

论 IAT 重定向之修复

本章我将给大家介绍几种常见的修复 IAT 重定向的方法。

我们只是介绍修复 IAT 重定向的基本思路,有些方法可能对有个壳有效,对有的壳无效,这个就需要大家多加练习,举一反三,触类旁通了。

```
File View Debug Plugins Options Window Help
[Icons] [L] [E] [M] [T] [W] [H] [C] [K] [B] [R] [S] [Icons]
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 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 FF15 DC0A4600 CALL DWORD PTR DS:[460ADC]
004271DC 33D2 XOR EDX,EDX
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
EBP=0012FFF0
Real entry point of SFX code
```

我们用 OD 加载 UnPackMe_tElock0.98,按照上一章节介绍的方法定位到其 OEP,这里就不再赘述了。IAT 中有部分元素被重定向到壳创建的区段中去了。

Address	Hex dump	ASCII
0046080C	00 00 00 00 00 00 00 80 00 00 00 00 F0 6B DA 77C....-krw
0046081C	1B 76 DA 77 F4 EA DA 77 E7 EB DA 77 83 78 DA 77	+Orwq0 rwb0 rwbx rw
0046082C	00 00 00 00 DD 15 C5 58 2E BD C3 58 00 00 00 00	...!s+X.c+X....
0046083C	00 00 A1 00 11 00 A1 00 22 00 A1 00 33 00 A1 00	...l.4.l...i.3.i.
0046084C	41 00 A1 00 50 00 A1 00 5F 00 A1 00 7F 00 A1 00	A.l.P.l...i.d.i.
0046085C	8D 00 A1 00 B0 00 A1 00 C1 00 A1 00 D2 00 A1 00	i.l.:.i.i.i.e.i.
0046086C	E4 00 A1 00 F4 00 A1 00 05 01 A1 00 24 01 A1 00	s.l.4.l.i.+0i.s0i.
0046087C	3E 01 A1 00 4F 01 A1 00 60 01 A1 00 71 01 A1 00	>0i.00i.'0i.q0i.
0046088C	7F 01 A1 00 8E 01 A1 00 90 01 A1 00 B0 01 A1 00	00i.A0i.00i.c0i.
0046089C	CB 01 A1 00 EE 01 A1 00 FF 01 A1 00 10 02 A1 00	r0i.'0i. 0i.b0i.
004608AC	22 02 A1 00 32 02 A1 00 43 02 A1 00 62 02 A1 00	"0i.20i.C0i.b0i.
004608BC	7C 02 A1 00 8D 02 A1 00 9E 02 A1 00 AF 02 A1 00	!0i.i0i.x0i.>0i.
004608CC	BD 02 A1 00 CC 02 A1 00 DB 02 A1 00 FB 02 A1 00	c0i.l0i.00i.'0i.
004608DC	09 03 A1 00 2C 03 A1 00 3D 03 A1 00 4E 03 A1 00	.0i..0i.=0i.N0i.
004608EC	60 03 A1 00 70 03 A1 00 81 03 A1 00 9A 03 A1 00	'0i.p0i.u0i.s0i.
004608FC	BA 03 A1 00 CB 03 A1 00 DC 03 A1 00 ED 03 A1 00	0i.r0i.00i.Y0i.
0046090C	FB 03 A1 00 0A 04 A1 00 19 04 A1 00 39 04 A1 00	'0i..0i..0i.90i.
0046091C	47 04 A1 00 6A 04 A1 00 7B 04 A1 00 8C 04 A1 00	G0i.j0i.C0i.i0i.
0046092C	9E 04 A1 00 AE 04 A1 00 BF 04 A1 00 DE 04 A1 00	x0i.<0i.r0i.i0i.
0046093C	F8 04 A1 00 09 05 A1 00 1A 05 A1 00 2B 05 A1 00	o0i..0i..0i.+0i.
0046094C	39 05 A1 00 48 05 A1 00 57 05 A1 00 77 05 A1 00	90i.H0i.W0i.w0i.
0046095C	85 05 A1 00 A8 05 A1 00 B9 05 A1 00 CA 05 A1 00	s0i.d0i.l0i.s0i.
0046096C	DC 05 A1 00 EC 05 A1 00 00 00 9F 00 11 00 9F 00	00i.g0i...f.4.f.
0046097C	22 00 9F 00 33 00 9F 00 41 00 9F 00 50 00 9F 00	"..f.3.f.A.f.P.f.
0046098C	5F 00 9F 00 7F 00 9F 00 8D 00 9F 00 B0 00 9F 00	..f.d.f.i.f.s.f.
0046099C	C1 00 9F 00 D2 00 9F 00 E4 00 9F 00 F4 00 9F 00	..f.e.f.s.f.4.f.

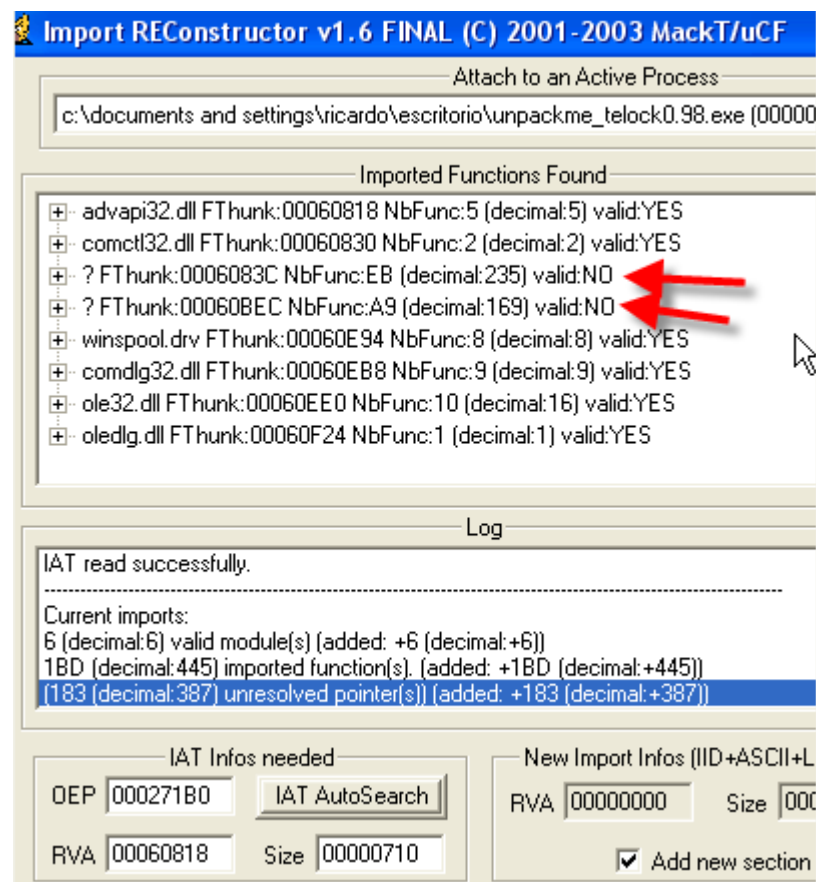
这些重定向过的 IAT 项我们需要修复。

我们首先单步到第一个 API 函数 GetVersion 的调用处。

004271D1 56 PUSH ESI	
004271D2 57 PUSH EDI	
004271D3 8965 E8 MOV DWORD PTR SS:[EBP-18],ESP	
004271D6 FF15 DC0A4600 CALL DWORD PTR DS:[460ADC]	
004271DC 33D2 XOR EDX,EDX	
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	
DS:[00460ADC]=009F06F7	

