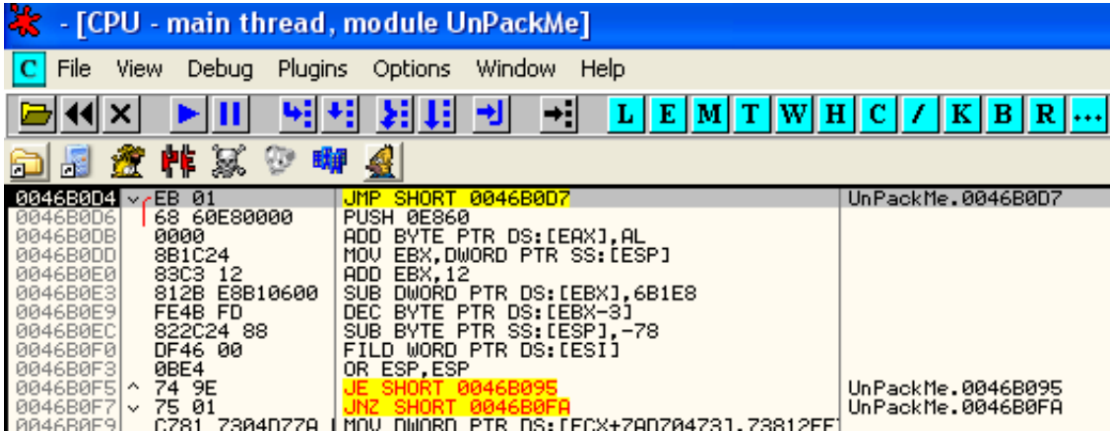


第四十八章-PeSpin V1.3.04 脱壳-Part1

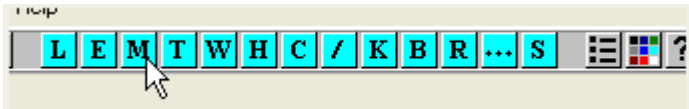
如果有的童鞋看了前两个章节,还是没有搞定那个 CrackMe 的话,可能考虑先跳过这两章从第 48 章开始看,可能确实这两章难度有点大。**(PS:其实还好,基础扎实的话,看起来还是没有问题的)**

本章我们来脱 PeSpin 这款壳,版本是 1.3.04。

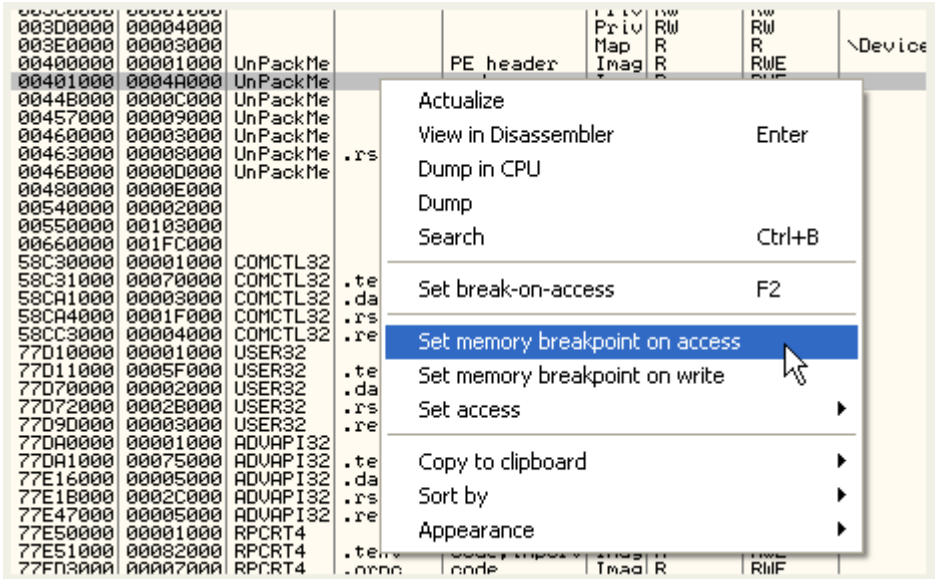
首先我们来定位该壳的 OEP。



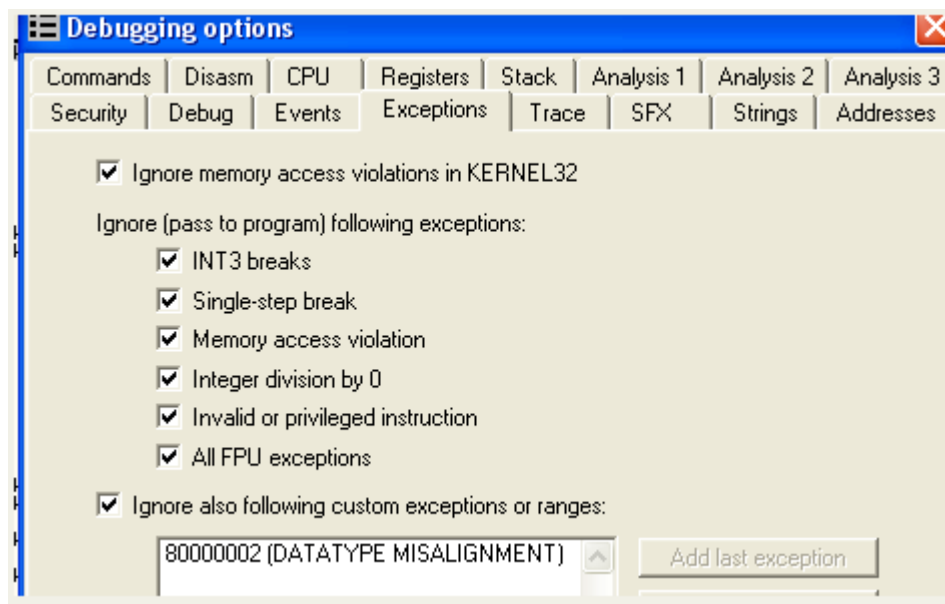
我们用专门定位 OEP 的 OD(olly_parcheado_para_vb)来加载它,断在了入口点处。



