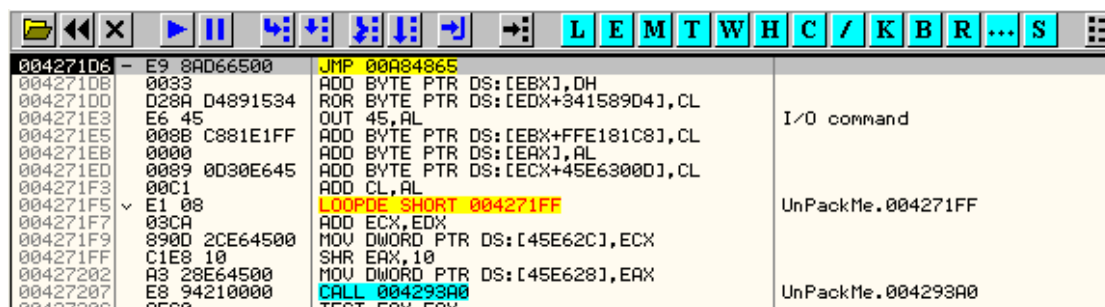


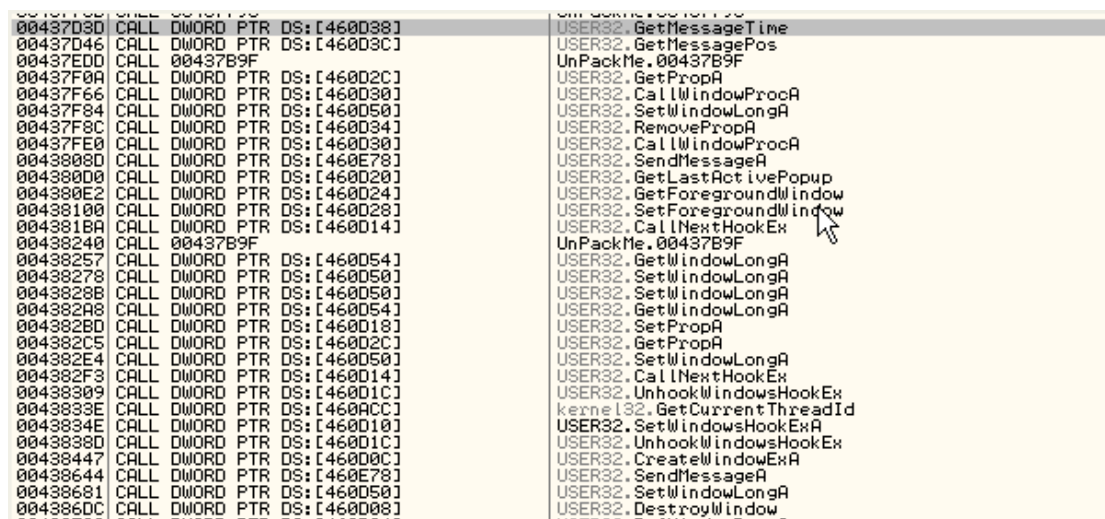
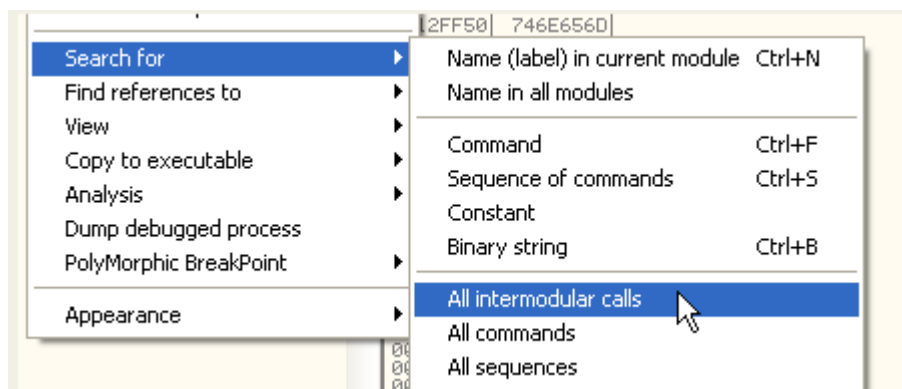
第四十一章-神马是 AntiDump

本章我们继续脱 PELock,修复其 IAT。

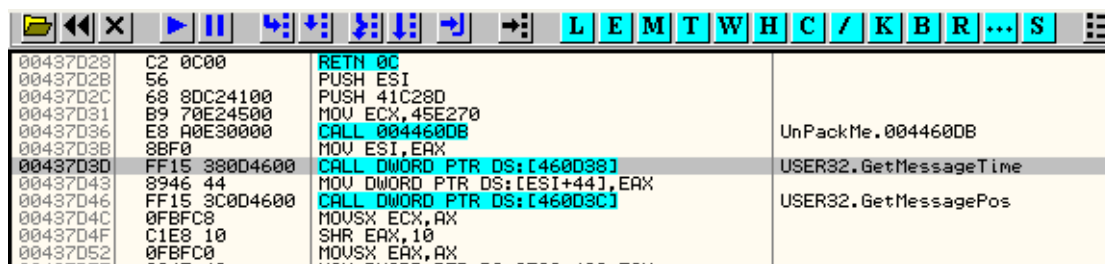
这里我们仍然采用最后一次异常法定位到 OEP 处。



好了,现在我们到达了假的 OEP 处,接着我们单击鼠标右键选择 Search for-All intermodular calls,查看 API 函数的调用列表。



这里我们随便选择一个,选中并双击之。



这里我们可以看到该 API 函数的入口地址位于 460D38 这个内存单元中的,也就是说 460D38 为 IAT 的其中一项。好,在数据窗口

中定位到该地址。

Address	Hex dump	ASCII
00460D38	08 C4 01 77 94 BF 01 77	[-0w070w
00460D40	7D BC 01 77 1B C0 01 77	[-0w+[-0w
00460D48	28 8E 01 77 79 F6 02 77	([-0wy[-0w
00460D50	0D 06 01 77 5D 94 01 77	.[-0w]00w
00460D58	C3 91 02 77 E2 16 02 77	[-0w0[-0w
00460D60	4F 02 03 77 15 E5 03 77	000w300w
00460D68	3D 02 03 77 AE 90 02 77	-00w<00w
00460D70	88 C1 01 77 8E 1A 03 77	[-0w0[-0w
00460D78	2F EA 01 77 12 0B 02 77	/00w000w
00460D80	12 0B 02 77 44 00 03 77	000wD00w
00460D88	5F F7 02 77 73 F9 02 77	-00w<00w
00460D90	DC F6 02 77 37 F4 02 77	[-0w700w
00460D98	28 F7 02 77 42 8C 01 77	([-0wB[-0w
00460DA0	2E 8C 01 77 8B 14 03 77	.[-0w[-0w
00460DA8	FE EC 03 77 83 F7 04 77	[-0w0[-0w
00460DB0	DE F2 02 77 DF 1A 03 77	i[-0w[-0w
00460DB8	F6 F0 04 77 9C F3 04 77	+[-0w0[-0w
00460DC0	33 F2 02 77 6C C9 01 77	3[-0w[-0w
00460DC8	F6 8B 01 77 B8 96 01 77	+[-0w0[-0w
00460DD0	0C 94 01 77 61 C6 03 77	.00w000w
00460DD8	01 FF 03 77 0A 00 03 77	[-0w0[-0w

我们可以看到这部分 IAT 项是正确的,我们往上看,定位 IAT 的起始位置。

Address	Hex dump	ASCII
004607F8	9E BB EB 32 89 1A 41 0D	[-0w0[-0w
00460800	88 E2 6F 0D 8B 40 6D 50	[-0w0[-0w
00460808	C0 BA BB E9 E9 00 9F 06	[-0w0[-0w
00460810	1B B7 A8 38 F0 F6 28 3F	[-0w0[-0w
00460818	F0 6B 0A 77 1B 76 0A 77	[-0w0[-0w
00460820	F4 EA 0A 77 E7 EB 0A 77	[-0w0[-0w
00460828	83 78 0A 77 0D 5B 02 70	[-0w0[-0w
00460830	DD 15 C5 58 2E 8D C3 58	[-0w0[-0w
00460838	52 8B F5 31 04 6A EF 77	[-0w0[-0w
00460840	66 95 EF 77 89 6A EF 77	[-0w0[-0w
00460848	F3 AD EF 77 ED 09 EF 77	[-0w0[-0w
00460850	99 8B EF 77 C0 B5 EF 77	[-0w0[-0w
00460858	2A 7D EF 77 B2 7C EF 77	[-0w0[-0w
00460860	77 53 F2 77 1E C9 F1 77	[-0w0[-0w
00460868	0C BC EF 77 52 D4 EF 77	[-0w0[-0w
00460870	FA 8D EF 77 F1 DD EF 77	[-0w0[-0w
00460878	51 B2 EF 77 26 D5 EF 77	[-0w0[-0w
00460880	2A E3 EF 77 5F 39 F2 77	[-0w0[-0w
00460888	71 B4 EF 77 2E AD EF 77	[-0w0[-0w
00460890	E1 61 EF 77 B8 85 EF 77	[-0w0[-0w
00460898	CC D2 EF 77 43 70 EF 77	[-0w0[-0w
004608A0	FB EA F0 77 12 83 EF 77	[-0w0[-0w
004608A8	01 72 F0 77 A9 34 F0 77	[-0w0[-0w
004608B0	D5 93 EF 77 68 EF EF 77	[-0w0[-0w
004608B8	AA D2 EF 77 B2 6F EF 77	[-0w0[-0w
004608C0	3F 38 F2 77 D6 E8 EF 77	[-0w0[-0w
004608C8	68 E0 EF 77 00 60 EF 77	[-0w0[-0w
004608D0	01 FF 03 77 0A 00 03 77	[-0w0[-0w