Address	Hex dump	ASCII
00460ADC	F7 06 9F 00 08 07 9F 00 1A 07 9F 00 2A 07 9F 00	..f.0.f..f.*.f.
00460AEC	3B 07 9F 00 CA 07 9F 00 74 07 9F 00 85 07 9F 00	;.f.2.f.t.f.s.f.
00460AFC	96 07 9F 00 07 07 9F 00 B5 07 9F 00 C4 07 9F 00	U.f.9.f.A.f.-.f.
00460B0C	D3 07 9F 00 F3 07 9F 00 01 08 9F 00 24 08 9F 00	E.f.%f.0f.s0f.
00460B1C	35 08 9F 00 46 08 9F 00 58 08 9F 00 68 08 9F 00	S0f.F0f.X0f.h0f.

在数据窗口中定位到该 IAT 项 460ADC,这里 460ADC 保存的内容是 9F06F7,我们打开 IMP REC,填充上 IAT 起始地址的 RVA,大小以及 OEP 的 RVA 后,看一看无效的项。



我们可以看到 9F06F7 这一项是无效的。

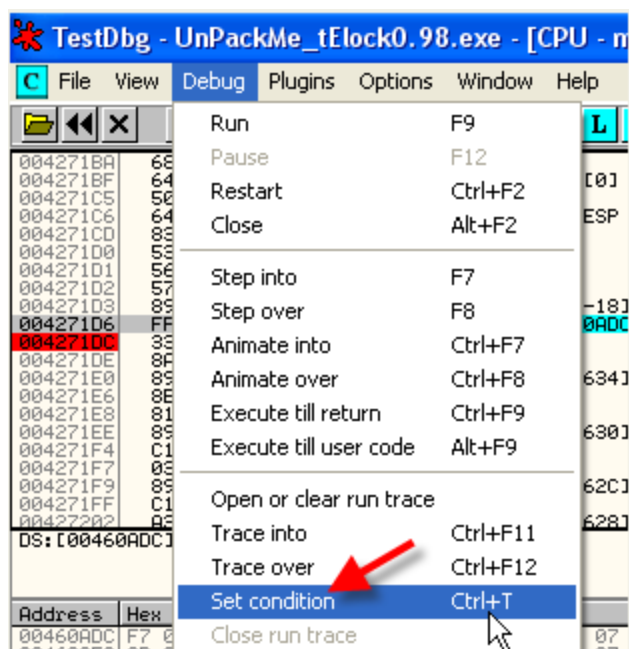


RVA 60ADC 对应的 IAT 项正好是 460ADC。

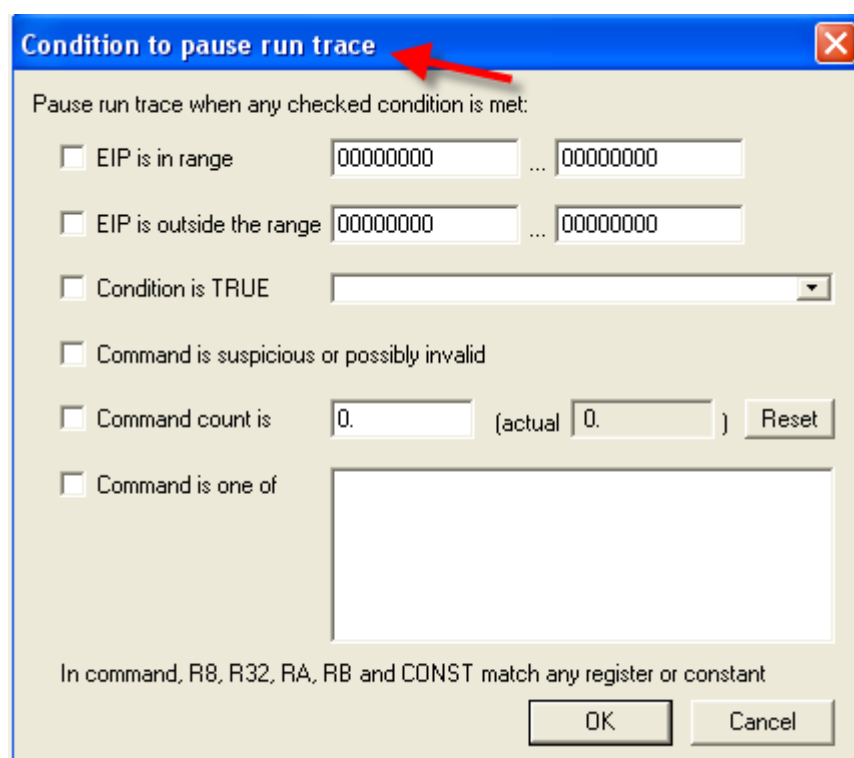
我们回到 OD 中,上一章我们已经介绍过了如何定位 GetVersion 这个 API 函数的入口地址,其实 OD 有一个自动单步跟踪的功能,我们用这个自动单步跟踪的功能可以方便定位到重定向过的 IAT 项对应的 API 函数入口地址,重定向过的 IAT 项单步步入 5,6 行就可以定位到其对应的 API 函数。

00427101	56	PUSH ESI
00427102	57	PUSH EDI
00427103	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
00427106	FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]
0042710C	33D2	XOR EDX,EDX
0042710E	8AD4	MOV DL,AH
0042710A	8915 34F64500	MOV DWORD PTR DS:[45F6341],EDX

我们在 4271D6 这条 CALL 指令的返回地址处设置一个断点,这里我们缺少一步,由于 OD 的自动跟踪功能是不会主动停止的,我们应该设置自动跟踪停止的条件。