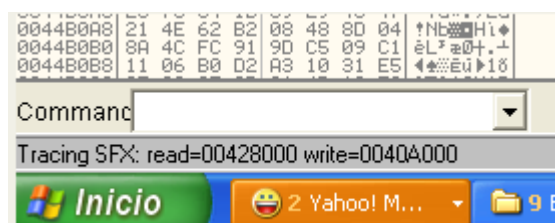
接着我们单击工具栏中的 M 按钮打开区间列表窗口,我们对主程序的代码段设置内存访问断点,大家还记得这个内存断点吧,这个 OD 的内存断点被 Patch 过,其仅仅是内存执行断点,只有执行才会断下,读取和写入并不会断下。



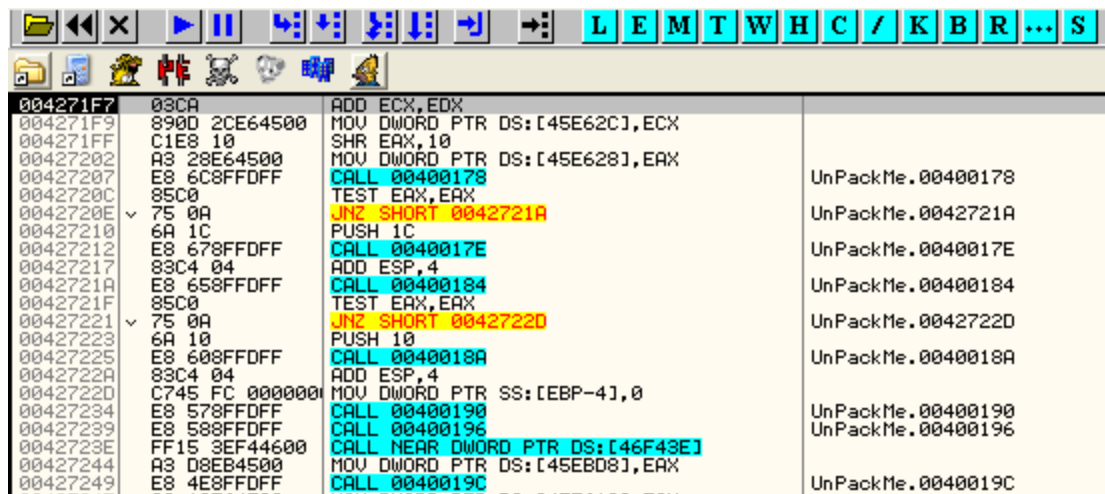
这里检查下看看要忽略的异常选项是否都勾选上了。