这里很明显 460818 为 IAT 的起始位置,因为上面的项,查看参考引用的话,会发现没有参考引用处,并且也不属于系统 DLL 的区段。

这里很幸运,IAT 项都是正确的,没有被重定向,所以我们并不需要修复重定向,接下来,我们来定位 IAT 的结束位置。

Address	Hex dump	ASCII
00460E78	9A F3 02 77 B5 37 02 77	[-0w0[-0w
00460E80	78 8E 01 77 8B EE 04 77	[-0w0[-0w
00460E88	B7 52 81 9D F7 A8 B1 76	[-0w0[-0w
00460E90	EF 54 91 08 C8 74 F8 72	[-0w0[-0w
00460E98	73 66 F9 72 87 72 F8 72	[-0w0[-0w
00460EA0	43 80 F8 72 67 37 F9 72	[-0w0[-0w
00460EA8	FB 41 F9 72 67 83 F8 72	[-0w0[-0w
00460EB0	90 53 F8 72 E1 58 FD 13	[-0w0[-0w
00460EB8	CE 00 37 76 7C 86 37 76	[-0w0[-0w
00460EC0	B0 86 37 76 33 25 36 76	[-0w0[-0w
00460EC8	1E 31 36 76 D8 7C 37 76	[-0w0[-0w
00460ED0	89 C2 37 76 CD 46 38 76	[-0w0[-0w
00460ED8	CE EE 36 76 00 EC B6 EC	[-0w0[-0w
00460EE0	48 D0 4C 77 9C CB 4D 77	[-0w0[-0w
00460EE8	CC 42 4F 77 2C D0 4C 77	[-0w0[-0w
00460EF0	DA F6 4C 77 73 33 50 77	[-0w0[-0w
00460EF8	10 64 4D 77 03 0E 52 77	[-0w0[-0w
00460F00	33 0F 52 77 40 A6 54 77	[-0w0[-0w
00460F08	F1 A7 54 77 92 9C 4F 77	[-0w0[-0w
00460F10	6F 57 52 77 99 33 4E 77	[-0w0[-0w
00460F18	B2 5D 4E 77 90 C0 5A 77	[-0w0[-0w
00460F20	A6 2C A9 56 F3 F0 CC 74	[-0w0[-0w
00460F28	8E 3B E9 56 64 97 B5 F6	[-0w0[-0w
00460F30	9E 7D 00 00 00 00 00 00	[-0w0[-0w

这里灰色的部分没有问题,但是 460F24 这一项就是比较可疑了,这一项没有参考引用。灰色的这部分是属于 ole32.dll 的代码段的。

7749C000	00006000	comctl32	.reloc	relocations	Image	RWE	
774B0000	00001000	ole32		PE header	Image	RWE	
774B1000	0011F000	ole32	.text	code, imports	Image	RWE	
775D0000	00006000	ole32	.orpc	code	Image	RWE	
775D6000	00007000	ole32	.data	data	Image	RWE	
775D0000	00002000	ole32	.rsrc	resources	Image	RWE	
775DF000	0000E000	ole32	.reloc	relocations	Image	RWE	

这里我们选中属于该 DLL 的某些项,会发现也没有参考引用。

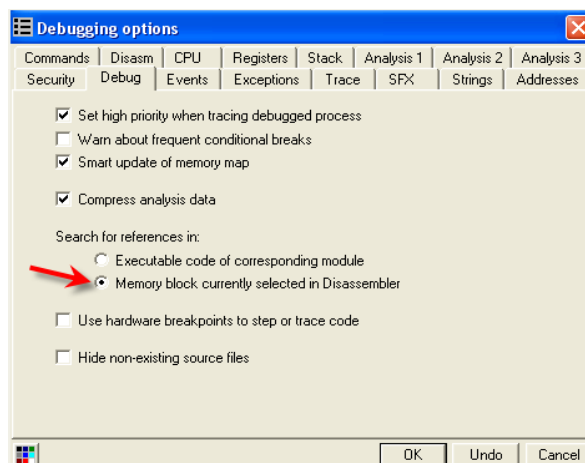
004271D6	E9 8AD66500	JMP 00A84865	
004271D8	0033	ADD BYTE PTR DS:[EBX],0H	
004271DD	D28A D4891534	ROR BYTE PTR DS:[EDX+341589D4],CL	
004271E3	E6 45	OUT 45,AL	I/O command
004271E5	008B C881E1FF	ADD BYTE PTR DS:[EBX+FFE181C8],CL	
004271EB	0000	ADD BYTE PTR DS:[EAX],AL	
004271ED	0089 0D30E645	ADD BYTE PTR DS:[ECX+45E630D1],CL	
004271F3	00C1	ADD CL,AL	
004271F5	E1 08	LOOPDE SHORT 004271FF	UnPackMe.004271FF
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 94210000	CALL 004293A0	UnPackMe.004293A0
0042720C	85C0	TEST EAX,EAX	
0042720E	75 0A	JNZ SHORT 0042721A	UnPackMe.0042721A
00427210	6A 1C	PUSH 1C	
00427212	E8 49010000	CALL 00427360	UnPackMe.00427360
00427217	83C4 04	ADD ESP,4	
0042721D	E9 D12E0000	JMP 00420150	UnPackMe.00420150

我们回到 OEP 处,可以看到该 JMP 指令将要跳转的一个地址为 Axxxxx(在我的机器上为这种形式,在其他的机器上可能会不一样)的形式,该地址不属于 exe 所在的区段,我们来看一看区段列表。

00400000	00001000	UnPackMe			Priv	RW	
00401000	00004000	UnPackMe	.teddy	PE header	Image	R	RWE
0040B000	0000C000	UnPackMe	.teddy	code	Image	R	RWE
00457000	00009000	UnPackMe	.teddy		Image	R	RWE
00460000	00003000	UnPackMe	.teddy		Image	R	RWE
00463000	00008000	UnPackMe	.teddy	resources	Image	R	RWE
0046B000	0000A000	UnPackMe	.teddy	SFX, imports	Image	R	RWE
00480000	00021000				Priv	RW	RW
00480000	00009000				Map	R E	R E
00570000	00002000				Map	R E	R E
00580000	00103000				Map	R	R
00690000	00001000				Priv	RW	RW
006A0000	00141000				Map	R E	R E
009A0000	00001000				Priv	RW	RW
009B0000	00004000				Priv	RW	RW
009C0000	00003000				Map	R	R
009D0000	00001000				Priv	RW	RW
00A50000	00004000				Priv	RW	RW
00A60000	00003000				Priv	RW	RW
00A70000	00002000				Map	R	R
00A80000	0003C000				Priv	RW	RW
00AC0000	00001000				Priv	RW	RW
00BC0000	00002000				Priv	RW	RW
01280000	00002000				Map	R	R
58C30000	00001000	COMCTL32		PE header	Image	R	RWE
58C31000	00070000	COMCTL32	.text	code, imports	Image	R	RWE
58CA1000	00003000	COMCTL32	.data	data	Image	R	RWE

我们知道原始程序在 OEP 附近是太可能跳转到 exe 之外的区段的,那么 Axxxxx 所在的这个区段很可能是通过 VirtualAlloc 或者类似的函数创建的,所以这个区段很可能是壳创建的。