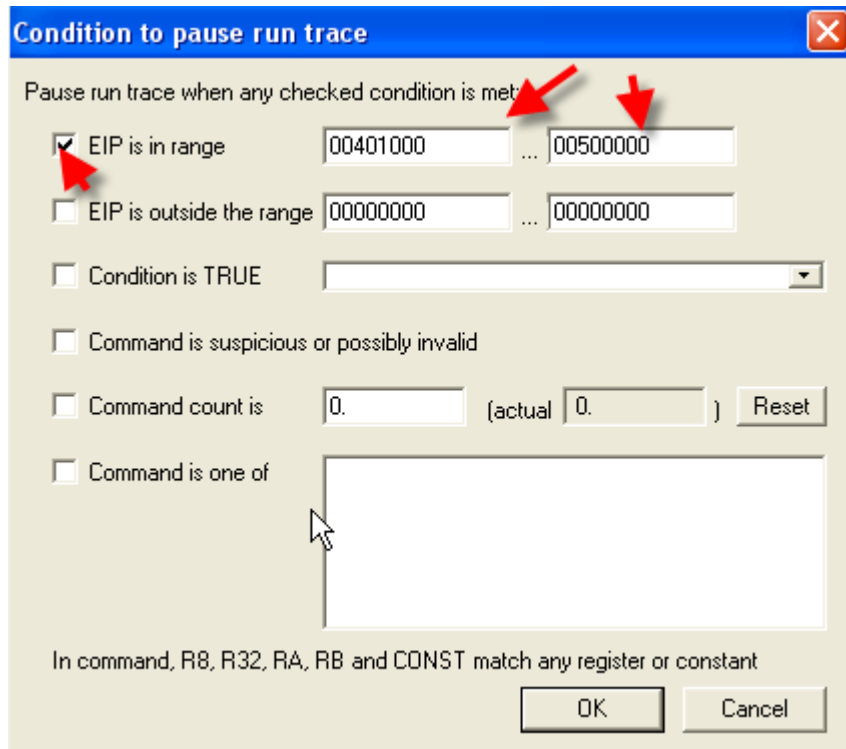
我们选择主菜单中 Debug-Set condition,设置跟踪停止的条件。



我们可以看到弹出了一个对话框,对话框的标题为 Condition to pause run trace(设置自动跟踪终止的条件)。

如果设置了 EIP is in range(EIP 的范围)这个选项的话,当 EIP 位于这个范围的时候,OD 就会停止自动跟踪。

例如:

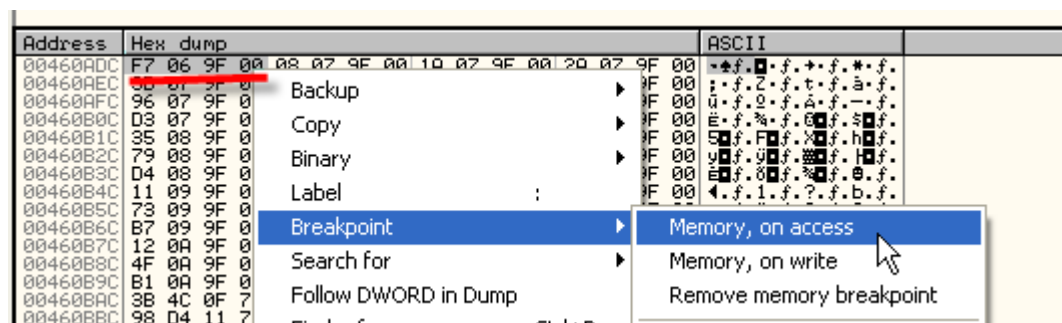


例如,这里我们将这个范围指定为 401000 ~ 500000,那么 OD 自动跟踪,当 EIP 在 401000 ~ 500000 这个范围的时候就会停止自动跟踪,这里我们只是举个例子,并不是真的要将范围指定为 401000 ~ 500000, 因为我们需要 OD 自动跟踪停在 API 函数的入口处,所以我们打开区段列表窗口看看将这个范围指定为什么比较合适。

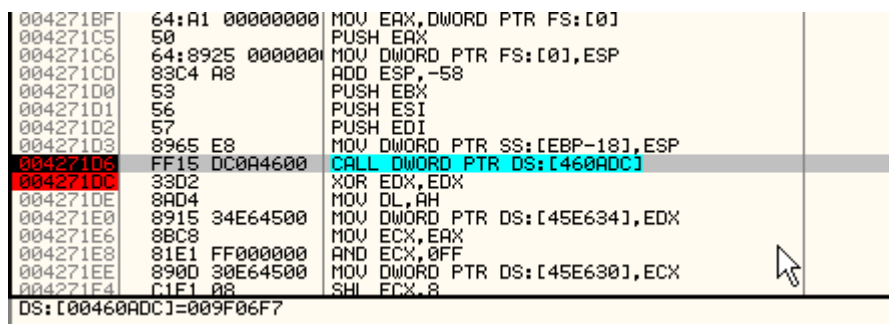
00320000	00000000			Map	R	R	\Device\HarddiskVolume1\Win
00330000	00041000			Map	R	R	
00340000	00001000			Priv	RWE	RWE	
00350000	00001000			Priv	RWE	RWE	
00360000	00001000			Priv	RW	RW	
00370000	00001000			Priv	RW	RW	
00400000	00001000	UnPackMe		Image	RW	RWE	
00410000	00040000	UnPackMe	.teddy	Image	RW	RWE	
00440000	00000000	UnPackMe	.teddy	Image	RW	RWE	
00457000	00009000	UnPackMe	.teddy	Image	RW	RWE	
00460000	00003000	UnPackMe	.teddy	Image	RW	RWE	
00463000	00002000	UnPackMe	.rsrc	Image	RW	RWE	
00465000	00004000	UnPackMe	.teddy	SFX, imports	Image	RW	
00470000	00005000			Map	R E	R E	
00530000	00002000			Map	R E	R E	
00540000	00103000			Map	R E	R E	
00650000	000EE000			Map	R E	R E	
00970000	00001000			Priv	RW	RW	
009F0000	00002000			Priv	RW	RW	
00A00000	00002000			Priv	RW	RW	
00A10000	00001000			Priv	RW	RW	
00A20000	00004000			Priv	RW	RW	
00A30000	00003000			Map	R	R	
00A40000	00004000			Priv	RW	RW	
00A50000	00003000			Priv	RW	RW	
00A60000	00002000			Map	R	R	
00A70000	00001000			Priv	RW	RW	
01270000	00002000			Map	R	R	
58C30000	00001000	CONCTL32		PE header	Image	R	RWE
58C31000	00070000	CONCTL32	.text	code, import	Image	R	RWE
58C31000	00003000	CONCTL32	.data	data	Image	R	RWE
58C34000	0001F000	CONCTL32	.rsrc	resources	Image	R	RWE
58C35000	00004000	CONCTL32	.reloc	relocations	Image	R	RWE
58480000	00001000	umdmxfrm		PE header	Image	R	RWE
58481000	00003000	umdmxfrm	.text	code, import	Image	R	RWE
58484000	00001000	umdmxfrm	.data	data	Image	R	RWE
58485000	00001000	umdmxfrm	.rsrc	resources	Image	R	RWE
58486000	00001000	umdmxfrm	.reloc	relocations	Image	R	RWE
5D160000	00001000	serwudrv		PE header	Image	R	RWE
5D161000	00003000	serwudrv	.text	code, import	Image	R	RWE
5D164000	00001000	serwudrv	.data	data	Image	R	RWE
5D165000	00001000	serwudrv	.rsrc	resources	Image	R	RWE
5D166000	00001000	serwudrv	.reloc	relocations	Image	R	RWE
72F80000	00001000	WINSPPOOL		PE header	Image	R	RWE
72F81000	00002000	WINSPPOOL	.text	code, import	Image	R	RWE
72F81000	00002000	WINSPPOOL	.data	data	Image	R	RWE
72F83000	00001000	WINSPPOOL	.rsrc	resources	Image	R	RWE
72F84000	00002000	WINSPPOOL	.reloc	relocations	Image	R	RWE
74CC0000	00001000	oledlg		PE header	Image	R	RWE
74CC1000	00011000	oledlg	.text	code, import	Image	R	RWE
74CC2000	00002000	oledlg	.data	data	Image	R	RWE
74CD4000	0000B000	oledlg	.rsrc	resources	Image	R	RWE
74CDF000	00001000	oledlg	.reloc	relocations	Image	R	RWE
76360000	00001000	comdlg32		PE header	Image	R	RWE
76361000	00030000	comdlg32	.text	code, import	Image	R	RWE
76391000	00004000	comdlg32	.data	data	Image	R	RWE
76395000	00012000	comdlg32	.rsrc	resources	Image	R	RWE
763A7000	00003000	comdlg32	.reloc	relocations	Image	R	RWE
76B00000	00001000	WINMM		PE header	Image	R	RWE
76B01000	0001F000	WINMM	.text	code, import	Image	R	RWE
76B20000	00002000	WINMM	.data	data	Image	R	RWE
76B22000	0000B000	WINMM	.rsrc	resources	Image	R	RWE