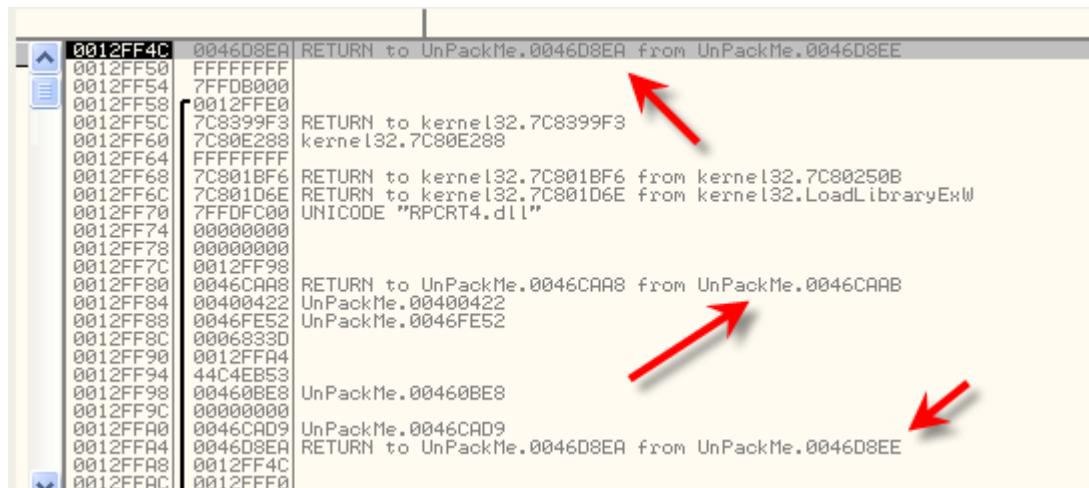
我们运行起来。



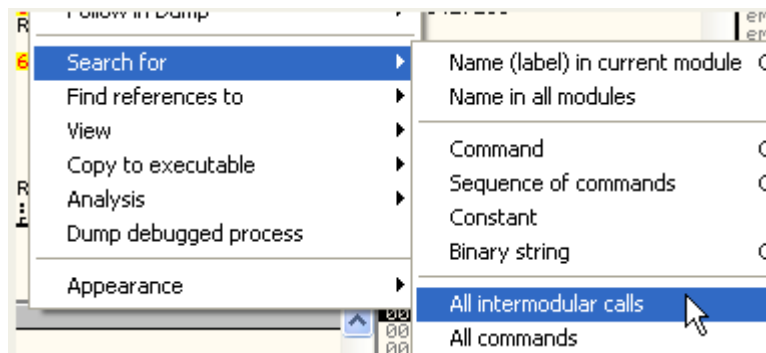
这里大家需要等待一段时间,不一会儿工夫断在了这里,应该是 OEP,但是入口点特征又不太像,应该是存在 Stolen Bytes。



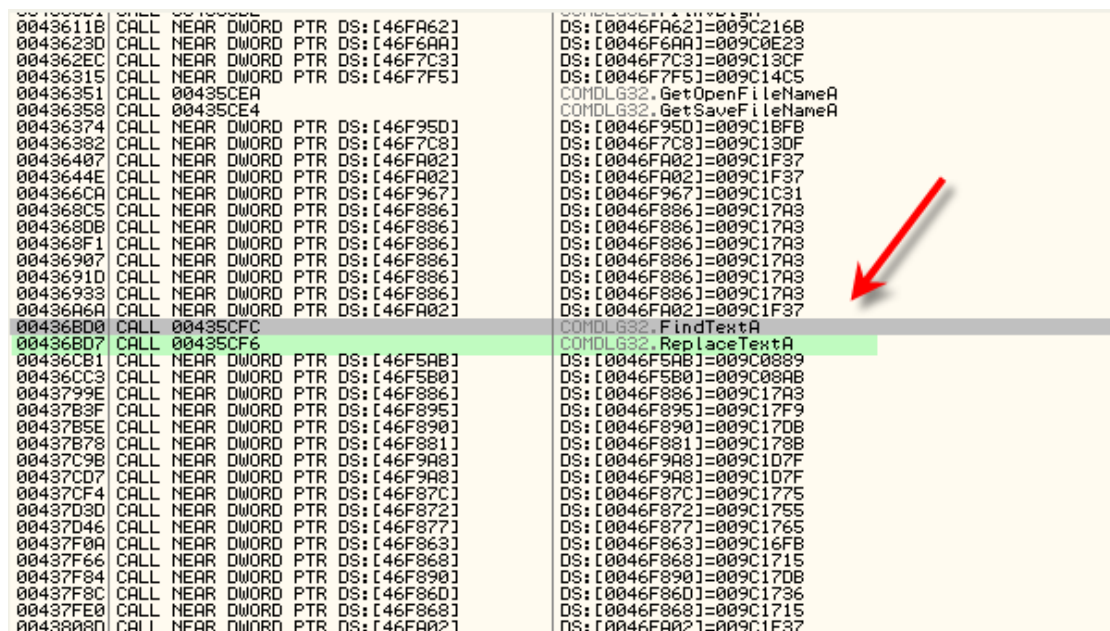
我们来看看堆栈:



从堆栈中我们可以看到在到达伪造的 OEP 之前经过了多层函数调用,执行了很多代码,这些代码应该就是我们要找到的 stolen bytes。



这里我们在反汇编窗口中单击鼠标右键选择 Search for-All intermodule calls,查看下主模块调用了哪些 API 函数。



我们可以看到只有少量调用处显示了 API 函数名称,我们随便选择一个。

00436BCD	50	PUSH EAX	
00436BCE	74 07	JE SHORT 00436BD7	UnPackMe.00436BD7
00436BD0	E8 27F1FFFF	CALL 00435CFC	JMP to COMDLG32.FindTextA
00436BD5	EB 05	JMP SHORT 00436BDC	UnPackMe.00436BDC
00436BD7	E8 1AF1FFFF	CALL 00435CFC	JMP to COMDLG32.ReplaceTextA
00436BDC	8BF8	MOV EDI,EAX	
00436BDE	E8 85170000	CALL 00438368	UnPackMe.00438368
00436BE3	85C0	TEST EAX,EAX	
00436BE5	75 0A	JNZ SHORT 00436BF1	UnPackMe.00436BF1
00436BE7	8B06	MOV EAX,DWORD PTR DS:[ESI]	
00435CBF	90	NOP	
00435CC0	FF25 43F44600	JMP NEAR DWORD PTR DS:[46F443]	
00435CC6	FF25 ECF64600	JMP NEAR DWORD PTR DS:[46F6EC]	
00435CCC	FF25 F1F64600	JMP NEAR DWORD PTR DS:[46F6F1]	
00435CD2	FF25 D00E4600	JMP NEAR DWORD PTR DS:[460ED0]	COMDLG32.ChooseFontA
00435CD8	FF25 D80E4600	JMP NEAR DWORD PTR DS:[460ED8]	COMDLG32.ChooseColorA
00435CDE	FF25 D40E4600	JMP NEAR DWORD PTR DS:[460ED4]	COMDLG32.PrintDlgA
00435CE4	FF25 CC0E4600	JMP NEAR DWORD PTR DS:[460ECC]	COMDLG32.GetSaveFileNameA
00435CEA	FF25 C80E4600	JMP NEAR DWORD PTR DS:[460EC8]	COMDLG32.GetOpenFileNameA
00435CF0	FF25 C40E4600	JMP NEAR DWORD PTR DS:[460EC4]	COMDLG32.GetFileTitleA
00435CF6	FF25 C00E4600	JMP NEAR DWORD PTR DS:[460EC0]	COMDLG32.ReplaceTextA
00435CFC	FF25 BC0E4600	JMP NEAR DWORD PTR DS:[460EBC]	COMDLG32.FindTextA
00435D02	FF25 B80E4600	JMP NEAR DWORD PTR DS:[460EB8]	COMDLG32.CommDlgExtendedError
00435D08	FF25 B00E4600	JMP NEAR DWORD PTR DS:[460EB0]	WINSP00L.ClosePrinter
00435D0E	FF25 940E4600	JMP NEAR DWORD PTR DS:[460E94]	WINSP00L.EndDocPrinter
00435D14	FF25 AC0E4600	JMP NEAR DWORD PTR DS:[460EAC]	WINSP00L.StartPagePrinter
00435D1A	FF25 A80E4600	JMP NEAR DWORD PTR DS:[460EA8]	WINSP00L.StartDocPrinterA
00435D20	FF25 A40E4600	JMP NEAR DWORD PTR DS:[460EA4]	WINSP00L.OpenPrinterA
00435D26	FF25 A00E4600	JMP NEAR DWORD PTR DS:[460EA0]	WINSP00L.EndPagePrinter
00435D2C	FF25 9C0E4600	JMP NEAR DWORD PTR DS:[460E9C]	WINSP00L.WritePrinter
00435D32	FF25 980E4600	JMP NEAR DWORD PTR DS:[460E98]	WINSP00L.DocumentPropertiesA
00435D38	FF25 240F4600	JMP NEAR DWORD PTR DS:[460F24]	OLEDLG.0leUIBusyA
00435D3E	CC	INT3	
00435D40	CC	INT3	

这里可以看到少量的间接调用 API 函数的指令-跳转表,我们任选一项定位到 IAT。

Address	Hex dump	ASCII
00460EBC	7C 86 37 76 B0 86 37 76	!37v37v
00460EC4	33 25 36 76 1E 31 36 76	3%6v16v
00460ECC	D8 7C 37 76 89 C2 37 76	i!7v87v
00460ED4	CD 46 38 76 CE EE 36 76	=F8vft-6v
00460EDC	00 00 00 00 48 D0 4C 77	...H3Lw
00460EE4	9C CB 4D 77 CC 42 4F 77	67HwlfB0w
00460EEC	2C D0 4C 77 DA F6 4C 77	.3LwrLw
00460EF4	73 33 50 77 10 64 4D 77	s3PwrdHw
00460EFC	03 0E 52 77 33 0F 52 77	03Rw3Rw
00460F04	40 A6 54 77 F1 A7 54 77	@3Tw3Tw
00460F0C	92 9C 4F 77 6F 57 52 77	E60w0wRw
00460F14	99 33 4E 77 B2 5D 4E 77	03Nw3Nw
00460F1C	90 C0 5A 77 00 00 00 00	E'Zw....
00460F24	F3 F0 CC 74 00 00 00 00	%-ft....
00460F2C	50 00 00 00 00 00 00 00	P.....
00460F34	00 00 00 00 00 00 00 00

这里我们可以看到 IAT 的结束地址为 460F28,我们往上拉。

Address	Hex dump	ASCII
00460E24	72 1A 06 00 62 1A 06 00	r+*.b+*.
00460E2C	50 1A 06 00 40 1A 06 00	P+*.@+*.
00460E34	2E 1A 06 00 1E 1A 06 00	.+*.A+*.
00460E3C	0C 1A 06 00 FE 19 06 00	.+*.m+*.
00460E44	F6 19 06 00 EA 19 06 00	+*.*.u+*.
00460E4C	DC 19 06 00 CC 19 06 00	+*.*.f+*.
00460E54	BC 19 06 00 AC 19 06 00	+*.*.k+*.
00460E5C	9C 19 06 00 90 19 06 00	+*.*.e+*.
00460E64	7C 19 06 00 6E 19 06 00	!+*.*.n+*.
00460E6C	60 19 06 00 4C 19 06 00	+*.*.L+*.
00460E74	3E 19 06 00 2E 19 06 00	>+*.*.+*.
00460E7C	20 19 06 00 12 19 06 00	+*.*.0+*.
00460E84	D0 1E 06 00 00 00 00 00	\$+*.*....
00460E8C	F7 A8 B1 76 00 00 00 00	-48v.....
00460E94	C8 74 F8 72 73 66 F9 72	4t°rsf-r
00460E9C	87 72 F8 72 43 80 F8 72	Cr°rCC°r
00460EA4	67 37 F9 72 FB 41 F9 72	g7-r'A-r
00460EAC	67 83 F8 72 90 53 F8 72	g3°reS°r
00460EB4	00 00 00 00 CE 00 37 76	...ft.7v
00460EBC	7C 86 37 76 B0 86 37 76	!37v37v
00460EC4	33 25 36 76 1E 31 36 76	3%6v16v
00460EC4	33 25 36 76 1E 31 36 76	3%6v16v

