Unpacking Malware Manually

blog.cyber5w.com/the-most-known-unpacking-technique

Objectives

In this blog post, we will go through a famous packing technique which is the use of VirualAlloc and VirtualProtect to decrypt data in memory and execute it, and how to unpack it manually, we are going to apply it to **Death Ransomware** malware

Introduction

What is packed malware?

packed malware refers to malicious software that has been compressed and/or encrypted to obfuscate its code and make it more difficult to detect by antivirus or other security solutions.

Static Analysis

Let's open the sample in DIE to see if it is packed or not

DIE is a tool that detects if the malware is packed or not. It does this by measuring the entropy of the file, which is a measure of randomness. If the data in a file is more random, it usually means that the file is packed.

When the entropy of a file is greater than 7, it generally indicates that the file is likely compressed or encrypted.



Yeah It's packed

Let's see its imports in IDA

1	43				TICK VICH I	ou actur co	LATE .	Linuma	 42
1	Add	dress	Ordinal	Name	Library				
	1	00424000		GetSystemTimes	KERNEL32				
		00424004		GetFirmwareEnvironmentVariableA	KERNEL32				
	1	00424008		lstrlenA	KERNEL32				
		0042400C		LocalAlloc	KERNEL32				
	1	00424010		OpenJobObjectW	KERNEL32				
	M	00424014		GetCurrentDirectoryA	KERNEL32				
		00424018		OpenSemaphoreA	KERNEL32				
	1	0042401C		IsProcessInJob	KERNEL32				
	1	00424020		GetModuleHandleA	KERNEL32				
		00424024		GetUserDefaultLangID	KERNEL32				
		00424028		GetMailslotInfo	KERNEL32				
	1	0042402C		HeapReAlloc	KERNEL32				
	1	00424030		LoadLibraryW	KERNEL32				
		00424034		пеарліюс	KERNEL32				
	1	00424038		GetProcAddress	KERNEL32				
		0042403C		CreateMailslotA	KERNEL32				
	2	00424040		FreeEnvironmentStringsA	KERNEL32				
		00424044		IstrcmpW	KERNEL32				
	1	00424048		GetFileTime	KERNEL32				
	1	0042404C		GetStdHandle	KERNEL32				
	21	00424050		CreateDirectoryExA	KERNEL32				
	1	00424054		GetFileAttributesExA	KERNEL32				
		00424058		ReadConsoleInputA	KERNEL32				
		0042405C		GetModuleFileNameA	KERNEL32				
1		00424060		MultiByteToWideChar	KERNEL32				
		00424064		WideCharToMultiByte	KERNEL32				
_		00424068		GetStringTypeW	KERNEL32				
		0042406C		EnterCriticalSection	KERNEL32				
		00424070		LeaveCriticalSection	KERNEL32				
		00424074		DeleteCriticalSection	KERNEL32				
		00424078		EncodePointer	KERNEL32				
		0042407C		DecodePointer	KERNEL32				
		00424080		SetLastError	KERNEL32				
		00424084		InitializeCriticalSectionAndSpinCount	KERNEL32				
	*								

Virtual Alloc, **Virtual protect** are not listed, but I think that the malware resolves them dynamically

As we can see the sample resolves Virtual protect

VirtualAlloc and **VirtualProtect** are two Windows API functions commonly used by the malware to unpack itself.

Malware uses **VirtualAlloc** to allocate memory for the unpacked malware code then uses **VirtualProtect** to change the protection to mark the memory allocated as executable, writable, or both to be able to execute the dynamically unpacked code.



Let's open our sample into x64dbg

I'll put a breakpoint in VirtualAlloc

Press ctrl+g and write in the search bar "VirtualAlloc" and click ok.

To put a breakpoint in **VirtualAlloc** we need to click on the circle on the left side of the **VirtualAlloc** instruction

Let's run the sample until we hit the breakpoint



Let's go to the return of the function and step over it and follow **EAX** In a dump.