在这个区段中壳可能会将 IAT 的中某些值取出来加以变换就转化为 API 函数的调用了。下面我们来对 OD 的调试选项加以配置。



Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped
00380000	00001000				Priv	RWE	RWE	
00390000	00001000				Priv	RWE	RWE	
003A0000	00001000				Priv	RW	RW	
00400000	00001000	UnPackMe		PE header	Imag	R	RWE	
00401000	0004A000	UnPackMe	.teddy	code	Imag	R	RWE	
0044B000	0000C000	UnPackMe	.teddy		Imag	R	RWE	
00457000	00009000	UnPackMe	.teddy		Imag	R	RWE	
00460000	00003000	UnPackMe	.teddy		Imag	R	RWE	
00463000	00008000	UnPackMe	.teddy	resources	Imag	R	RWE	
0046B000	0000A000	UnPackMe	.teddy	SFX, imports	Imag	R	RWE	
00480000	00021000				Priv	RW	RW	
004B0000	00009000				Map	R E	R E	
00570000	00002000				Map	R E	R E	
00580000	00103000				Map	R	R	
00690000	00001000				Priv	RW	RW	
006A0000	00141000				Map	R E	R E	
009A0000	00001000				Priv	RW	RW	
009B0000	00004000				Priv	RW	RW	
009C0000	00003000				Map	R	R	\Device
009D0000	00001000				Priv	RW	RW	
00A50000	00004000				Priv	RW	RW	
00A60000	00003000				Priv	RW	RW	
00A70000	00002000				Map	R	R	
00A80000	00003000				Priv	RW	RW	
00AC0000	00001000				Priv	RW	RW	
00BCE000	00002000				Priv	RW	Gua: RW	
01280000	00002000				Map	R	R	
58C30000	00001000	COMCTL32		PE header	Imag	R	RWE	

Address	Disassembly	Comment
00AC0000	JMP SHORT 00AC0004	
00AC0002	OR AL,49	
00AC0004	JMP SHORT 00AC0008	
00AC0006	INT 20	
00AC0008	PUSH 7C81082F	
00AC000A	JMP SHORT 00AC0011	
00AC000C	INT 20	
00AC000E	RET	
00AC0010	LEA ESP,DWORD PTR SS:[ESP-14]	
00AC0012	PUSH 0AC0021	
00AC0014	JMP 7C81094C	
00AC0016	OR AL,0C1	
00AC0018	LOCK ADD BYTE PTR DS:[EBX+CD02EBFF],CL	kernel32.7C81094C
00AC001A	AND BYTE PTR SS:[EBP+68],DL	LOCK prefix
00AC001C	XOR EAX,EB0AC00	
00AC001E	ADD DWORD PTR DS:[EBX+EAX*8],ECX	
00AC0020	ADD BYTE PTR DS:[EBX+C02EBEC],CL	
00AC0022	DEC ECX	
00AC0024	PUSH DWORD PTR SS:[EBP+8]	
00AC0026	FF75 08	

也看不出什么端倪。

我们回头再看看 A80000 这个区段,我们注意到 A8003D 这处 CALL [460930],其对应的机器码为 FF15 30094600,也就是说如果 CALL [460F24]的话,其对应的机器码应该是 FF15 240F4600,也就是说可以通过这种方式来调用 API。

Address	Disassembly	Comment
00A80031	LEA ESI,DWORD PTR DS:[ESI+77053A7]	
00A80033	JMP 004010AB	
00A80035	OUTS DX,DWORD PTR ES:[EDI]	UnPackMe.004010AB
00A80037	CALL DWORD PTR DS:[460930]	I/O command
00A80039	JMP 004010CF	GDI32.DeleteObject
00A8003B	XCHG EDI,EDI	UnPackMe.004010CF
00A8003D	ADC EAX,460B9C	
00A8003F	JMP 00401118	UnPackMe.00401118
00A80041	INC EAX	Illegal use of register

好了,下面我们搜索一下 FF15 240F4600 这串二进制串。

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00001000				Priv	RW	RW	
00020000	00001000				Priv	RW	RW	
0012C000	00001000				Priv	RW	Gu	
0012D000	00003000			stack of na	Priv	RW	Gu	
00130000	00002000				Priv	RW	RW	
00140000	00003000				Map	R	R	
00150000	00023000				Priv	RW	RW	
00250000	00006000				Priv	RW	RW	
00260000	00003000				Map	R	R	
00270000	00016000				Map	R	R	
00290000	0003D000				Map	R	R	
002D0000	000041000				Map	R	R	
00320000	00006000				Map	R	R	
00330000	000041000				Map	R	R	
00390000	00001000				Priv	RME	RME	
00390000	00001000				Priv	RME	RME	
003A0000	00010000				Priv	RW	RW	
00400000	00001000	UnPackf		FE header	Priv	RW	RW	
00401000	00004000	UnPackf	.teddy	code	Priv	RW	RW	
0044B000	0000C000	UnPackf	.teddy		Priv	RW	RW	
00457000	00009000	UnPackf	.teddy		Priv	RW	RW	
00460000	00003000	UnPackf	.teddy		Priv	RW	RW	
00463000	00008000	UnPackf	.teddy	resources	Priv	RW	RW	
0046B000	0000A000	UnPackf	.teddy	SFX, imports	Priv	RW	RW	
00490000	00021000				Priv	RW	RW	
004B0000	00003000				Map	R E	R E	
00570000	00002000				Map	R E	R E	
00580000	00103000				Map	R	R	
00630000	00001000				Priv	RW	RW	
006A0000	00141000				Map	R E	R E	
009A0000	00001000				Priv	RW	RW	
009B0000	00004000				Priv	RW	RW	
009C0000	00003000				Map	R	R	
009D0000	00001000				Priv	RW	RW	
00A50000	00004000				Priv	RW	RW	
00A60000	00003000				Priv	RW	RW	
00A70000	00002000				Map	R	R	
00A80000	0003C000				Priv	RW	RW	
00AC0000	00001000				Priv	RW	RW	
00BCE000	00002000				Priv	RW	RW	
01200000	00002000				Map	R	R	

单击工具栏上面的 M 按钮打开区间列表窗口,选中第一个区间,单击鼠标右键选择-Search。

Type	Access	Initial	Mapped as
Priv	RW	RW	
Priv	RW	RW	
ma	RW	Gu	
Priv	RW	Gu	
Priv	RWE		
Map	R		
Priv	RW		
Priv	RW		
Map	RW		
Map	R		
Map	R		
Map	R		

Actualize
 Dump in CPU
 Dump
 Search Ctrl+B
 Set break-on-access F2

Enter binary string to search for

ASCII

UNICODE

HEX +06

☒ Entire block

☐ Case sensitive

<< >>

OK Cancel

在 16 进制编辑框中写入 FF15240F4600 字节序列,单击 OK,会发现,什么也没找到,说明有可能不是通过间接 CALL,我们去掉前面的 FF15,再次搜索。

Enter binary string to search for

ASCII

UNICODE

HEX +04

☒ Entire block

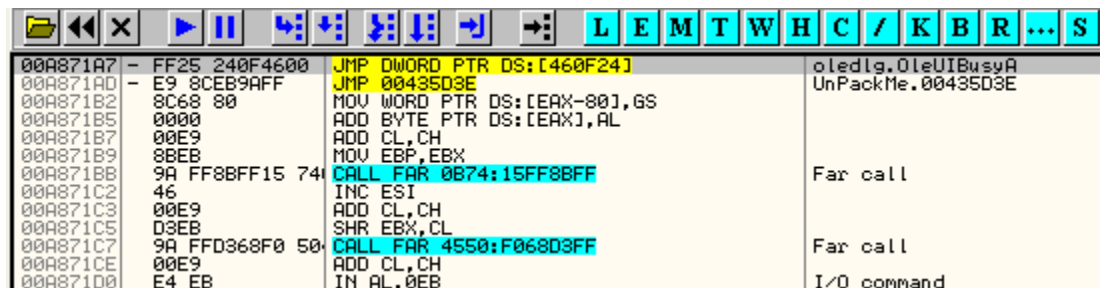
☐ Case sensitive

<< >>

OK Cancel



正好定位到了壳创建的这个区段,我们其反汇编代码是什么。



这里我们可以看到是通过 JMP 来调用的,而非 CALL,所以机器码为 FF25,并非 FF15。也就是说 460F24 也是 IAT 中的一项。

Address	Hex dump	ASCII
00460E78	9A F3 02 77 B5 37 02 77	U%EW7Ew
00460E80	78 8E 01 77 8B EE 04 77	xAbwi"ew
00460E88	B7 52 81 9D F7 A8 B1 76	ARU0-48v
00460E90	EF 54 91 08 C8 74 F8 72	*Tai"t"r
00460E98	73 66 F9 72 87 72 F8 72	sf-rGr"r
00460EA0	43 80 F8 72 67 37 F9 72	CC"rg7-r
00460EA8	FB 41 F9 72 67 83 F8 72	*A-rga"r
00460EB0	90 53 F8 72 E1 58 FD 13	ES"rpX"!!
00460EB8	CE 00 37 76 7C 86 37 76	fr,7v!37v
00460EC0	B0 86 37 76 33 25 36 76	337v376v
00460EC8	1E 31 36 76 D8 7C 37 76	16vii7v
00460ED0	89 C2 37 76 CD 46 38 76	et7v=F8v
00460ED8	CE EE 36 76 00 EC B6 EC	fr"6v,yay
00460EE0	48 D0 4C 77 9C C8 4D 77	HsLw6frMw
00460EE8	CC 42 4F 77 2C D0 4C 77	frB0w,sLw
00460EF0	DA F6 4C 77 73 33 50 77	rt+Lws3Pw
00460EF8	10 64 4D 77 03 0E 52 77	rdMw08Rw
00460F00	33 0F 52 77 40 A6 54 77	33Rw03Tw
00460F08	F1 A7 54 77 32 9C 4F 77	z2Twie60w
00460F10	6F 57 52 77 99 33 4E 77	oUrW03Nw
00460F18	B2 5D 4E 77 90 C0 5A 77	80JNwE"Zw
00460F20	A6 2C A8 56 F3 F0 CC 74	3,-U%=-frE
00460F28	8E 38 E9 4E 64 97 B5 F6	A;UNduA+
00460F30	9E 7D D9 B0 5F 95 14 1B	x)7-07+
00460F38	61 02 D2 16 14 62 59 CC	a0E-7bVfr
00460F40	A9 4F 28 35 EF 76 0B 48	00(5"v3H
00460F48	CB 6F 5C A7 98 A9 2B 3C	fr"2g0+<
00460F50	CD A6 F6 5F 53 A3 16 A3	=3+ Su-u

