Unpacking malicious software using IDA Pro extensions

A paper by Dennis Elser

In almost all cases of today's malicious software, executable packers or -crypters are used in order to obfuscate code and data. In some cases unpackers and dumpers are available. In very few cases they actually work on packed malware executables due to modifications of internal structures such as the PE header.

In the following example an unknown binary is loaded into IDA Pro¹. The code at the entry point of the executable looks like this:

UPX1:01015360 UPX1:01015360 UPX1:01015360 start * UPX1:01015360	public start proc near pusha
• UPX1:01015361 • UPX1:01015366	mov esi, offset dword_1011000 lea edi, [esi-10000h]
• UPX1:0101536D • UPX1:0101536D • UPX1:01015370	or ebp, OFFFFFFFh jmp short loc_1015382
heve everyone	Fig. 1

A segment named "UPX1", an invalid import address table and an empty list of strings are an indicator for a packed file. UPX^2 however, can not unpack the file because internal structures have been modified. This technique often is used by malware authors to make unpacking and reverse engineering harder.

The first step now is to obtain a readable representation of the packed executable. A good and quick start in achieving this is to run the executable and dump the previously packed segment(s), once they have been unpacked. Preferably, the dump should be made right after the executable has been completely unpacked in memory. This often is the case after the original entry point (OEP) has been reached. Finding the OEP isn't always trivial and can be a time consuming process because you need to single step through the code. Using the IDA Pro SDK, a plugin named EPF³ (Entry Point Finder) has been created, aimed towards automating the process of finding the original entry point.

An isolated environment (a virtual machine for example) is used to carefully

	UP×1:01015360	public	c start
-	UPX1:01015360	start	proc near
EIP •	UPX1:01015360	pusha	
	UPX1:01015361	mov	esi, offset dword_1011000
	UPX1:01015366	lea	edi, [esi-10000h]
	UPX1:0101536C	push	edi
	UPX1:0101536D	or	ebp, OFFFFFFFh
	UPX1:01015370	jmp	short loc_1015382

Fig. 2

run the executable in IDA Pro's debugger. Figure 2 shows the extended instruction

pointer (EIP) pointing to a "pusha" mnemonic. This statement is used as the first instruction to "back up" the content of all standard registers. Many executable compressors use a "popa" instruction at the end of their code to restore the previously saved state. This behavior can be exploited by the EPF plugin; the plugin offers an option to let the IDA Pro debugger trace code until a specific mnemonic is reached.

C Trac	e until EIP reaches a different	section
C Trac	e until EIP reaches a specific	memory area
C Trac	e until EIP reaches a specific e until a register bolds a speci	mnemonic fic value
C Trac	e until anv register h ^{Please er}	nter a mnemonic b
- ndo		
nemonic	popa	<u> </u>
egister	eax	•
alue	0xDEADBEEF	-
	2 C 2 C 2 C 2 C 2 C 2 C 2 C 2 C 2 C 2 C	and the second se

Fig. 3

After the EPF plugin has been started and configured, the process can be resumed (be careful, don't run malware on your host system!). After a few seconds, the process is paused and EPF turned off. The following message appears:

-> EPF is -> EPF: Pl	istopatmne nowon ease resume t	monic popa he process_n	ow!
-> EPF: Mn -> EPF is	now off	at 010154AE.	
Arthur	Down Dick: 10	B UNKNOWN	0004

Fig. 4

The code at EIP points to a "popa" mnemonic followed by a jump and the end of

1. J	UPX1:010154A8	loc_10154A8:
4.	UPX1:010154A8	call dword ptr [esi+1BEE4h]
	UPX1:010154AE	
100	UPX1:010154AE	loc_10154AE:
EIP.	UPX1:010154AE	popa
	UPX1:010154AF	jmp near ptr dword_100739D
	UPX1:010154AF	start endp

Fig. 5

the "start" procedure. Single stepping over the jump leads to the following message box:





Fig. 7

Choosing "Yes" creates an instruction at EIP and IDA Pro begins to analyze control flow.

	UPX0:0100739D	loc_1007	739D:	
÷	UPX0:0100739D	push	70h	
	UPX0:0100739F	push	10018	98h
.*	UPX0:010073A4	call	sub_1	007568
	UPX0:010073A9	xor	ebx.	ebx
. •	UPX0:010073AB	push	ebx	
٠	UPX0:010073AC	mov	edi.	dword_1001000+0CCh

EIP now points to a segment labeled "UPX0". This is very likely the original entry point. It is reasonable to make a dump of the segment now. The DumpSeg⁴ plugin can list and dump all segments available.