After stepping over some code there is some data written into the dump

E1P 00320180 8840 F4 00320183 8908 00320183 FF65 FC 00320188 F65 FC 00320188 8908 00320188 S08 00320188 8845 0C 00326110 8885 0C 00526102 53 00526102 53 00526102 50 00526102 8845 08 00526102 8945 08 ecx=FFFFFFFC	<pre>mov dev.dword ptr ss:[ebp-4] mov dword ptr ss:[ebp-4] leave ret ebp mov ebpesp mov eax.dword ptr ss:[ebp+6] nusiheby push esi nov esi,dword ptr ss:[ebp+8] add eax.esi mov day.dword ptr ss:[ebp+14]</pre>	[ebp+14]:"VirtualProtect"	01 00 51 00 00 00 CF 0 TF 0 IF 1 LastError 00000006 (ERROR_INVALID_HANDI LastStatus CO000008 (STATUS_INVALID_HANDI CS 0028 FS 0053 ES 0028 DS 0028 Default (mitral) Default (mitral) 12 (Esp+4) 00780000 00780000 32 (Esp+2) 0017870 0019870
dword ptr ss:[ebp-0C]=[0019F928]=E650 00526180			4: [esp+10] 005264C3 005264C3 5: [esp+14] 0019FB80 0019FB80
Image Dump 1 Image Dump 2 Image Dump 3 Image Dump 4 Image Dump 3 Image Dump 4 Image Dump 3 Image Dump 4 Image Dump 3 Image Dump 3 Image Dump 4 Image Dump 3 Im	Watch 1 Locals 2 Shutt ASCII (45) (4, A, U, 1, 1,, CE) (45) (4, A, U, 1, 1,, CE) FF (5, C, C, L) (4, C, L) (4, C, L) (4, C, L) OD yyp, EOP E. Peil, (5, C, C, L) (4, C, L) (4, L)	0019928 0000650 0019920 00780000 0019930 0019600 0019938 0005463 return 0019938 0052643 return 0019940 0045048 ab8281 0019940 0045048 ab8281 0019940 002528 ab8281 0019940 000528 ab8281 0019940 0000058 ab8281 0019940 000058 ab8281 0019940 ab8281 000000000000000000000000000000000000	1 to 005264C3 from 0052612D F0e0555f88e3005387cb523f221a1933bbd7db4f05902a1 F0e0555f88e3005387cb523f221a1933bbd7db4f05902a1
Command: Commands are comma separated (like assembly in			Default 👻
Paused Dump: 00780000 -> 00780000 (0x00000001 bytes)			Time Wasted Debugging: 0:00:04:2

Let's run the debugger to hit the second **VirtualAlloc** function and do the same thing we did above.

After some stepping over we can see a loop. I'll put a breakpoint at the end of it.

007B0281 007B0286	E8 E30/0000 83C4 14	add esp,14	
007B0289	✓ EB 43	jmp 7B02CE	
007B028B	83A5 48FFFFFF 00	and dword ptr ss:[ebp-B8],0	
00780292	EB OD	imp 7B02A1	
Image: 007B0294		mov eax.dword ptr ss:[ebp-B8]	
007B029A	40	inc eax	
00780298	8985 48FFFFFF	mov dword ntr ss:[ebn-B8] eax	
$\longrightarrow 007B0241$	8885 58FFFFFF	mov eax dword ntr ss:[ebn-48]	
007B0247	888D 48FFFFFF	mov ecv dword ptr ss:[ebp-R8]	
00780240	3848 02	cmp ecv dword ptr 33.[cop b0]	
00780230	73 10	ino 7802CE	
00780230		may any dward at a collabor 10]	
00780252		add eav dward ath ssilebp P2]	
		add eax, dword ptr ss:[ebp-bo]	
00780238		mov ecx, dword ptr ss:[ebp-A8]	
00/802_1	038D 48FFFFFF	add ecx, dword ptr ss:[ebp-B8]	
00/B02C/	8A49 3A	mov cl,byte ptr ds:[ecx+3A]	
00/B02CA	8808	mov byte ptr ds:[eax],cl	
007B02CC	▲LEB C6	jmp 7B0294	
	8D45 E0	lea_eax,dword ptr ss:[ebp-20]	
007B02D1	50	push eax	
007B02D2	6A 40	push 40	
007B02D4	8B85 58FFFFFF	mov eax, dword ptr ss:[ebp-A8]	
© 007B02DA	FF70 0A	push dword ptr ds:[eax+A]	
007B02DD	FFB5 50FFFFFF	push dword ptr ss:[ebp-B0]	
007B02E3	FF55 D8	call dword ptr ss:[ebp-28]	[ebp-28]:Virtua]Protect