所以说 IAT 的结束位置为 460F28。

IAT 的长度 = 结束位置 - 起始位置 = 460F28 - 460818 = 710



因此:

OEP 的 RVA = 271B0

IAT 起始地址的 RVA = 60818

现在我们可以看到 IAT 项都是正确的了,我们回到假的 OEP 处,将 stolen bytes 填充上。

stolen bytes 是 55 8B EC 6A FF 68 60 0E 45 00 68 C8 92 42 00 64 A1 00 00 00 00 50 64 89 25 00 00 00 00 83 C4 A8 53 56 57 89 65 E8 。

我们在数据窗口中定位到 4271B0 处,按照第 39 章介绍的方式将 stolen bytes 填充好。

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	E9 8AD66500	JMP 00A84865	
004271DB	0033	ADD BYTE PTR DS:[EBX],DH	
004271DD	D28A D4891534	ROR BYTE PTR DS:[EDX+341589D4],CL	
004271E3	E6 45	OUT 45,AL	I/O command
004271E5	008B C881E1FF	ADD BYTE PTR DS:[EBX+FFE181C8],CL	
004271EB	0000	ADD BYTE PTR DS:[EAX],AL	
004271ED	0089 0D30E645	ADD BYTE PTR DS:[ECX+45E6300D],CL	
004271F3	00C1	ADD CL,AL	
004271F5	E1 08	LOOPDE SHORT 004271FF	UnPackMe.004271FF
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 94210000	CALL 004293A0	UnPackMe.004293A0
0042720C	85C0	TEST EAX,EAX	
0042720E	75 0A	JNZ SHORT 0042721A	UnPackMe.0042721A
00427210	60 1C	PUSH 1C	

Address	Hex dump	ASCII
00427190	F9 00 00 01 00 75 08 80	...0.000
00427198	CC 02 EB 03 80 CC 03 F7	if00000000
004271A0	C2 00 00 04 00 74 03 80	T...t000
004271A8	CC 10 C3 90 90 90 90 90	if00000000
004271B0	55 8B EC 6A FF 68 60 0E	Utyj h"0
004271B8	45 00 68 C8 92 42 00 64	E.hEB.d
004271C0	A1 00 00 00 00 50 64 89	i....Pd
004271C8	25 00 00 00 00 83 C4 A8	%....s-d
004271D0	53 56 57 89 65 E8 E9 8A	SUM000000
004271D8	D6 65 00 00 33 02 8A D4	ie...3000
004271E0	89 15 34 E6 45 00 8B C8	034pE.i
004271E8	81 E1 FF 00 00 00 89 0D	00...00
004271F0	30 E6 45 00 C1 E1 08 03	0pE...00
004271F8	CA 89 0D 2C E6 45 00 C1	00...pE.1

现在我们来 dump,将 OEP 指定为 4271B0。

OllyDump - UnPackMe_PELock1.06.d.exe

Start Address: 400000 Size: 75000 Dump

Entry Point: 6B05C -> Modify: 271b0 Get EIP as OEP Cancel

Base of Code: 1000 Base of Data: C

☒ Fix Raw Size & Offset of Dump Image

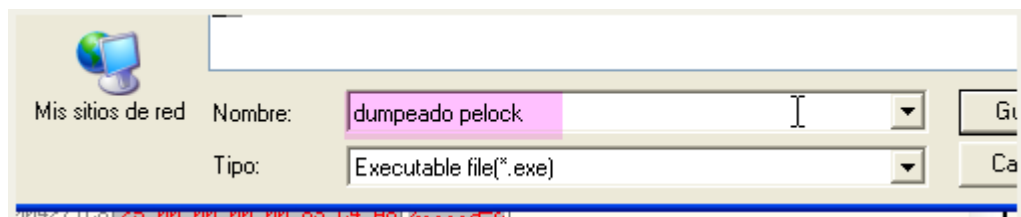
Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.teddy	0004A000	00001000	0004A000	00001000	C00000E0
.teddy	0000C000	0004B000	0000C000	0004B000	C00000E0
.teddy	00009000	00057000	00009000	00057000	C00000E0
.teddy	00003000	00060000	00003000	00060000	C00000E0
.teddy	00008000	00063000	00008000	00063000	C00000E0
.teddy	0000A000	0006B000	0000A000	0006B000	C00000E0

☐ Rebuild Import

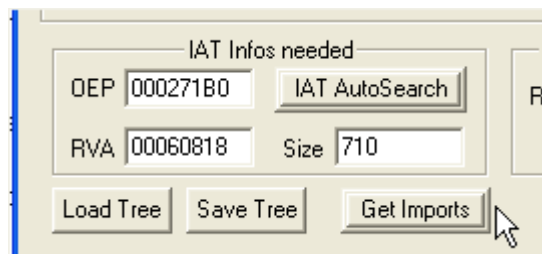
☒ Method1 : Search JMP[API] | CALL[API] in memory image

☐ Method2 : Search DLL & API name string in dumped file

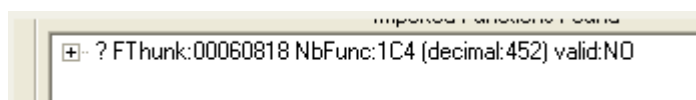
这里我们把 OEP 修正了。去掉 Rebuild Import 的对勾,我们不使用 OllyDump 的重建输入表的功能。



将 dump 出来的文件命名为 dumpeado pelock,现在我们来修复 IAT,打开 IMP REC,将 OEP 修正。



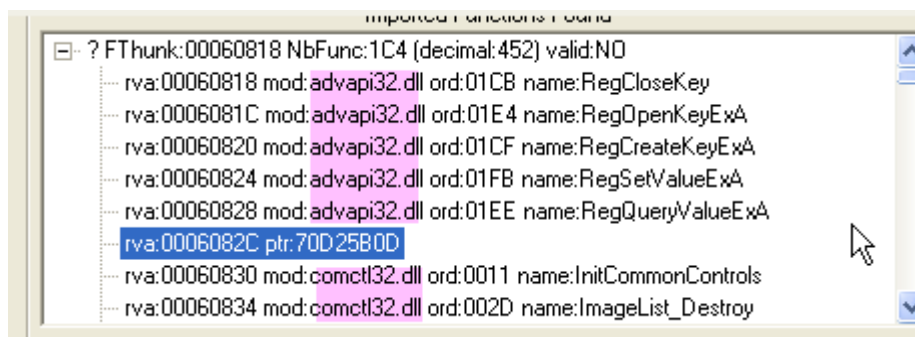
将 IAT 的信息填好,接着单击 Get Imports 按钮。



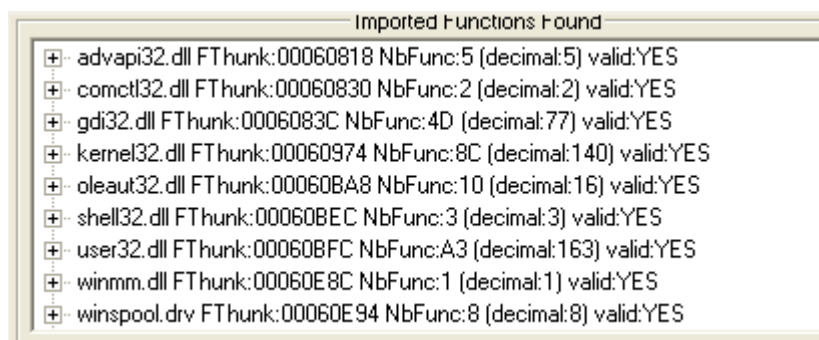
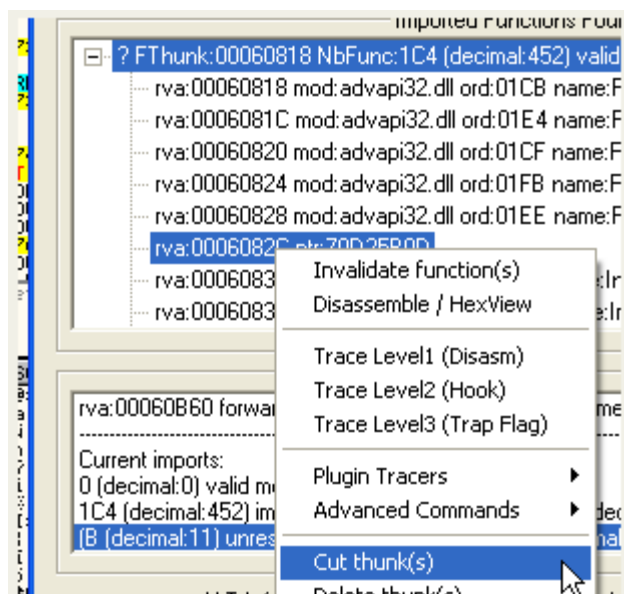
我们可以看到提示有无效的项,但是并没有经过重定向,只是每个 DLL 的 IAT 项原本应该用零间隔开的,这里壳使用垃圾数据将零覆盖掉了。我们来看一看。