我们看到区段列表窗口中用浅蓝色标注出来的部分,这些是系统 DLL 的区段,当 EIP 位于主程序的区段或者壳创建的区段中,我们让 OD 自动继续跟踪,OD 跟进到 DLL 中的时候,我们需要 OD 停止跟踪,所以这里我们将该范围设置为 58C30000 ~ 7FFFFFFF,这样就能确保 OD 跟进到任意一个系统 DLL 中的时候就会停止跟踪,这里我并没有精确的设置为 58C30000 ~ 7FFFFFFF,大家想设置的精确一点也是可以的。

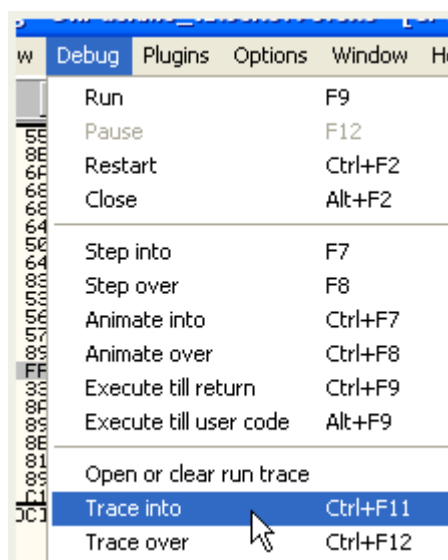
读取的时候,OD 就会断下来。



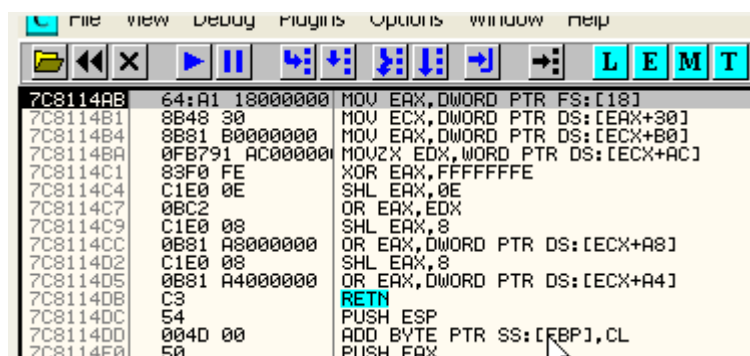
我们现在跟到了 4271D6 的 CALL 指令处。



现在可以让 OD 自动跟踪了。



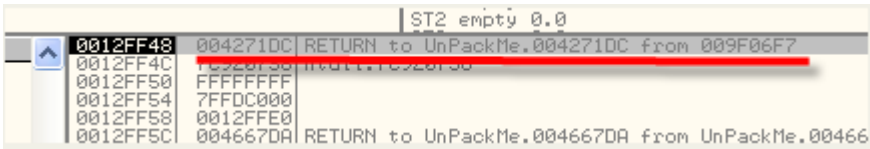
这里我们在反汇编窗口中单击鼠标右键,有两个自动跟踪的选项,第一个是 Trace into(自动单步入),第二个是 Trace over(自动单步步过),这里我们选择-Trace into。



这里满足了停止跟踪的条件,OD 停了下来。



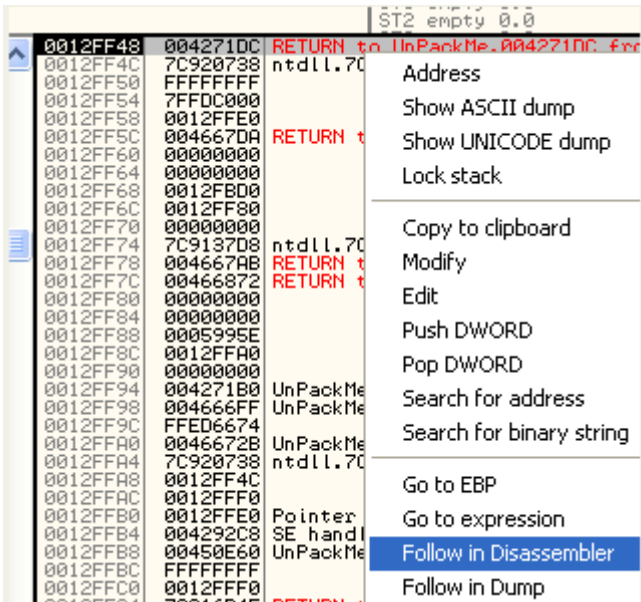
当 EIP 位于 58000000~7FFFFFFF 这个范围的时候,OD 就会断下来,为了保险起见,我们还是需要看一下 OD 的下方的提示信息,看看是不是由于其他异常原因导致断下来的。OD 下面的提示信息显示是由于 EIP 位于 58000000~7FFFFFFF 的范围内,满足了停止自动跟踪的条件,所以断了下来。