Let's run the malware

007B02CE	
💷 Dump 1 💷 Dump 2 🕮 Dump 3 🕮 Dump 4 🕮 Dump 5 🙀 Watch 1 Locals 2 Struct	0019F904
	0019F908
Address Hex ASCII	▲ 0019F90C
00900000 4D 5A 90 00 00 00 00 00 00 00 00 00 00 00 00	0019F910
	0019F914
00900020 00 00 00 00 00 00 00 00 00 00 00 0	0019F918
00900030 00 00 00 00 00 00 00 00 00 00 00 0	0019F91C
00900040 00 00 00 00 00 00 00 00 00 00 00 0	0019F920
00900050 00 00 00 00 00 00 00 00 00 00 00 0	0019F924
	0019F928
00900070 00 00 00 00 00 00 00 00 00 00 00 0	0019F92C
00900080 00 00 00 00 00 00 00 00 00 00 00 0	0019F930
	0019F934
	00105038
	▼ 4
Commands Formands are comma senarated (like assembly instructions). moy eav eby	
command, commands are comma separated (fixe assembly instructions). Mov eax, esx	

A **PE** file is being written in the dump.

This is the final result

00/B02CE									
🛄 Dump 1	🛄 Dump 2	🛄 Dump 3	🛄 Dump 4	🋄 Dump 5 🛛 🍪 Wat	ch 1 🔹 Locals 🤰	Struct			
Address	Hex				ASCII				
00900000	4D 5A 90 0	0 03 00 00	00 04 00 0	0 00 FF FF 00 00) MZÿÿ				
00900010	B8 00 00 0	0 00 00 00	00 40 00 0	0 00 00 00 00 00)@				
00900020	00 00 00 0	0 00 00 00	00 00 00 0	000000000000000000000000000000000000	2				
00900030	00 00 00 0	0 00 00 00	00 00 00 0	0 00 08 00 00 00	0 · · · · · · · · · · · · Ø · · ·				
00900040	OE 1F BA O	E 00 B4 09	CD 21 B8 0	01 4C CD 21 54 68	3 ºI!,.LI!Th				
00900050	69 /3 20 /	0 /2 6F 6/	/2 61 6D 2	0 63 61 6E 6E 6F	is program canno				
00900060	74 20 62 6	5 20 72 75	6E 20 69 6	DE 20 44 4F 53 20	t be run in DOS				
00900070	6D 6F 64 6	2 ZE UD UD	0A 24 00 0		/ mode				
00900080	AT OF DD O	4 A8 70 38	97 A6 70 3		1.VA $30. $ $30. $ $30.$				
00900090	2R 61 26 0		97 40 02 5 97 A1 05 A	R 07 RR 70 30 97	[] 0.0000000000000000000000000000000				
00900040	AR 70 30 0	7 00 70 28	07 2A 1/ 2	1 06 A2 70 28 07	$\begin{bmatrix} +a0. & 50. \\ 0 & 12 \\ 0 & $		T		
Command: Co:									