Idebug034 0088000 0088800 Idebug035 0098000 0098200 Idebug036 009000 0090200 Idebug037 009000 0090300 Imalware.exe 010000 0101100 IUPX1 0101100 010100 IUPX1 010100 010100 IUPX1 010100 010100 Idebug038 010100 010100 Idebug039 010100 010100 Idebug039 010100 010100 Idebug040 6650100 6650100 Idebug041 6650500 6650600 Idebug043 6650800 6650400 Imalware.exe 6660100 6660100 Imalware.exe 6660200 6660300 Imalware.exe 6660200 6660300 Imalware.exe 6660200 6660300	e of segment	Start address	End address		
Idebug035 0098000 0098200 Idebug036 009000 0090200 Idebug037 009000 0090300 Imalware.exe 010000 010100 IUPX1 010100 010100 Idebug038 010100 010100 Idebug039 010100 010100 Idebug040 6650100 6650500 Idebug041 6650800 6650800 III debug042 6650800 6660100 III debug044 6660100 6660200 III debug045 6660200 6660300 III debug045<	ebug034	008B000	008B800		
If debug036 009C000 009C200 If debug037 009D000 009D300 If malware.exe 0100000 0101100 IVFX0 0100100 0101100 IVFX1 0101100 010100 IVFX1 010100 010100 IVFX1 00000 6650100 Idebug038 010100 0101E00 If debug040 6650100 6650500 If debug041 6650500 6650800 If debug042 6650800 6650400 If debug044 6660100 6660200 If debug045 6660200 6660300 If debug045 6660200 6660300	ebug035	009B000	009B200		
8ff debug037 009D000 009D300 8ff debug037 0100000 0100100 8ff debug005 010100 010100 8ff debug005 0101600 010100 8ff debug038 010100 010100 8ff debug039 010100 010100 8ff debug039 010100 010100 8ff debug039 010100 010100 8ff debug040 6650100 6650100 8ff debug041 6650500 6650600 8ff debug042 6650600 6650800 8ff debug043 6650800 6650400 8ff debug044 6660100 6660200 8ff debug045 6660200 6660300 8ff debug045 6660200 6660300	ebug036	0090000	009C200		
Simalware.exe 0100000 0100100 Simalware.exe 0100100 0101100 Simulary 0100100 0101100 Simulary 0101100 0101600 Simulary 010100 010100 Simulary 010000 00	ebug037	009D000	009D 300		bua
UFX0 0100100 0101100 WI PX1 0101100 0101600 Dump segment to disk Ctrl Webug005 0101600 0101000 Refresh Ctrl Webug038 0101000 0101000 Copy Ctrl Webug039 0101000 6650100 Copy Ctrl Webug040 6650100 6650500 Copy Ctrl Webug041 6650500 6650600 Copy Ctrl Webug043 6650800 6650800 Ctrl Ctrl Webug043 6650800 6650400 Ctrl Ctrl Webug043 6650800 6650400 Ctrl Ctrl Webug044 6660100 6660100 Ctrl Ctrl Webug045 6660200 6660300 Ctrl Ctrl Webug045 6660200 6660300 Ctrl Ctrl	alware.exe	0100000	0100100		1
Bit UPX1 0101100 0101600 Dump segment to disk. Ctu Bit debug005 0101600 0101000 Refresh Ctu Bit debug038 0101C00 0101D00 Copy Ctul Bit debug039 0101D00 0101E00 Copy Ctul Bit debug040 6650100 6650500 Copy Ctul Bit debug041 6650500 6650600 Copy Ctul Bit debug042 6650600 6650800 Ctul Copy Ctul Bit debug043 6650800 6650400 Ctul Copy Ctul Ctul Bit debug044 6660100 6660200 Ctul	IPX0	0100100	0101100 💻		
Siff debug005 0101600 0101000 Refresh Ctr Siff debug038 0101000 0101000 Copy Ctrl4 Siff debug039 0101000 0101E00 Copy Ctrl4 Siff debug040 6650100 6650500 Copy Ctrl4 Siff debug041 6650500 6650600 Copy Ctrl4 Siff debug042 6650600 6650800 Ctrl4 Ctrl4 Siff debug043 6650800 6650400 Ctrl4 Ctrl4 Ctrl4 Siff debug043 6650800 6660100 Ctrl4 Ctrl4 Ctrl4 Siff debug044 6660100 6660200 Ctrl4 Ctrl4 Ctrl4 Siff debug045 6660200 6660300 Ctrl4 Ctrl4 Ctrl4	IPX1	0101100	0101600	Dump segment to disk	Ctrl+E