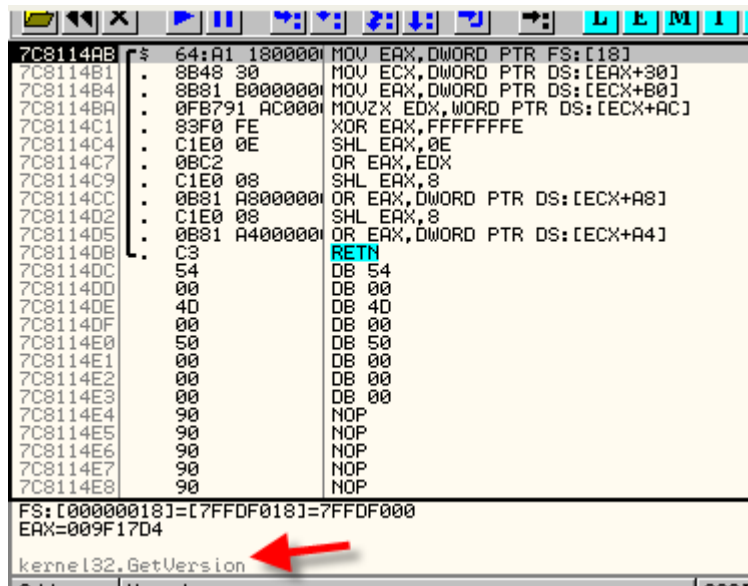
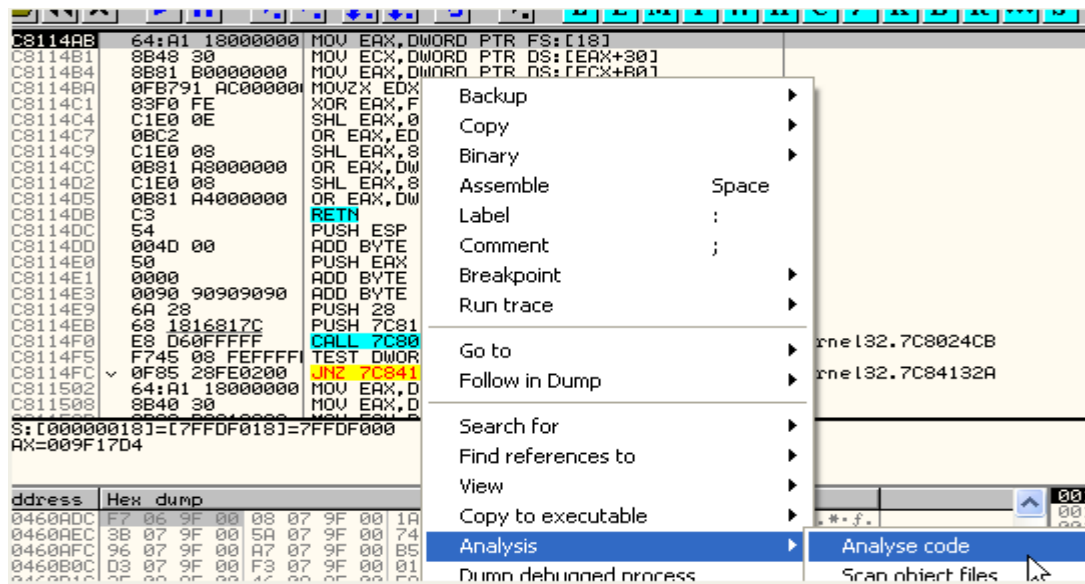
接下来的工作就是检查调用完该 API 函数以后是不是返回到我们刚刚设置了断点的返回地址处,因为有些壳会调用多个 API 函数来混淆视听,前面调用的几个 API 函数都是用来迷惑我们的,最后一个 API 函数才是其真正要调用的。

这里我们可以看到堆栈窗口显示的返回地址正好是我们刚刚设置了断点的那个返回地址 4271DC。

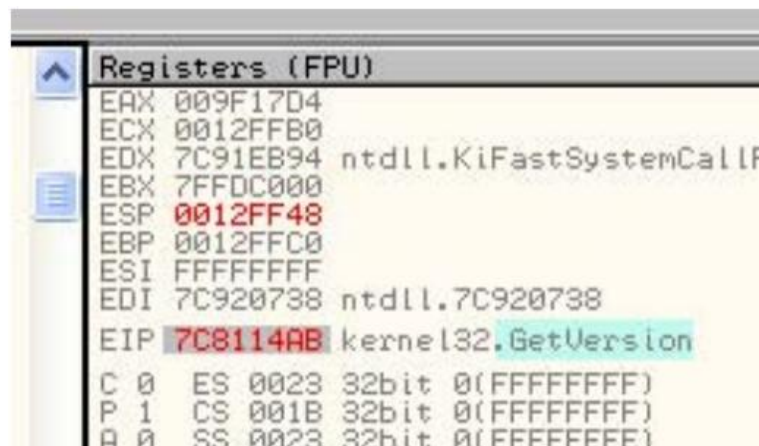
我们也可以在堆栈窗口上面单击鼠标右键选择 Follow in Disassembler 定位反汇编窗口中,也可以看到正好是我们设置了断点的返回地址处。



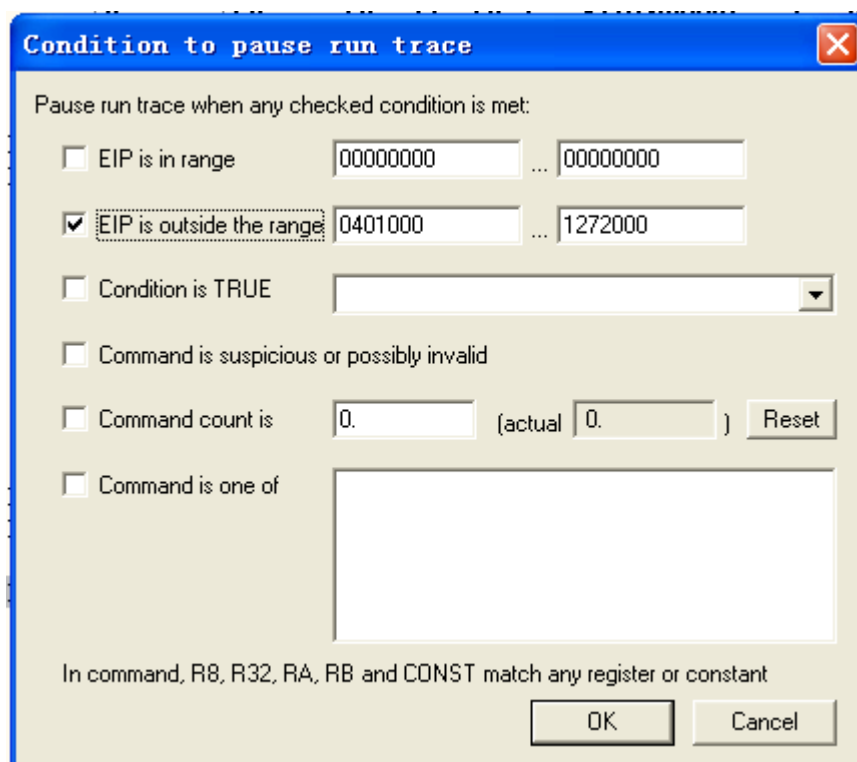
好了,现在停在了 API 函数的入口处,但是 OD 这里并没有提示 API 函数的名称,我们需要确定调用的是哪个 API 函数,我们来分析一下 DLL 的代码,在反汇编窗口中单击鼠标右键选择-Analysis-Analyse code。