Address	Hex dump	ASCII
0046091C	1A 40 F2 77 55 EA EF 77	+@=wU0'w
00460924	C5 61 EF 77 70 E6 EF 77	+a'wpp'w
0046092C	F0 81 EF 77 2D 6C EF 77	-u'w-l'w
00460934	98 6E EF 77 4F 83 EF 77	yn'w03'w
0046093C	09 ED EF 77 EB AA EF 77	.Y'w0'w
00460944	26 69 F0 77 B1 95 EF 77	&i'w0'w
0046094C	6F B0 EF 77 8A 5A EF 77	c'w0'w
00460954	E9 49 F2 77 26 F1 F0 77	0I=w0'w
0046095C	C9 DD F0 77 51 E0 F0 77	f'w0'w
00460964	33 8C EF 77 6C EC EF 77	3'w0'w
0046096C	29 94 EF 77 69 05 E6 3E)0'w0'w
00460974	68 17 80 7C C1 C9 80 7C	k'w0'w
0046097C	69 10 81 7C EE 1E 80 7C	i'w0'w
00460984	8D 2C 81 7C 40 7A 94 7C	l'w0'w
0046098C	E1 EA 81 7C A2 CA 81 7C	p'w0'w
00460994	16 1E 80 7C 43 99 80 7C	-'w0'w
0046099C	10 11 81 7C 29 29 81 7C	u'w0'w
004609A4	14 9B 80 7C 81 9A 80 7C	u'w0'w
004609AC	FB 2C 82 7C AE 94 83 7C	'e'w0'w
004609B4	2B 2E 83 7C C4 CE 80 7C	+a'w0'w
004609BC	8D 2B 86 7C 3F 0C 81 7C	a+B'w0'w

这里应该为零,用 IMP REC 修复很容易,我们单击 Show Invalid 即可。

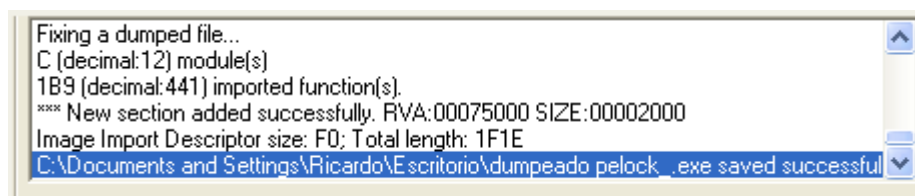


这里我们可以看到 advapi32.dll 与 comctl32.dll 的项之间被垃圾数据隔开了,选中这些无效的项,单击鼠标右键选择 Cut thunk(s),剪切掉这些无效的项。



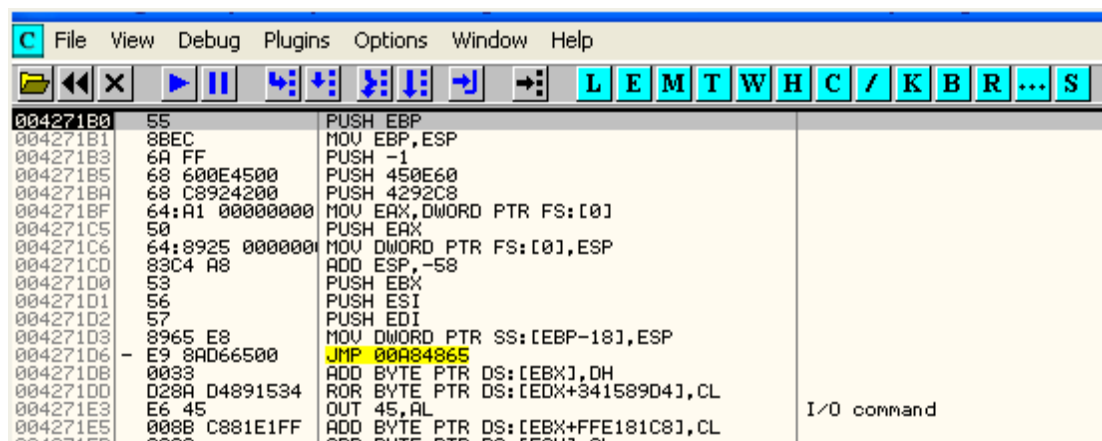
我们可以看到剪切掉这些无效的项后,显示的项都标识为有效的了。

接着我们单击 Fix Dump 按钮来修复刚刚 dump 出来的文件。



修复后的文件被重命名为了 dumpeado pelock_.exe。如果我们运行它的话,会提示错误。

下面我们需要用到 LordPe 来修复。



此时我们处在正确的入口点处,我们来看看 IAT。

Address	Hex dump	ASCII
00460818	F0 6B DA 77 1B 76 DA 77	-k r w + v r w
00460820	F4 EA DA 77 E7 EB DA 77	q u r w p u r w
00460828	83 78 DA 77 00 00 00 00	â x r w
00460830	DD 15 C5 58 2E BD C3 58	! 3 + X . c t X
00460838	00 00 00 00 04 6A EF 77 é j ' w
00460840	66 95 EF 77 89 6A EF 77	f ô ' w é j ' w
00460848	F3 AD EF 77 ED 09 EF 77	% ð ' w y ' w
00460850	99 8B EF 77 C0 B5 EF 77	ô i ' w ' A ' w
00460858	2A 7D EF 77 B2 7C EF 77	* j ' w 2 ' w
00460860	77 53 F2 77 1E C9 F1 77	w S = w A f ' w
00460868	0C BC EF 77 52 D4 EF 77	. ' w R é ' w
00460870	FA 8D EF 77 F1 DD EF 77	. i ' w t ! ' w
00460878	51 B2 EF 77 26 D5 EF 77	C 2 ' w & ' w
00460880	2A E3 EF 77 5F 39 F2 77	* ð ' w _ 9 = w
00460888	71 B4 EF 77 2E AD EF 77	q i ' w . ð ' w
00460890	E1 61 EF 77 B8 85 EF 77	p a ' w 0 a ' w
00460898	CC D2 EF 77 43 70 EF 77	f f é ' w C p ' w
004608A0	FB EA F0 77 12 83 EF 77	' u - w 3 a ' w
004608A8	01 72 F0 77 A9 34 F0 77	0 r - w 0 4 - w
004608B0	05 93 EF 77 68 EF EF 77	' o ' w h ' ' w
004608B8	AA D2 EF 77 B2 6F EF 77	- é ' w 2 o ' w
004608C0	3F 38 F2 77 D6 E8 EF 77	? 8 = w i p ' w
004608C8	68 E0 EF 77 00 60 EF 77	h ó ' w . ' w
004608D0	90 5B EF 77 6D AC EF 77	é [' w m % ' w
004608D8	94 6C F0 77 22 8D EF 77	ô l - w ' i ' w
004608E0	3D C8 F1 77 3D 6D F0 77	= é : w = m - w
004608E8	6F C0 EF 77 85 7B EF 77	o l ' w a f ' w
004608F0	26 D9 EF 77 F8 5E EF 77	& j ' w i ^ ' w
004608F8	36 8A EF 77 FC 8A EF 77	6 é ' w f é ' w
00460900	0F 62 EF 77 49 5E EF 77	* b ' w i ^ ' w
00460908	97 5D EF 77 1A 9A EF 77	ù j ' w + ü ' w
00460910	6B FA EF 77 7B C9 F0 77	k . ' w l f - w
00460918	00 98 F2 77 10 40 F2 77	- ü = w 0 0 = w

我们可以看到这里 IMP REC 已经把 IAT 都修复了,各个 DLL 中的 IAT 项中间的间隔也由垃圾数据替换为了零。但是为什么运行起来还报错呢?