我们可以看到这些 IAT 项好像是经过重定向的,我们随便选中一项查看一下参考引用。

DS:[00460EBC]=7637867C (COMDLG32.FindTextA)

Address	Hex dump	ASCII
00460E24	72 1A 06 00 62 1A 06 00	r++..b++.
00460E2C	50 1A 06 00 40 1A 06 00	P++..@++.
00460E34	2E 1A 06 00	
00460E3C	0C 1A 06 00	Backup
00460E44	F6 19 06 00	Copy
00460E4C	DC 19 06 00	Binary
00460E54	BC 19 06 00	Label
00460E5C	9C 19 06 00	Breakpoint
00460E64	7C 19 06 00	Search for
00460E6C	60 19 06 00	Find references Ctrl+R
00460E74	3E 19 06 00	View executable file
00460E7C	20 19 06 00	
00460E84	D0 1E 06 00	
00460E8C	F7 A8 B1 76	
00460E94	C8 74 F8 72	
00460E9C	87 72 F8 72	
00460EA4	67 37 F9 72	
00460EAC	67 83 F8 72	

我们会发现这些项并没有参考引用处,我们继续往上翻,马上就可以看到 IAT 的起始地址了,所以说明刚刚这些查不到参考引用的项的确是 IAT 项,这些项被该壳重定向过了而已。

Address	Hex dump	ASCII
004607F4	00 00 00 00 00 00 00 00
004607FC	00 00 00 00 00 00 00 00
00460804	00 00 00 00 00 00 00 00
0046080C	00 00 00 00 00 00 00 00
00460814	00 00 00 00 F0 6B DA 77-krw
0046081C	1B 76 DA 77 F4 EA DA 77	+Urw9Urw
00460824	E7 EB DA 77 83 78 DA 77	!Urwaxrw
0046082C	00 00 00 00 DD 15 C5 58!S+X
00460834	2E BD C3 58 00 00 00 00	.cTX....
0046083C	56 27 06 00 64 27 06 00	U'..d'..
00460844	76 27 06 00 8A 27 06 00	U'..e'..
0046084C	96 27 06 00 A0 27 06 00	U'..a'..
00460854	E0 25 06 00 B0 27 06 00	0%..%..
0046085C	C4 27 06 00 D6 27 06 00	-'..i'..
00460864	42 27 06 00 F0 27 06 00	B'..-'..
0046086C	FC 27 06 00 0A 28 06 00	''..(..
00460874	18 28 06 00 22 28 06 00	↑(..'”(..
0046087C	2E 28 06 00 38 28 06 00	.(..8(..
00460884	48 28 06 00 D4 25 06 00	H(..E%..
0046088C	C8 25 06 00 B6 25 06 00	%%..A%..
00460894	AC 25 06 00 9C 25 06 00	%%..E%..
0046089C	82 25 06 00 6A 25 06 00	e%..j%..
004608A4	56 25 06 00 42 25 06 00	U%..B%..
004608AC	36 25 06 00 28 25 06 00	6%..(%..
004608B4	1E 25 06 00 0E 25 06 00	A%..B%..
004608BC	04 25 06 00 F8 24 06 00	o%..o%..
004608C4	E2 24 06 00 D0 24 06 00	O\$..S\$..
004608CC	BA 24 06 00 AA 24 06 00	\$..~\$..
004608D4	96 24 06 00 30 27 06 00	Q\$..0'..
004608DC	1E 27 06 00 08 27 06 00	A'..0'..

现在我们 IAT 的起始地址和结束位置都知道了,起始地址为 460818,结束地址为 460F28,下面我们的任务就是要修改 IAT 以及 stolen bytes 了。