这里我们可以 OD 左下方的解释窗口中看到该 API 函数的名称为 Kernel32.GetVersion,EIP 寄存器中也显示了 API 函数的名称。

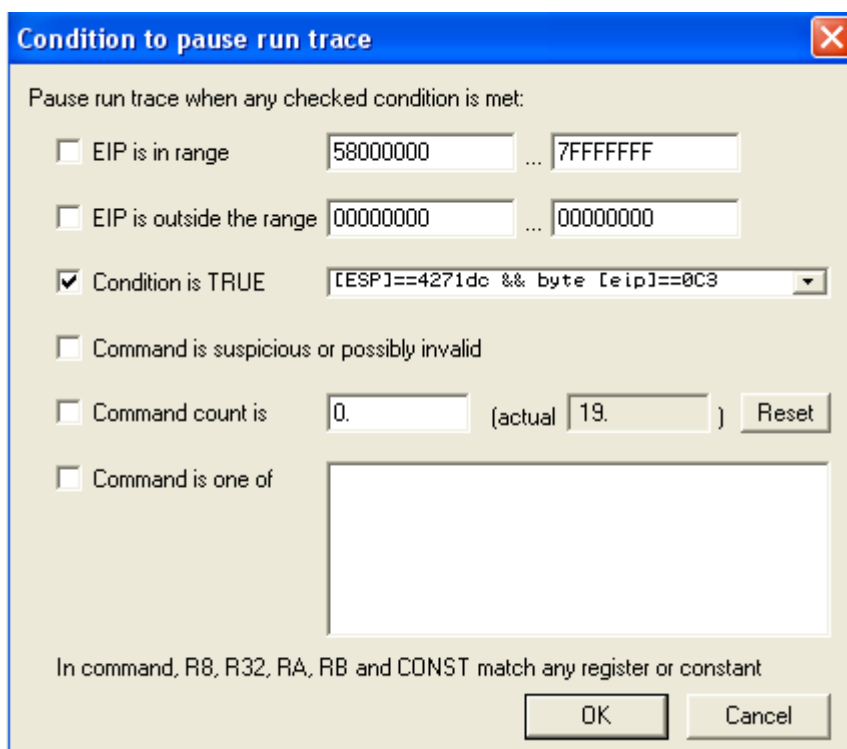


这里我们再来看下一个自动跟踪终止的条件选项 EIP is outside the range,通过设置这个选项我们也可以定位到其要调用的 API 函数:



(PS:这里原文笔误,作者写错了,作者写成了 401000 ~ 129000,因为是 401000 ~ 1272000 才对)

这里将 EIP is outside the range 这个选项的设置为 401000~1272000,即当 EIP 超出了主程序所在区段的范围就会停止自动跟踪。大家可以自行尝试。接下来还有一种情况。



有少数壳会将区段创建在系统 DLL 的区段之间,这样的话,我们需要用到 Condition is TRUE(当条件成立时)这个选项了,这里我们搜索 EIP 指向的指令为 RET,并且栈顶指针指向的是 4271DC 的指令。

这两个条件同时满足我们可以使用&&,表示这两个条件要同时满足,相当于 AND。

如果两个条件满足任意一个的话,我们可以使用||,相当于 OR。

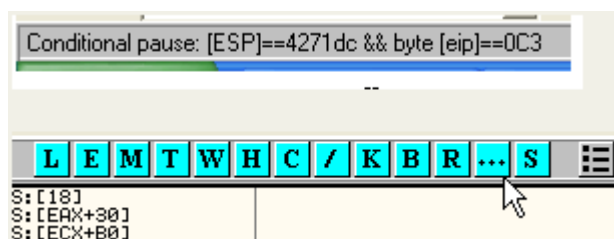
这里我们将自动跟踪终止条件设置为[ESP] == 4271DC && byte [EIP] == 0C3

[ESP] == 4271DC 即栈顶指针指向了返回地址 4271DC。

byte [EIP] == 0C3 即 EIP 指向了 API 函数的返回指令 RET。

接着让 OD 自动跟踪,我们可以看到断在了 API 函数的返回指令 RET 处。

7C8114A8	90	NOP	
7C8114A9	90	NOP	
7C8114AA	90	NOP	
7C8114AB	64:A1 180000	MOV EAX,DWORD PTR FS:[18]	
7C8114B1	8B48 30	MOV ECX,DWORD PTR DS:[EAX+30]	
7C8114B4	8B81 B0000000	MOV EAX,DWORD PTR DS:[ECX+B0]	
7C8114BA	0FB791 AC000	MOVZX EDX,WORD PTR DS:[ECX+AC]	
7C8114C1	83F0 FE	XOR EAX,FFFFFFFE	
7C8114C4	C1E0 0E	SHL EAX,0E	
7C8114C7	0BC2	OR EAX,EDX	
7C8114C9	C1E0 08	SHL EAX,8	
7C8114CC	0B81 A8000000	OR EAX,DWORD PTR DS:[ECX+A8]	
7C8114D2	C1E0 08	SHL EAX,8	
7C8114D5	0B81 A4000000	OR EAX,DWORD PTR DS:[ECX+A4]	
7C8114DB	C3	RET	
7C8114DC	54	DB 54	CHAR
7C8114DD	00	DB 00	
7C8114DE	4D	DB 4D	CHAR



由于[ESP] == 4271DC,byte [EIP] == 0C3 这两个条件都满足了,所以 OD 停止了自动跟踪,现在我们位于 API 函数的返回指令 RET 处,这里由于我们并不在该 API 函数的入口地址处,所以 OD 并不会提示该 API 函数的名称,因此有些壳会自己模拟执行 API 函数开头的两,三条指令,然后从 API 函数的第四,五行开始指令,这样 OD 也不会提示该 API 函数的名称,尽管如此,我们还是想办法可以定位该 API 函数的名称。

我们单击工具栏中的...按钮查看自动跟踪的日志信息。

Back	Thread	Module	Address	Command	Comment or message
19.	Main	UnPackMe	004271D6	CALL DWORD PTR DS:[460ADC]	
18.	Main		009F06F7	TEST ESP,ESP	
17.	Main		009F06F9	JNS SHORT 009F06FE	
16.	Main		009F06FE	INC EAX	
15.	Main		009F06FF	MOV EAX,9F17D3	
14.	Main		009F0704	INC EAX	
13.	Main		009F0705	PUSH DWORD PTR DS:[EAX]	
12.	Main		009F0707	RET	
11.	Main	kernel32	7C8114AB	MOV EAX,DWORD PTR FS:[18]	
10.	Main	kernel32	7C8114B1	MOV ECX,DWORD PTR DS:[EAX+30]	
9.	Main	kernel32	7C8114B4	MOV EAX,DWORD PTR DS:[ECX+B0]	
8.	Main	kernel32	7C8114BA	MOVZX EDX,WORD PTR DS:[ECX+AC]	
7.	Main	kernel32	7C8114C1	XOR EAX,FFFFFFFE	
6.	Main	kernel32	7C8114C4	SHL EAX,0E	
5.	Main	kernel32	7C8114C7	OR EAX,EDX	
4.	Main	kernel32	7C8114C9	SHL EAX,8	
3.	Main	kernel32	7C8114CC	OR EAX,DWORD PTR DS:[ECX+A8]	
2.	Main	kernel32	7C8114D2	SHL EAX,8	
1.	Main	kernel32	7C8114D5	OR EAX,DWORD PTR DS:[ECX+A4]	
0.	Main	kernel32	7C8114DB	RET	