这就是我们接下来要讨论的一个话题,在修复了 IAT,以及 stolen bytes 以后,很多壳还会有 AntiDump。

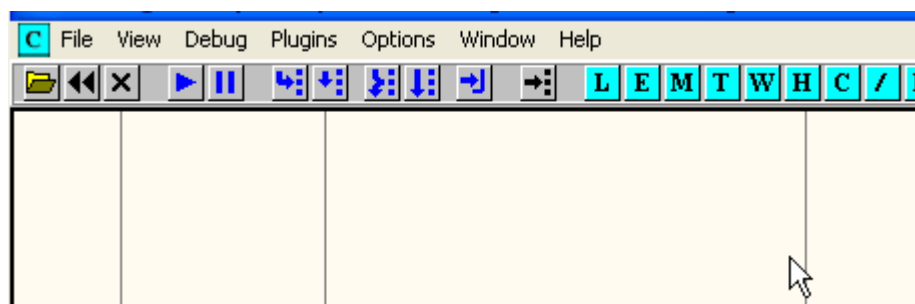
有很多类型的 AntiDump,简单一点的 AntiDump 会校验映像的大小或者区段的个数,我们这个例子的 AntiDump 是壳创建了几个区段,在运行时会对原程序做一些处理,如果 dump 出来的程序不包含这些区段,那么程序就不能正常运行,我们来到入口点处。

L E M T W H C / K B R			
004271B0	55	PUSH EBP	
004271B1	8BEC	MOV EBP,ESP	
004271B3	6A FF	PUSH -1	
004271B5	68 600E4500	PUSH 450E60	
004271B8	68 C8924200	PUSH 4292C8	
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
004271C5	50	PUSH EAX	
004271C6	64:8925 00000000	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	- E9 8AD66500	JMP 00A84865	
004271D8	0033	ADD BYTE PTR DS:[EBX],DH	
004271DD	D28A D4891534	ROR BYTE PTR DS:[EDX+341589D4],CL	
004271E3	E6 45	OUT 45,AL	I/O command
004271F5	0000 0001F1FF	ADD BYTE PTR DS:[EBX+FFF181C8],CL	

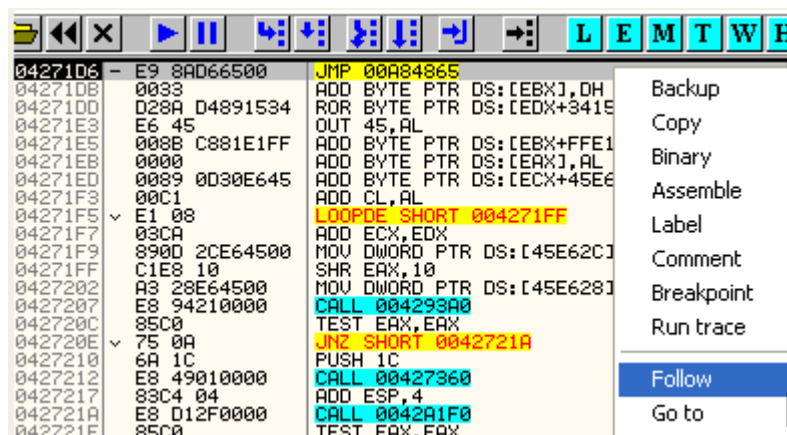
当程序运行到 4271D6 处的 JMP 指令处的时候就会报错。

L E M T W H C / K B R			
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	- E9 8AD66500	JMP 00A84865	
004271D8	0033	ADD BYTE PTR DS:[EBX],DH	
004271DD	D28A D4891534	ROR BYTE PTR DS:[EDX+341589D4],CL	
004271E3	E6 45	OUT 45,AL	I/O command
004271F5	0000 0001F1FF	ADD BYTE PTR DS:[EBX+FFF181C8],CL	

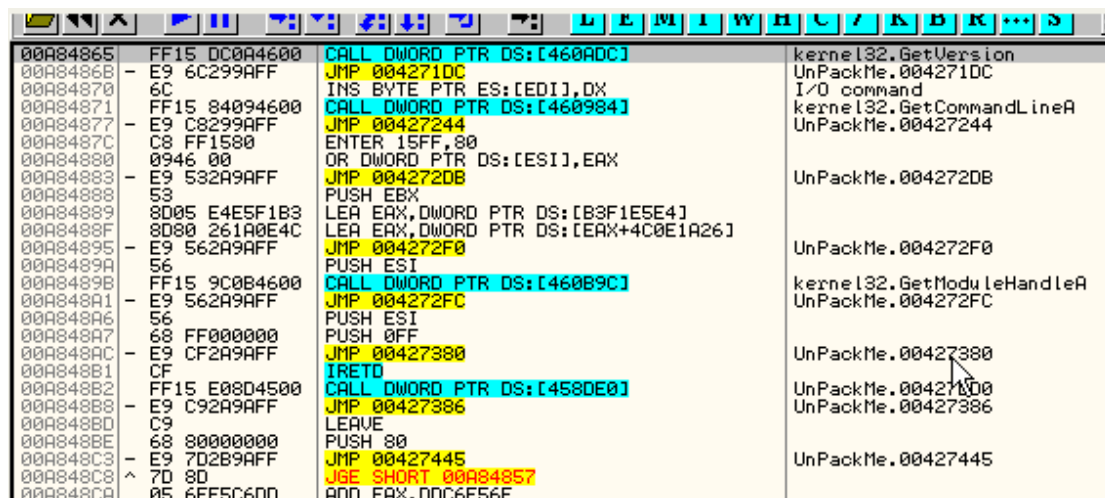
如果我们按 F7 键,会跳转到一个不存在的区段。



我们来看一看未脱壳的程序,会发现将跳转到壳运行时创建的一个区段中。



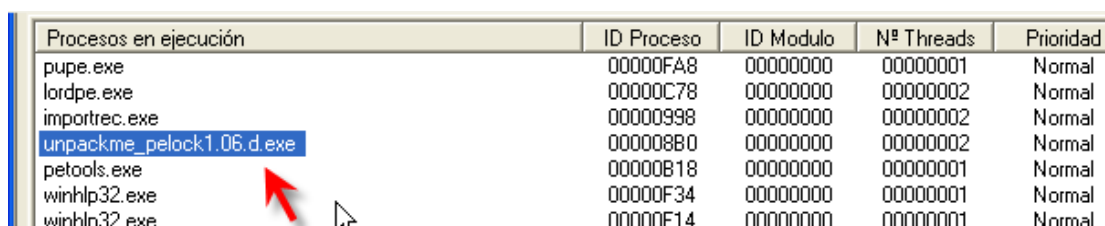
单击鼠标右键选择 Follow,就能够跟随到壳创建的区段中。



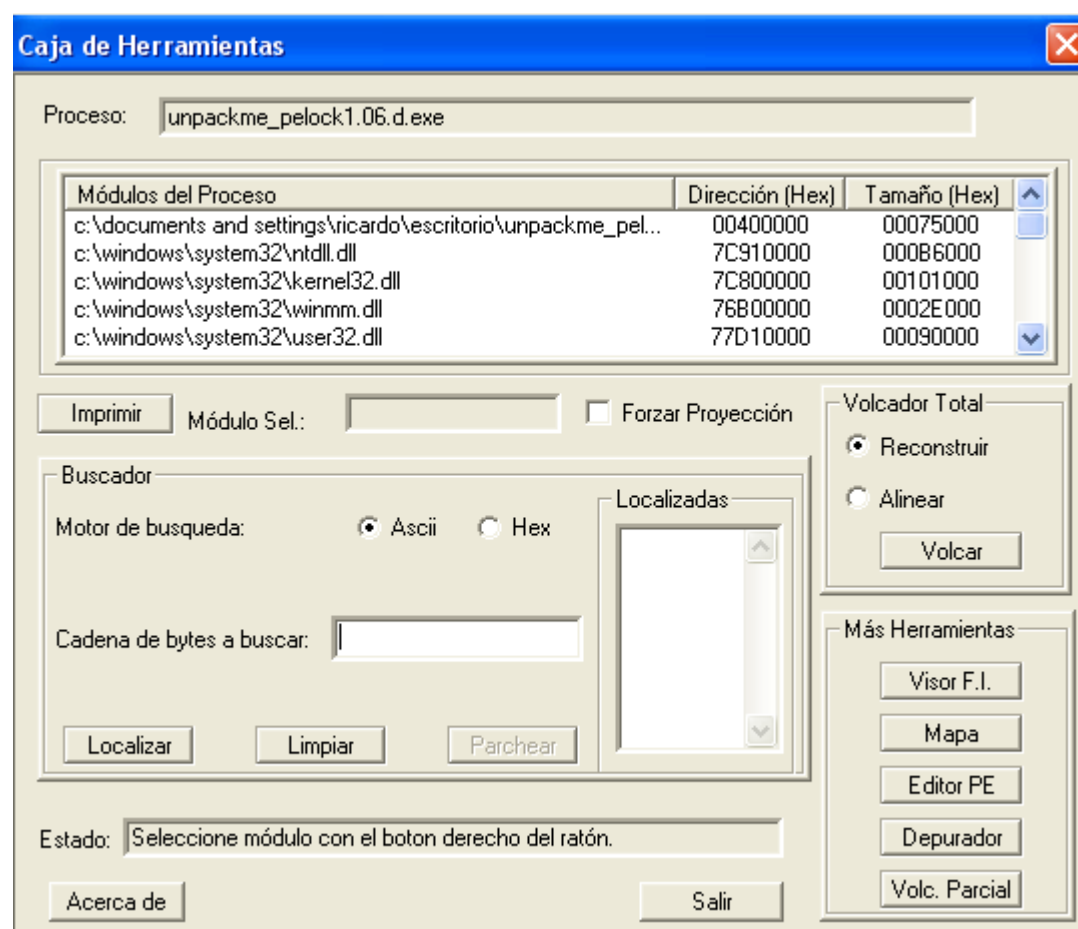
这里我们可以看到是调用了 API 函数,接着返回到主程序代码段,有很多方法可以解决这个问题,最简单的方法就是给 dump 出来的文件添加一个相同的区段,我们来看看。