Let's follow this in the memory map and dump it into a file

⇒ > = →		¬ ✔ ☴ ∅ ∅ fx # A₁ ℝ, ≣ ♀		
🖾 CPU 🍃 Log	Binary	ry Map 🗐 Call Stack 🗣 SEH 🧕 Script 🎽 Symbols 🔇 Source 🏓 Refere	nces 🛸 Threads 뤔 Handles 👸 Trace	
	Copy	FF 00 and dword ptr ss:[ebp-B8],0		
l		FFF mov eax, dword ptr ss: [ebp-B8]		<u>AX</u> 007
	Follow DWORD In Disassembler	FF mov dword ptr ss:[ebp_B8],eax		BX 000 CX 000
	Follow DWORD in Current Dump	FF mov ecx, dword ptr ss:[ebp=A8] FF mov ecx, dword ptr ss:[ebp=B8]		DX 000 BP 001
	Follow DWOKD In Dump	cmp ecx,dword ptr ds:[eax+2] jae 7B02CE		SP 001
	Label Current Address	FFF add eax,dword ptr ss:[ebp-10]	E	DI 000
	Watch DWORD	FF mov ecx,dword ptr ss:[ebp-A8] FFF add ecx,dword ptr ss:[ebp-B8]	E	
	Modify Value Space	mov cl,byte ptr ds:[ecx+3A] mov byte ptr ds:[eax].cl	EF	FLAGS
ETP	Breakpoint	jmp 780294 Jea eav dword ptr ss:[ebp-20]		F 1 PF F 0 SF
C.1.1	Find Pattern Ctrl+B	push eax	ci	F Ö TF
	The Find References Ctrl+R	FF mov eax, dword ptr ss:[ebp-A8]	La	astError
	Sync with expression S	FFF push dword ptr ss:[ebp-B0]	intur ID-start	astStatu
	Allocate Memory	mov dword ptr ss:[ebp-28] [ebp-28]:v1	Generation (Generation)	S 002B S 002B
	🛋 Go to	FFF mov dword ptr ss:[ebp-B0] FFF mov dword ptr ss:[ebp-98],eax		6 0000
	Hex	FF mov eax,dword ptr ss:[ebp-A8] push dword ptr ds:[eax+A]	Def	fault (stdcall)
	A2 Text	FFF push dword ptr ss:[ebp-B0]	▼ 1: ▼ 2:	[esp+4] [esp+8]
eax=007B0E16 dword ptr ss:[e	Integer		3: 4:	[esp+C]
007B02CF	💧 Float			[esp+14
🕮 Dump 1 🕮 Du	un 📑 Address	Dump 5 😽 Watch 1 📖 Locals 💋 Struct	0019F904 0019FB80	
Address Hex	Disassembly	ASCII	0019F908 00000000 ▲ 0019F90C 0019F934	
00900000 4D 5A 00900010 B8 00	<u>00 00 00 00 00 00 40 00 00</u>	∂ FF FF 00 00 MZÿÿ 0 00 00 00 00	0019F910 76474F20 kernel32.ExitPi 0019F914 00900000	rocess
00900020 00 00 00900030 00 00	00 00<	0 00 00 00 00ø	0019F918 00008088 0019F91C 007B0E16	
00900040 OE 1F	BA OE 00 B4 09 CD 21 B8 01 20 70 72 65 67 72 61 6D 20	C CD 21 54 68	0019F920 0019F928 0019F924 0019F934	
00900060 74 20	62 65 20 72 75 6E 20 69 6E	0 44 4F 53 20 t be run in DOS	0019F928 0000E650	
00900070 6D 6F 00900080 EC 1C	56 C4 A8 7D 38 97 A8 7D 38	7 A8 7D 38 97 1.VA 8. 8. 8. 8.	0019F920 00780000	
00900090 A1 05 009000A0 28 61	BB 97 A9 7D 38 97 40 62 3C 36 97 AA 7D 38 97 A1 05 AB	/ AA /D 38 9/ j.».0}8.@b<.°}8. 7 <u>BR 7D 38 97 +a6 *38 : « »38 </u>	0019F934 0019FB70 return to 00526	GAC3 fro
004A0000 (00001000			
00460000		u in Diazaramhlar		
004D0000 (00001000 🚺 Us 🦷 🖓	w in Disassendier	PRV ER	
004E0000	00001000 👗 Us 🖬 Fol	w in Dump	PRV ER	
004F0000 0		a Memory to Sila		
00510000				W
00530000 0	000E0000 🚺 Us 🖛 👓	nent ;	PRV	
00610000	000C5000 👗 Us 👔 📑	Pattern Ctrl+B Ime2\Windows\S	MAP -R	
006E0000				
00700000	00001000 1 Use 🗄 Reg	on view	PRV ER	
00710000 ()0001000 🧵 Us 👔 🚲 _{Fin}	references to region	PRV ER	
00720000	00001000		PRV ER	
00730000		ate memory		
00750000	00001000 I Use 📖 Fra	memory	PRV	
00760000	00001000 🧵 Use 📑 🗰		PRV ER	
00770000	00001000 🕌 Us 🕬 Add	virtual module	PRV ER	
00780000 0		> >		
007A0000	00001000 1 Use		PRV ER	
007B0000	0000F000 🧘 Us e 👎 Set	age Memory Rights	PRV ERI	W
007C0000	00035000	nry Breakpoint	PRV	
007F5000 0				w-G
008FD000	00003000 🧵 Use 🗅 Cor	r →	PRV -RI	W-G
00900000	0000E000 🧘 User		PRV -RI	W
74CD0000 (00001000 🖳 System a	phelp.dll	IMG -R	
74CD1000 (0007A000 System	. Lext		 W/
74D4D000	00003000 🔤 System	'.idata"		

Let's see the dumped file in IDA



The malware is successfully unpacked

SHA256:ab828f0e0555f88e3005387cb523f221a1933bbd7db4f05902a1e5cc289e7ba4

This blog is authored by Mostafa Farghaly(M4lcode).



Previous Post

Hard disk structure and analysis



Next Post

Email Forensics