我们看到红色箭头标注的这一行也没有显示 API 函数名称,我们分析一下代码,然后在该行上双击鼠标左键。

7C8114AB 64:A1 180000 MOV EAX,DWORD PTR FS:[18]

7C8114B1 8B48 30 MOV ECX,DWORD PTR DS:[EAX+30]

7C8114B4 8B81 B0000000 MOV EAX,DWORD PTR DS:[ECX+B0]

7C8114BA 0FB791 AC000 MOVZX EDX,WORD PTR DS:[ECX+AC]

7C8114C1 83F0 FE XOR EAX,FFFFFFFE

7C8114C4 C1E0 0E SHL EAX,0E

7C8114C7 0BC2 OR EAX,EDX

7C8114C9 C1E0 08 SHL EAX,8

7C8114CC 0B81 A8000000 OR EAX,DWORD PTR DS:[ECX+A8]

7C8114D2 C1E0 08 SHL EAX,8

7C8114D5 0B81 A4000000 OR EAX,DWORD PTR DS:[ECX+A4]

7C8114DB C3 RET

7C8114DC 54 DB 54

7C8114DD 00 DB 00

7C8114DE 4D DB 4D

7C8114E0 90 NOP

7C8114E1 90 NOP

7C8114E2 90 NOP

7C8114E3 90 NOP

7C8114E4 90 NOP

7C8114E5 90 NOP

7C8114E6 90 NOP

7C8114E7 90 NOP

7C8114E8 90 NOP

7C8114E9 90 NOP

7C8114EA 90 NOP

7C8114EB 90 NOP

7C8114EC 90 NOP

7C8114ED 90 NOP

7C8114EE 90 NOP

7C8114EF 90 NOP

7C8114F0 90 NOP

7C8114F1 90 NOP

7C8114F2 90 NOP

7C8114F3 90 NOP

7C8114F4 90 NOP

7C8114F5 90 NOP

7C8114F6 90 NOP

7C8114F7 90 NOP

7C8114F8 90 NOP

7C8114F9 90 NOP

7C8114FA 90 NOP

7C8114FB 90 NOP

7C8114FC 90 NOP

7C8114FD 90 NOP

7C8114FE 90 NOP

7C8114FF 90 NOP

7C811500 90 NOP

7C811501 90 NOP

7C811502 90 NOP

7C811503 90 NOP

7C811504 90 NOP

7C811505 90 NOP

7C811506 90 NOP

7C811507 90 NOP

7C811508 90 NOP

7C811509 90 NOP

7C81150A 90 NOP

7C81150B 90 NOP

7C81150C 90 NOP

7C81150D 90 NOP

7C81150E 90 NOP

7C81150F 90 NOP

7C811510 90 NOP

7C811511 90 NOP

7C811512 90 NOP

7C811513 90 NOP

7C811514 90 NOP

7C811515 90 NOP

7C811516 90 NOP

7C811517 90 NOP

7C811518 90 NOP

7C811519 90 NOP

7C81151A 90 NOP

7C81151B 90 NOP

7C81151C 90 NOP

7C81151D 90 NOP

7C81151E 90 NOP

7C81151F 90 NOP

7C811520 90 NOP

7C811521 90 NOP

7C811522 90 NOP

7C811523 90 NOP

7C811524 90 NOP

7C811525 90 NOP

7C811526 90 NOP

7C811527 90 NOP

7C811528 90 NOP

7C811529 90 NOP

7C81152A 90 NOP

7C81152B 90 NOP

7C81152C 90 NOP

7C81152D 90 NOP

7C81152E 90 NOP

7C81152F 90 NOP

7C811530 90 NOP

7C811531 90 NOP

7C811532 90 NOP

7C811533 90 NOP

7C811534 90 NOP

7C811535 90 NOP

7C811536 90 NOP

7C811537 90 NOP

7C811538 90 NOP

7C811539 90 NOP

7C81153A 90 NOP

7C81153B 90 NOP

7C81153C 90 NOP

7C81153D 90 NOP

7C81153E 90 NOP

7C81153F 90 NOP

7C811540 90 NOP

7C811541 90 NOP

7C811542 90 NOP

7C811543 90 NOP

7C811544 90 NOP

7C811545 90 NOP

7C811546 90 NOP

7C811547 90 NOP

7C811548 90 NOP

7C811549 90 NOP

7C81154A 90 NOP

7C81154B 90 NOP

7C81154C 90 NOP

7C81154D 90 NOP

7C81154E 90 NOP

7C81154F 90 NOP

7C811550 90 NOP

7C811551 90 NOP

7C811552 90 NOP

7C811553 90 NOP

7C811554 90 NOP

7C811555 90 NOP

7C811556 90 NOP

7C811557 90 NOP

7C811558 90 NOP

7C811559 90 NOP

7C81155A 90 NOP

7C81155B 90 NOP

7C81155C 90 NOP

7C81155D 90 NOP

7C81155E 90 NOP

7C81155F 90 NOP

7C811560 90 NOP

7C811561 90 NOP

7C811562 90 NOP

7C811563 90 NOP

7C811564 90 NOP

7C811565 90 NOP

7C811566 90 NOP

7C811567 90 NOP

7C811568 90 NOP

7C811569 90 NOP

7C81156A 90 NOP

7C81156B 90 NOP

7C81156C 90 NOP

7C81156D 90 NOP

7C81156E 90 NOP

7C81156F 90 NOP

7C811570 90 NOP

7C811571 90 NOP

7C811572 90 NOP

7C811573 90 NOP

7C811574 90 NOP

7C811575 90 NOP

7C811576 90 NOP

7C811577 90 NOP

7C811578 90 NOP

7C811579 90 NOP

7C81157A 90 NOP

7C81157B 90 NOP

7C81157C 90 NOP

7C81157D 90 NOP

7C81157E 90 NOP

7C81157F 90 NOP

7C811580 90 NOP

7C811581 90 NOP

7C811582 90 NOP

7C811583 90 NOP

7C811584 90 NOP

7C811585 90 NOP

7C811586 90 NOP

7C811587 90 NOP

7C811588 90 NOP

7C811589 90 NOP

7C81158A 90 NOP

7C81158B 90 NOP

7C81158C 90 NOP

7C81158D 90 NOP

7C81158E 90 NOP

7C81158F 90 NOP

7C811590 90 NOP

7C811591 90 NOP

7C811592 90 NOP

7C811593 90 NOP

7C811594 90 NOP

7C811595 90 NOP

7C811596 90 NOP

7C811597 90 NOP

7C811598 90 NOP

7C811599 90 NOP

7C81159A 90 NOP

7C81159B 90 NOP

7C81159C 90 NOP

7C81159D 90 NOP

7C81159E 90 NOP

7C81159F 90 NOP

7C8115A0 90 NOP

7C8115A1 90 NOP

7C8115A2 90 NOP