If debug038 0101C00 0101D00 Copy Ctrl+ If debug039 0101D00 0101E00 Copy Ctrl+ If debug040 6650100 6650500 6650600 Image: Copy Ctrl+ If debug041 6650500 6650600 6650800 Image: Copy Ctrl+ If debug042 6650600 6650800 6650800 Image: Copy Ctrl+ If debug043 6650800 6650800 6650800 Image: Copy Ctrl+ If debug043 6650800 6650800 6660100 Image: Copy Ctrl+ If debug044 6660100 6660200 6660200 Image: Copy Ctrl+ If debug045 6660200 6660300 Image: Copy Ctrl+ Image: Copy Ctrl+ If debug045 6660200 6660300 Image: Copy Ctrl+ Image: Copy Ctrl+	ebug005	0101600	0101C00	Refresh	Ctrl+U
Sff debug039 0101D00 0101E00 Copy Ctrl Sff debug039 0101D00 6650100 6650100 6650100 Sff debug040 6650100 6650500 6650600 6650800 6660100 6660100 6660200 6660200 6660300 6660300 6660300 6660300 6660400 6660400 6660400 6660400 6660400 6660400 6660400 6660400 6660400 6660400 6660400 6660400 6660400 6660400 6660400 6660400 6660400 6660400 660400 660400 660400 660400 660400 660400 660400 660400 660400 660400 660400 660400 660400 660400 660400 660400 660400	ebug038	0101C00	0101D00 -		see to exclude
Sff wbsys.dll 6650000 6650100 Sff debug040 6650100 6650500 Sff debug041 6650500 6650600 Sff debug042 6650600 6650800 Sff debug043 6650800 6650400 Sff debug043 6650800 6650400 Sff debug044 6660100 6660100 Sff debug044 6660100 6660200 Sff debug045 6660200 6660300 Sff debug045 6660300 6660400	ebug039	0101D00	0101E00 🔔	Сору	Ctrl+Ins
Sff debug040 6650100 6650500 Sff debug041 6650500 6650600 Sff debug042 6650600 6650800 Sff debug043 6650800 6650400 Sff debug043 6650800 6650400 Sff debug043 666000 6660100 Sff debug044 6660100 6660200 Sff debug045 6660200 6660300 Sff debug045 6660300 6660400	vbsys.dll	6650000	6650100		ce
Sff debug041 6650500 6650600 Sff debug042 6650600 6650800 Sff debug043 6650800 6650400 Sff debug044 666000 6660100 Sff debug044 6660100 6660200 Sff debug045 6660200 6660300 Sff debug045 6660200 6660400	ebug040	6650100	6650500		
Sff debug042 6650600 6650800 Sff debug043 6650800 6650400 Sff debug043 6660000 6660100 Sff debug044 6660100 6660200 Sff debug045 6660200 6660300 Sff debug045 6660200 6660300 Sff debug045 6660300 6660400	ebug041	6650500	6650600		
Sff debug043 6650800 6650400 Sff wbhelp.dll 6660000 6660100 Sff debug044 6660100 6660200 Sff debug045 6660200 6660300 Sff debug045 6660200 6660400	ebug042	6650600	6650800		
Sff wbhelp.dll 6660000 6660100 Sff debug044 6660100 6660200 Sff debug045 6660200 6660300 Sff debug045 6660200 6660400	ebug043	6650800	6650A00		in an alter
8ff debug044 6660100 6660200 8ff debug045 6660200 6660300 8ff debug046 6660300 6660400	bhelp.dll	6660000	6660100		art
8ff debug045 6660200 6660300 8ff debug046 6660300 6660400	ebug044	6660100	6660200		375
% debua046 6660300 6660400	ebug045	6660200	6660300		0
Un second s	ebug046	6660300	6660400		20
%# debug047 6660400 6660700	ebug047	6660400	6660700		CD
8ff debug048 6660700 6660800 🐙	ebug048	6660700	6660800		+1
047 6660400 6660700	9043 Ip.dll 9044 9045 9046 9047	6650800 6660000 6660100 6660200 6660300 6660400	6650A00 6660100 6660200 6660300 6660400 6660400		art

The dumped segment can then be reloaded and analyzed by IDA Pro.

References

- 1.) IDA Pro, DateRescue (http://www.datarescue.com/idabase)
- 2.) UPX, Markus F.X.J. Oberhumer & Laszlo Molnar
- 3.) EPF, Dennis Elser (<u>http://www.backtrace.de</u>)
- 4.) DumpSeg, Dennis Elser (http://www.backtrace.de)