DWORD PTR SS:[EBP-4],ESI	

At last, we found possibly start point because executables commonly starts with GetVersionEx or GetCommandLine functions. Now we have to repair import table with Scylla or ImpRec.

We have found OEP as 4088DB, repair import table and reconstruct with Scylla.

Imports

+	✓	advapi32.dll (31) FThunk: 00001000
+	✓	crypt32.dll (8) FThunk: 00001080
+	✓	gdi32.dll (13) FThunk: 000010A4
+	✓	kernel32.dll (129) FThunk: 000010DC
+	✓	netapi32.dll (3) FThunk: 000012E4
+	✓	oleaut32.dll (4) FThunk: 000012F4
+	✓	psapi.dll (2) FThunk: 00001308
+	✓	shell32.dll (3) FThunk: 00001314
+	✓	shlwapi.dll (22) FThunk: 00001324
+	✓	sspicli.dll (1) FThunk: 00001380
+	✓	user32.dll (101) FThunk: 00001388
+	✓	version.dll (3) FThunk: 00001520

IAT Info

OEP:

VA:

Size:

Actions

Dump

Log

File: zeus-unpacked.exe

Entry Point: oo < EP Section: .text

File Offset: First Bytes: 55.8B.EC.83.EC

Linker Info: SubSystem: Windows GUI

File Size: < u Overlay: NO 00000000

Image is 32bit executable RES/OVL: 0 / 0 % 2013

Microsoft Visual C++ ~v.7.10 - 10 - Visual 2010

Lamer Info - Help Hint - Unpack info

Address	Ordinal	Name	Library
00000000...	21	setsockopt	WS2_32
00000000...	2	bind	WS2_32
00000000...		freeaddrinfo	WS2_32
00000000...		WSAEventSelect	WS2_32
00000000...	6	getsockname	WS2_32
00000000...	1	accept	WS2_32
00000000...	3	closesocket	WS2_32
00000000...	112	WSASetLastError	WS2_32
00000000...	23	socket	WS2_32
00000000...	19	send	WS2_32
00000000...		CLSIDFromString	ole32
00000000...		CoUninitialize	ole32
00000000...		CoCreateInstance	ole32
00000000...		CoSetProxyBlanket	ole32
00000000...		CoInitializeEx	ole32

Line 372 of 372

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	129	0002F2D0	00000000	00000000	00030152	000010DC
USER32.dll	101	0002F57C	00000000	00000000	00030816	00001388
ADVAPI32.dll	31	0002F1F4	00000000	00000000	00030ADC	00001000
SHLWAPI.dll	22	0002F518	00000000	00000000	00030C68	00001324
SHELL32.dll	3	0002F508	00000000	00000000	00030CAE	00001314
Secur32.dll	1	0002F574	00000000	00000000	00030CCC	00001380
PSAPI.DLL	2	0002F4FC	00000000	00000000	00030D0C	00001308
ole32.dll	7	0002F7E0	00000000	00000000	00030D94	000015EC
GDI32.dll	13	0002F298	00000000	00000000	00030E6E	000010A4
WS2_32.dll	24	0002F77C	00000000	00000000	00030ED4	00001588
CRYPT32.dll	8	0002F274	00000000	00000000	00030FB0	00001080
WININET.dll	21	0002F724	00000000	00000000	00031192	00001530
OLEAUT32.dll	4	0002F4E8	00000000	00000000	0003119E	000012F4
NETAPI32.dll	3	0002F4D8	00000000	00000000	000311E0	000012E4