首先我们来定位 stolen bytes,定位到伪造的 OEP 处,往上拉,可以看到一片零区域。

004271AD	90	NOP	
004271AE	90	NOP	
004271AF	90	NOP	
004271B0	0000	ADD BYTE PTR DS:[EAX],AL	
004271B2	0000	ADD BYTE PTR DS:[EAX],AL	
004271B4	0000	ADD BYTE PTR DS:[EAX],AL	
004271B6	0000	ADD BYTE PTR DS:[EAX],AL	
004271B8	0000	ADD BYTE PTR DS:[EAX],AL	
004271BA	0000	ADD BYTE PTR DS:[EAX],AL	
004271BC	0000	ADD BYTE PTR DS:[EAX],AL	
004271BE	0000	ADD BYTE PTR DS:[EAX],AL	
004271C0	0000	ADD BYTE PTR DS:[EAX],AL	
004271C2	0000	ADD BYTE PTR DS:[EAX],AL	
004271C4	0000	ADD BYTE PTR DS:[EAX],AL	
004271C6	0000	ADD BYTE PTR DS:[EAX],AL	
004271C8	0000	ADD BYTE PTR DS:[EAX],AL	
004271CA	0000	ADD BYTE PTR DS:[EAX],AL	
004271CC	0000	ADD BYTE PTR DS:[EAX],AL	
004271CE	0000	ADD BYTE PTR DS:[EAX],AL	
004271D0	0000	ADD BYTE PTR DS:[EAX],AL	
004271D2	0000	ADD BYTE PTR DS:[EAX],AL	
004271D4	0000	ADD BYTE PTR DS:[EAX],AL	
004271D6	0000	ADD BYTE PTR DS:[EAX],AL	
004271D8	0000	ADD BYTE PTR DS:[EAX],AL	
004271DA	0000	ADD BYTE PTR DS:[EAX],AL	
004271DC	0000	ADD BYTE PTR DS:[EAX],AL	
004271DE	0000	ADD BYTE PTR DS:[EAX],AL	
004271E0	0000	ADD BYTE PTR DS:[EAX],AL	
004271E2	0000	ADD BYTE PTR DS:[EAX],AL	
004271E4	0000	ADD BYTE PTR DS:[EAX],AL	
004271E6	0000	ADD BYTE PTR DS:[EAX],AL	
004271E8	0000	ADD BYTE PTR DS:[EAX],AL	
004271EA	0000	ADD BYTE PTR DS:[EAX],AL	
004271EC	0000	ADD BYTE PTR DS:[EAX],AL	
004271EE	0000	ADD BYTE PTR DS:[EAX],AL	
004271F0	0000	ADD BYTE PTR DS:[EAX],AL	
004271F2	0000	ADD BYTE PTR DS:[EAX],AL	
004271F4	0000	ADD BYTE PTR DS:[EAX],AL	
004271F6	0003	ADD BYTE PTR DS:[EAX],AL	
004271F8	CA 890D	RET 0Dh	Far return
004271FA	2C E6	SUB AL,0E6h	
004271FC	4E	INC EBX	

下面我们重启 OD,看下堆栈。

0012FFC4	7C816D4F	RETURN to kernel32.7C816D4F
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFD5000	
0012FFD4	8054AC5C	
0012FFD8	0012FFC8	
0012FFDC	82C6F020	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C8399F3	SE handler
0012FFE8	7C816D58	kernel32.7C816D58
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	0046B0D4	UnPackMe.<ModuleEntryPoint>
0012FFFC	00000000	

这里我们可以首次断在壳的入口点处时,栈顶指针指向了 12FFC4,也就是说,基于堆栈平衡的原理,当壳解密区段完毕,跳往真实的 OEP 处时,此时的栈顶指针也应该是指向的 12FFC4。我们知道通过情况下 OEP 处的第一个指令为 PUSH EBP,执行了这条指令后,栈顶指针应该指向的是 12FFC0,所以在数据窗口中定位到 12FFC0,给该内存单元设置硬件写入断点。如果该壳对硬件断点没有检测,我们这么做就没有问题,如果该壳对硬件断点有检测的话,就另当别论了。

Address	Hex dump	ASCII	
0012FFC0	F8 FF 12 00 4E 4D 01 7C	0 + 0m11	0012FFC4
0012FFC8	38 07 92 7		0012FFC8
0012FFD0	00 50 FD 7		0012FFCC
0012FFD8	C8 FF 12 0		0012FFD0
0012FFE0	FF FF FF F		0012FFD4
0012FFE8	58 6D 81 7		0012FFD8
0012FFF0	00 00 00 0		0012FFDC
0012FFF8	D4 B0 46 0		0012FFE0
			0012FFE4
			0012FFE8
			0012FFEC
			0012FFF0
			0012FFF4
			0012FFF8
			0012FFFC

我们运行起来。

0046B0D3	00EB	ADD BL,CH	
0046B0D5	0168 60	ADD DWORD PTR DS:[EAX+60],EBP	
0046B0D8	E8 00000000	CALL 0046B0D0	UnPackMe.0046B0D0
0046B0DD	8B1C24	MOV EBX,DWORD PTR SS:[ESP]	
0046B0E0	83C3 12	ADD EBX,12	
0046B0E3	812B E8B10600	SUB DWORD PTR DS:[EBX],6B1E8	
0046B0E9	5F4D FD	DEC BYTE PTR DS:[EBX+31]	

断在了这里,这里壳还没有解密完毕,我们继续按 F9 键运行。

0046CB08	8D0D C165D317	LEA ECX,DWORD PTR DS:[17D365C1]	
0046CB0E	55	PUSH EBP	
0046CB11	EB 01	JMP SHORT 0046CB12	UnPackMe.0046CB12
0046CB13	6A 8B	PUSH -75	
0046CB14	EC	IN AL,DX	I/O command
0046CB16	EB 01	JMP SHORT 0046CB17	UnPackMe.0046CB17
0046CB18	E6 6A	OUT 6A,AL	I/O command
0046CB1A	FFEB	JMP FAR EBX	Illegal use of regis
0046CB1D	01F9	ADD ECX,EBP	

断在了这里,这里有条 PUSH EBP 指令,有点像 OEP 处的指令,嘿嘿,我们继续往下单步跟踪,验证一下看看到底是不是 OEP 处的指令,对于无 JMP 指令我们并不会感兴趣,因为其并不影响寄存器组以及堆栈的状态。