首先添加报错的这个区段,如果添加了以后还报错,再次添加其他缺失的区段。

这里我们需要用到 PUPE 这款工具来 dump 缺失的区段。



选中 Pelock 所在的进程,单击鼠标右键选择 BOX OF TOOLS。



单击 Mapa(即 MAP)按钮打开区段列表窗口,定位到报错的区段。

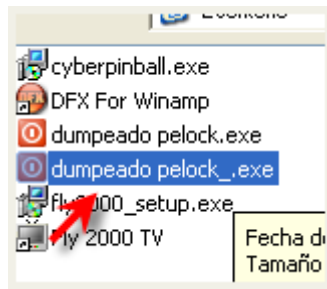
我们还记得报错的区段的起始地址为 A80000,我们一起来看看。



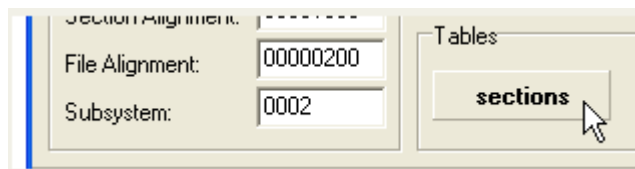
我们单击 Volcar(PS:西班牙语译为 DUMP)按钮,将其命名为 seccion(PS:西班牙语译为 section)。



现在我们打开 PEEditor,将该区段添加到 dump 文件中。



通过 PEEditor 打开 dump 出来的文件。



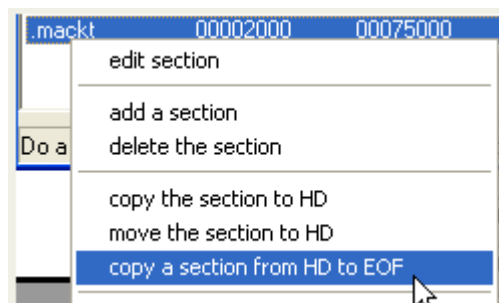
单击 sections 按钮。

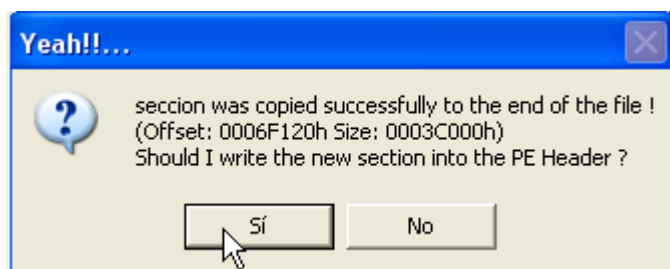
Section Table Viewer						
Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics	
.teddy	0004A000	00001000	00049183	00000400	C00000E0	
.teddy	0000C000	0004B000	0000BEDB	00049600	C00000E0	
.teddy	00009000	00057000	000056E0	00055600	C00000E0	
.teddy	00003000	00060000	000010DB	0005AE00	C00000E0	
.teddy	00008000	00063000	0000789D	0005C000	C00000E0	
.teddy	0000A000	0006B000	000096F8	00063A00	C00000E0	
.mackt	00002000	00075000	00001F20	0006D200	E0000060	

Do a right mouse click on a sectionname for more options...

这里我们可以看到 IMP REC 在为了修复 IAT 也给该程序添加了一个区段,名称为.mackt,我们将缺失的区段添加到最后。

在.mackt 这个区段上面单击鼠标右键选择 copy a section from HD to EOF。接着我们将新添加区段的起始地址修改为 A80000。



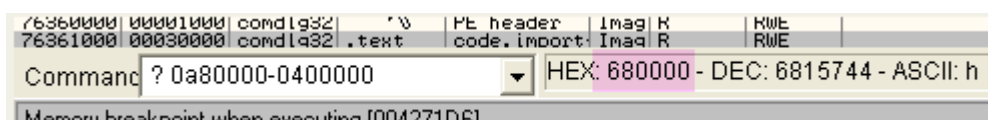


Section Table Viewer

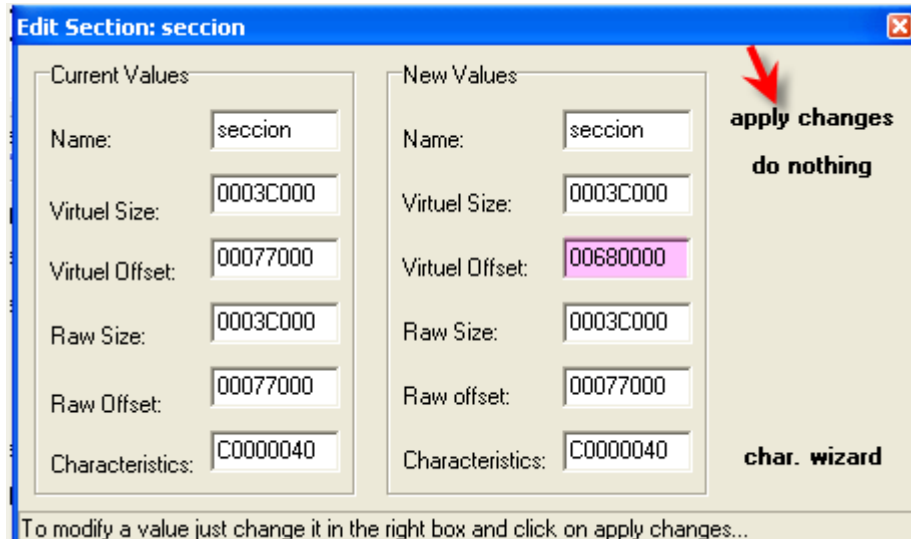
Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.teddy	0004A000	00001000	00049183	00000400	C00000E0
.teddy	0000C000	0004B000	0000BEDB	00049600	C00000E0
.teddy	00009000	00057000	000056E0	00055600	C00000E0
.teddy	00003000	00060000	000010DB	0005AE00	C00000E0
.teddy	00008000	00063000	0000789D	0005C000	C00000E0
.teddy	0000A000	0006B000	000096F8	00063A00	C00000E0
.mactt	00002000	00075000	00001F20	0006D200	E0000060
seccion	0003C000	00077000	0003C000	0006F120	C0000040

Do a right mouse click on a sectionname for more options...

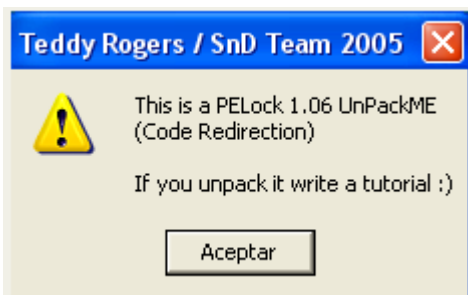
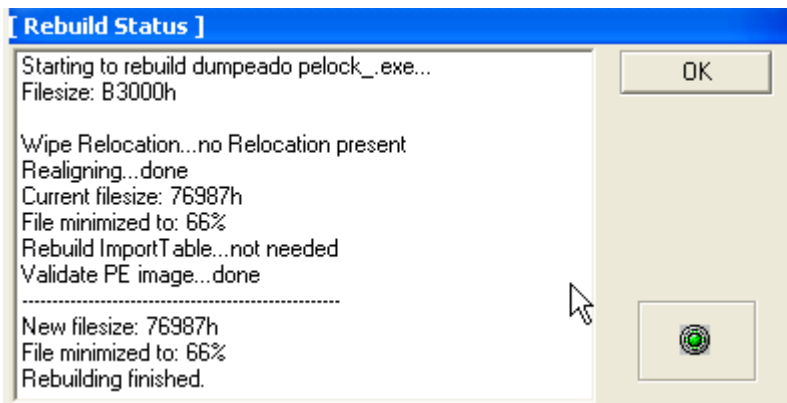
我们来计算一下该新区段的相对虚拟地址 = $A80000 - 400000 = 680000$ 。



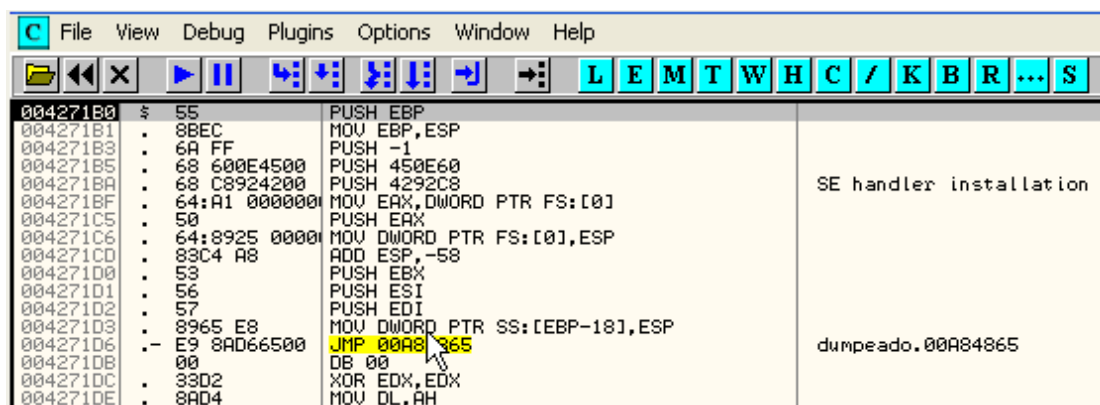
也就是说新区段的起始地址 RVA = 680000。



下面我们通过 LordPe 来重建该 PE 就大功告成了。



正常运行,我们用 OllyDbg 打开它。



查看一下区段列表。

00340000	00001000				Priv	RW	RW	
003B0000	00001000				Priv	RW	RW	
003C0000	00004000				Priv	RW	RW	
003D0000	00004000				Priv	RW	RW	
003E0000	00003000				Map	R	R	\Device\Harddi
003F0000	00002000				Map	R	R	
00400000	00001000	dumpeado		PE header	Imag	R	RWE	
00401000	0004A000	dumpeado	.teddy	code	Imag	R	RWE	
0044B000	0000C000	dumpeado	.teddy		Imag	R	RWE	
00457000	00009000	dumpeado	.teddy		Imag	R	RWE	
00460000	00003000	dumpeado	.teddy		Imag	R	RWE	
00463000	00008000	dumpeado	.teddy	resources	Imag	R	RWE	
0046B000	0000A000	dumpeado	.teddy		Imag	R	RWE	
00475000	0000B000	dumpeado	.mackt	imports	Imag	R	RWE	
00A80000	0000C000	dumpeado	seccion	data	Imag	R	RWE	
00C00000	0000B000				Map	R E	R E	
00E00000	00002000				Map	R E	R E	

我们可以看到起始地址为 A80000 新添加的区段。

004271B5	. 54:41 000000	MOV EAX,DWORD PTR FS:[0]	
004271C5	. 50	PUSH EAX	
004271C6	. 64:8925 0000	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	- E9 8AD66500	JMP 00A84865	
004271D8	. 00	DB 00	
004271DC	. 33D2	XOR EDX,EDX	
004271DE	. 8AD4	MOV DL,AH	
004271E0	. 8915 34E6450	MOV DWORD PTR DS:[45E6],EAX	
004271E6	. 8BC8	MOV ECX,EAX	
004271E8	. 81E1 FF000000	AND ECX,0FF	
004271EE	. 8900 30E6450	MOV DWORD PTR DS:[45E6],ECX	
004271F4	. C1E1 08	SHL ECX,8	
004271F7	. 03CA	ADD ECX,EDX	
004271F9	. 8900 2CE6450	MOV DWORD PTR DS:[45E6],ECX	
004271FF	. C1E8 10	SHR EAX,10	
00427202	. A3 28E64500	MOV DWORD PTR DS:[45E6],EAX	
00427207	. E8 94210000	CALL 004293A0	
0042720C	. 85C0	TEST EAX,EAX	
0042720E	. 75 0A	JNZ SHORT 0042721A	
00427210	. 6A 1C	PUSH 1C	
00427212	. 50	PUSH EAX	
00A84865=dumpeado.00A84865			
Address	Hex dump	ASCII	

- Backup
- Copy
- Binary
- Assemble
- Label
- Comment
- Breakpoint
- Hit trace
- Run trace
- Follow
- New origin here
- Go to

我们定位到 4271D6 地址处 JMP 指令,单击鼠标右键选择 Follow。

00A84865	FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]	kernel32.GetVersion
00A8486B	- E9 6C299AFF	JMP 004271DC	dumpeado.004271DC
00A84870	. 6C	INS BYTE PTR ES:[EDI],DX	I/O command
00A84871	FF15 84094600	CALL DWORD PTR DS:[460984]	kernel32.GetCommandLineA
00A84877	- E9 C8299AFF	JMP 00427244	dumpeado.00427244
00A8487C	. C8 FF1580	ENTER 15FF,80	
00A84880	. 0946 00	OR DWORD PTR DS:[ESI],EAX	
00A84883	- E9 532A9AFF	JMP 004272D8	dumpeado.004272D8
00A84888	. 53	PUSH EBX	
00A84889	. 8D05 E4E5F1B3	LEA EAX,DWORD PTR DS:[B3F1E5E4]	
00A8488F	. 8D08 261A0E4C	LEA EAX,DWORD PTR DS:[EAX+4C0E1A26]	
00A84895	- E9 562A9AFF	JMP 004272F0	dumpeado.004272F0
00A8489A	. 56	PUSH ESI	
00A8489B	FF15 9C0B4600	CALL DWORD PTR DS:[460B9C]	kernel32.GetModuleHandleA
00A848A1	- E9 562A9AFF	JMP 004272FC	dumpeado.004272FC
00A848A6	. 56	PUSH ESI	
00A848A7	. 68 FF000000	PUSH 0FF	

跳转到了缺失的区段中。

我们来对比一下未脱壳和脱完壳并修复后的区段。

未脱壳的区段:

003F0000	00001000				Priv	RW	RW	
003F0000	00001000	UnPackMe		PE header	Priv	RW	RW	
00400000	00001000	UnPackMe		code	Imag	R	RWE	
00401000	0004A000	UnPackMe	.teddy		Imag	R	RWE	
0044B000	0000C000	UnPackMe	.teddy		Imag	R	RWE	
00457000	00009000	UnPackMe	.teddy		Imag	R	RWE	
00460000	00003000	UnPackMe	.teddy		Imag	R	RWE	
00463000	00008000	UnPackMe	.teddy	resources	Imag	R	RWE	
0046B000	0000A000	UnPackMe	.teddy	SFX, imports	Imag	R	RWE	
00480000	00021000				Priv	RW	RW	
0048B000	0000B000				Map	R E	R E	
00570000	00002000				Map	R E	R E	
00580000	00103000				Map	R	R	
00690000	00001000				Priv	RW	RW	
006A0000	00176000				Map	R E	R E	
009A0000	00001000				Priv	RW	RW	
009B0000	00004000				Priv	RW	RW	
009C0000	00003000				Map	R	R	
009D0000	00001000				Priv	RW	RW	
00A50000	00004000				Priv	RW	RW	
00A60000	00003000				Priv	RW	RW	
00A70000	00002000				Map	R	R	
00A80000	0003C000				Priv	RW	RW	
00A90000	00001000				Priv	RW	RW	
00B00000	00002000				Priv	RW	RW	

脱完壳修复后的区段:

003F0000	00002000	dumpeado			Map	R	R	
00400000	00001000	dumpeado		PE header	Imag	R	RWE	
00401000	0004A000	dumpeado	.teddy	code	Imag	R	RWE	
0044B000	0000C000	dumpeado	.teddy		Imag	R	RWE	
00457000	00009000	dumpeado	.teddy		Imag	R	RWE	
00460000	00003000	dumpeado	.teddy		Imag	R	RWE	
00463000	00008000	dumpeado	.teddy	resources	Imag	R	RWE	
0046B000	0000A000	dumpeado	.teddy		Imag	R	RWE	
00475000	0060B000	dumpeado	.mact	imports	Imag	R	RWE	
00A80000	0003C000	dumpeado	section	data	Imag	R	RWE	
00AC0000	0000B000				Map	R E	R E	
00B80000	00002000				Map	R E	R E	

本章就到这里。(PS:后面一点总结部分,作者讲的有误,这里就略去了)