0046CB12	8BEC	MOV EBP,ESP	
0046CB14	EB 01	JMP SHORT 0046CB17	UnPackMe.0046CB17
0046CB16	E6 6A	OUT 6A,AL	I/O command
0046CB18	FFEB	JMP FAR EBX	Illegal use of register
0046CB1A	01E9	ADD ECX,EBP	
0046CB1C	68 08BAF7FB	PUSH FBF7BA08	

0046CB17	6A FF	PUSH -1	
0046CB19	EB 01	JMP SHORT 0046CB1C	UnPackMe.0046CB1C
0046CB1B	E9 6808BAF7	JMP F800D388	
0046CB1D	FB	STI	
0046CB1F	018424 50F44D8	ADD DWORD PTR SS:[ESP],44F44D8	

0046CB1C	68 08BAF7FB	PUSH FBF7BA08	
0046CB21	810424 58544D0	ADD DWORD PTR SS:[ESP],44D5458	
0046CB28	68 9D5578DC	PUSH DC78559D	
0046CB2D	810424 2B3DCA2	ADD DWORD PTR SS:[ESP],23CA3D2B	
0046CB34	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
0046CB3A	EB 01	JMP SHORT 0046CB3D	
0046CB3C	C150 EB 01	RCL DWORD PTR DS:[EAX-15],1	UnPackMe

这里我们可以看到 PUSH FBF7BA08 这条指令,FBF7BA08 被压入堆栈以后,下面又要加上 44D5458 这个值。

我们来看看相加的结果等于多少。

0012FFB8	00450E60	UnPackMe.00450E60	
0012FFBC	FFFFFFFF		
0012FFC0	0012FFF0		
0012FFC4	7C816D4F	RETURN to kernel32.7C816D4F	
0012FFC8	7C920738	ntdll.7C920738	
0012FFCC	FFFFFFFF		
0012FFD0	7FFD5000		
0012FFD4	8054AC5C		

这里我们可以看到相加的结果等于 450E60,以上两条指令的作用等价于 PUSH 450E60。

0046CB1C	68 08BAF7FB	PUSH FBF7BA08	
0046CB21	810424 58544D0	ADD DWORD PTR SS:[ESP],44D5458	
0046CB28	68 9D5578DC	PUSH DC78559D	
0046CB2D	810424 2B3DCA2	ADD DWORD PTR SS:[ESP],23CA3D2B	
0046CB34	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
0046CB3A	EB 01	JMP SHORT 0046CB3D	
0046CB3C	C150 EB 01	RCL DWORD PTR DS:[EAX-15],1	UnPackMe.

接下来两条指令是类似的,等价于 PUSH 4292C8。

0012FFB4	004292C8	UnPackMe.004292C8	
0012FFB8	00450E60	UnPackMe.00450E60	
0012FFBC	FFFFFFFF		
0012FFC0	0012FFF0		
0012FFC4	7C816D4F	RETURN to kernel32.	
0012FFC8	7C920738	ntdll.7C920738	

我们继续。

0046CB2D	810424 2B3DCA2	ADD DWORD PTR SS:[ESP],23CA3D2B	
0046CB34	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
0046CB3A	EB 01	JMP SHORT 0046CB3D	
0046CB3C	C150 EB 01	RCL DWORD PTR DS:[EAX-15],1	Un
0046CB40	60 6A	DB 60 6A	c..

这里又是一条语句,我们继续。

0046CB3D	50	PUSH EAX	
0046CB3E	EB 01	JMP SHORT 0046CB41	
0046CB40	F3 64	PREFIX REP:	

0046CB41	64:8925 000000	MOV DWORD PTR FS:[0],ESP	
0046CB48	EB 01	JMP SHORT 0046CB4B	
0046CB4A	D283 C4A8EB01	ROL BYTE PTR DS:[EBX+1EBA8C4],CL	

继续:

0046CB4B	83C4 A8	ADD ESP,-58	
0046CB4E	EB 01	JMP SHORT 0046CB51	
0046CB50	6353 EB	ARPL WORD PTR DS:[EBX-15],DX	

0046CB51	53	PUSH EBX	
0046CB52	EB 01	JMP SHORT 0046CB55	
0046CB54	8756 EB	XCHG DWORD PTR DS:[ESI-15],EC	

0046CB55	56	PUSH ESI	
0046CB56	EB 01	JMP SHORT 0046CB59	
0046CB58	64 67	PUSH EDI	

0046CB59	57	PUSH EDI
0046CB5A	EB 01	JMP SHORT 0046CB5D
0046CB5C	07	POP ES

0046CB59	57	PUSH EDI
0046CB5A	EB 01	JMP SHORT 0046CB5D
0046CB5C	07	POP ES
0046CB5D	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
0046CB60	EB 01	JMP SHORT 0046CB63
0046CB62	71 FF	JMP SHORT 0046CB63

0046CB63	FF15 ECF54600	CALL NEAR DWORD PTR DS:[46F5EC]	
0046CB69	EB 01	JMP SHORT 0046CB6C	UnPa
0046CB6B	B7 33	MOV BH,33	

这里应该是调用一个 API 函数,等我们修改了 IAT 就可以看出来了。

0046CB6C	33D2	XOR EDX,EDX
0046CB6E	EB 01	JMP SHORT 0046CB71
0046CB70	07	POP ES

0046CB71	8AD4	MOV DL,AH
0046CB73	EB 01	JMP SHORT 0046CB76
0046CB75	C189 1534E645	ROR DWORD PTR DS:[ECX+45],CL
0046CB77	07	POP ES

0046CB76	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX	
0046CB7C	EB 01	JMP SHORT 0046CB7F	UnP
0046CB7E	E6 8B	OUT 8B,AL	I/O
0046CB80	07	POP ES	

0046CB7F	8BC8	MOV ECX,EAX
0046CB81	EB 01	JMP SHORT 0046CB84
0046CB83	07	POP ES

0046CB84	81E1 FF000000	AND ECX,0FF
0046CB8A	EB 01	JMP SHORT 0046CB8D
0046CB8C	53	PUSH EBX

0046CB8A	EB 01	JMP SHORT 0046CB8D
0046CB8C	53	PUSH EBX
0046CB8D	890D 30E64500	MOV DWORD PTR DS:[45E63D],ECX
0046CB93	EB 01	JMP SHORT 0046CB96
0046CB95	4F	DEC EDI
0046CB96	C1E1 08	SHL ECX,8

0046CB8D	890D 30E64500	MOV DWORD PTR DS:[45E63D],ECX
0046CB93	EB 01	JMP SHORT 0046CB96
0046CB95	4F	DEC EDI
0046CB96	C1E1 08	SHL ECX,8
0046CB99	EB 01	JMP SHORT 0046CB9C
0046CB9B	43	INC EBX

0046CB93	EB 01	JMP SHORT 0046CB96	UnPackMe.0046CB96
0046CB95	4F	DEC EDI	
0046CB96	C1E1 08	SHL ECX,8	
0046CB99	EB 01	JMP SHORT 0046CB9C	UnPackMe.0046CB9C
0046CB9B	43	INC EBX	
0046CB9C	E9 56A6FBFF	JMP 004271F7	UnPackMe.004271F7
0046CB9E	07	POP ES	

跟到了这里,可以看到这里将要跳往伪造的 OEP 了,嘿嘿,stolen bytes 都知道了。

004271F7	03CA	ADD ECX,EDX	
004271F9	890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX	
004271FF	C1E8 10	SHR EAX,10	
00427202	A3 28E64500	MOV DWORD PTR DS:[45E628],EAX	
00427207	E8 6C8FFDFF	CALL 00400178	UnPackMe.00400178
0042720C	85C0	TEST EAX,EAX	
0042720E	75 0A	JNZ SHORT 0042721A	UnPackMe.0042721A
00427210	6A 1C	PUSH 1C	
00427212	E8 678FFDFF	CALL 0040017E	UnPackMe.0040017E
00427217	83C4 04	ADD ESP,4	
0042721A	E8 658FFDFF	CALL 00400184	UnPackMe.00400184
0042721F	85C0	TEST EAX,EAX	
00427221	75 0A	JNZ SHORT 0042722D	UnPackMe.0042722D
00427223	6A 10	PUSH 10	
00427225	E8 608FFDFF	CALL 0040018A	UnPackMe.0040018A
0042722A	83C4 04	ADD ESP,4	
0042722D	C745 FC 000000	MOV DWORD PTR SS:[EBP-4],0	
00427234	E8 578FFDFF	CALL 00400190	UnPackMe.00400190
00427239	E8 588FFDFF	CALL 00400196	UnPackMe.00400196
0042723E	FF15 3EF44600	CALL NEAR DWORD PTR DS:[46F43E]	
00427244	A3 D8EB4500	MOV DWORD PTR DS:[45EBD8],EAX	
00427249	E8 4E8FFDFF	CALL 0040019C	UnPackMe.0040019C
0042724F	A3 1AF64500	MOV DWORD PTR DS:[45F610],EAX	

我们将 stolen bytes 都拷贝到 OEP 处。

004271AF	90	NOP	
004271B0	55	PUSH EBP	
004271B1	8BEC	MOV EBP,ESP	
004271B3	6A FF	PUSH -1	
004271B5	68 600E4500	PUSH 450E60	
004271BA	68 C8924200	PUSH 4292C8	
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
004271C5	50	PUSH EAX	
004271C6	64:8925 000000	MOV DWORD PTR FS:[0],ESP	
004271CD	83C4 A8	ADD ESP,-58	
004271D0	53	PUSH EBX	
004271D1	56	PUSH ESI	
004271D2	57	PUSH EDI	
004271D3	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
004271D6	FF15 ECF54600	CALL NEAR DWORD PTR DS:[46F5EC]	
004271DC	00D2	ADD DL,DL	
004271DE	8AD4	MOV DL,AH	
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX	
004271E6	8BC8	MOV ECX,EAX	
004271E8	81E1 FF000000	AND ECX,0FF	
004271EE	890D 30E64500	MOV DWORD PTR DS:[45E630],ECX	
004271F4	C1E1 08	SHL ECX,8	
004271F7	03CA	ADD ECX,EDX	
004271F9	890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX	
004271FF	C1E8 10	SHR EAX,10	
00427202	A3 28E64500	MOV DWORD PTR DS:[45E628],EAX	
00427207	E8 6C8FFDFF	CALL 00400178	UnPackMe.00400178

好了,我们可以看到空间刚刚好,这里 stolen bytes 我们就修复完毕了,下一章我们来修改 IAT。