7C8115A3 90 NOP

7C8115A4 90 NOP

7C8115A5 90 NOP

7C8115A6 90 NOP

7C8115A7 90 NOP

7C8115A8 90 NOP

7C8115A9 90 NOP

7C8115AA 90 NOP

7C8115AB 90 NOP

7C8115AC 90 NOP

7C8115AD 90 NOP

7C8115AE 90 NOP

7C8115AF 90 NOP

7C8115B0 90 NOP

7C8115B1 90 NOP

7C8115B2 90 NOP

7C8115B3 90 NOP

7C8115B4 90 NOP

7C8115B5 90 NOP

7C8115B6 90 NOP

7C8115B7 90 NOP

7C8115B8 90 NOP

7C8115B9 90 NOP

7C8115BA 90 NOP

7C8115BB 90 NOP

7C8115BC 90 NOP

7C8115BD 90 NOP

7C8115BE 90 NOP

7C8115BF 90 NOP

7C8115C0 90 NOP

7C8115C1 90 NOP

7C8115C2 90 NOP

7C8115C3 90 NOP

7C8115C4 90 NOP

7C8115C5 90 NOP

7C8115C6 90 NOP

7C8115C7 90 NOP

7C8115C8 90 NOP

7C8115C9 90 NOP

7C8115CA 90 NOP

7C8115CB 90 NOP

7C8115CC 90 NOP

7C8115CD 90 NOP

7C8115CE 90 NOP

7C8115CF 90 NOP

7C8115D0 90 NOP

7C8115D1 90 NOP

7C8115D2 90 NOP

7C8115D3 90 NOP

7C8115D4 90 NOP

7C8115D5 90 NOP

7C8115D6 90 NOP

7C8115D7 90 NOP

7C8115D8 90 NOP

7C8115D9 90 NOP

7C8115DA 90 NOP

7C8115DB 90 NOP

7C8115DC 90 NOP

7C8115DD 90 NOP

7C8115DE 90 NOP

7C8115DF 90 NOP

7C8115E0 90 NOP

7C8115E1 90 NOP

7C8115E2 90 NOP

7C8115E3 90 NOP

7C8115E4 90 NOP

7C8115E5 90 NOP

7C8115E6 90 NOP

7C8115E7 90 NOP

7C8115E8 90 NOP

7C8115E9 90 NOP

7C8115EA 90 NOP

7C8115EB 90 NOP

7C8115EC 90 NOP

7C8115ED 90 NOP

7C8115EE 90 NOP

7C8115EF 90 NOP

7C8115F0 90 NOP

7C8115F1 90 NOP

7C8115F2 90 NOP

7C8115F3 90 NOP

7C8115F4 90 NOP

7C8115F5 90 NOP

7C8115F6 90 NOP

7C8115F7 90 NOP

7C8115F8 90 NOP

7C8115F9 90 NOP

7C8115FA 90 NOP

7C8115FB 90 NOP

7C8115FC 90 NOP

7C8115FD 90 NOP

7C8115FE 90 NOP

7C8115FF 90 NOP

7C811600 90 NOP

7C811601 90 NOP

7C811602 90 NOP

7C811603 90 NOP

7C811604 90 NOP

7C811605 90 NOP

7C811606 90 NOP

7C811607 90 NOP

7C811608 90 NOP

7C811609 90 NOP

7C81160A 90 NOP

7C81160B 90 NOP

7C81160C 90 NOP

7C81160D 90 NOP

7C81160E 90 NOP

7C81160F 90 NOP

7C811610 90 NOP

7C811611 90 NOP

7C811612 90 NOP

7C811613 90 NOP

7C811614 90 NOP

7C811615 90 NOP

7C811616 90 NOP

7C811617 90 NOP

7C811618 90 NOP

7C811619 90 NOP

7C81161A 90 NOP

7C81161B 90 NOP

7C81161C 90 NOP

7C81161D 90 NOP

7C81161E 90 NOP

7C81161F 90 NOP

7C811620 90 NOP

7C811621 90 NOP

7C811622 90 NOP

7C811623 90 NOP

7C811624 90 NOP

7C811625 90 NOP

7C811626 90 NOP

7C811627 90 NOP

7C811628 90 NOP

7C811629 90 NOP

7C81162A 90 NOP

7C81162B 90 NOP

7C81162C 90 NOP

7C81162D 90 NOP

7C81162E 90 NOP

7C81162F 90 NOP

7C811630 90 NOP

7C811631 90 NOP

7C811632 90 NOP

7C811633 90 NOP

7C811634 90 NOP

7C811635 90 NOP

7C811636 90 NOP

7C811637 90 NOP

7C811638 90 NOP

7C811639 90 NOP

7C81163A 90 NOP

7C81163B 90 NOP

7C81163C 90 NOP

7C81163D 90 NOP

7C81163E 90 NOP

7C81163F 90 NOP

7C811640 90 NOP

7C811641 90 NOP

7C811642 90 NOP

7C811643 90 NOP

7C811644 90 NOP

7C811645 90 NOP

7C811646 90 NOP

7C811647 90 NOP

7C811648 90 NOP

7C811649 90 NOP

7C81164A 90 NOP

7C81164B 90 NOP

7C81164C 90 NOP

7C81164D 90 NOP

7C81164E 90 NOP

7C81164F 90 NOP

7C811650 90 NOP

7C811651 90 NOP

7C811652 90 NOP

7C811653 90 NOP

7C811654 90 NOP

7C811655 90 NOP

7C811656 90 NOP

7C811657 90 NOP

7C811658 90 NOP

7C811659 90 NOP

7C81165A 90 NOP

7C81165B 90 NOP

7C81165C 90 NOP

7C81165D 90 NOP

7C81165E 90 NOP

7C81165F 90 NOP

7C811660 90 NOP

7C811661 90 NOP

7C811662 90 NOP

7C811663 90 NOP

7C811664 90 NOP

7C811665 90 NOP

7C811666 90 NOP

7C811667 90 NOP

7C811668 90 NOP

7C811669 90 NOP

7C81166A 90 NOP

7C81166B 90 NOP

7C81166C 90 NOP

7C81166D 90 NOP

7C81166E 90 NOP

7C81166F 90 NOP

7C811670 90 NOP

7C811671 90 NOP

7C811672 90 NOP

7C811673 90 NOP

7C811674 90 NOP

7C811675 90 NOP

7C811676 90 NOP

7C811677 90 NOP

7C811678 90 NOP

7C811679 90 NOP

7C81167A 90 NOP

7C81167B 90 NOP

7C81167C 90 NOP

7C81167D 90 NOP

7C81167E 90 NOP

7C81167F 90 NOP

7C811680 90 NOP

7C811681 90 NOP

7C811682 90 NOP

7C811683 90 NOP

7C811684 90 NOP

7C811685 90 NOP

7C811686 90 NOP

7C811687 90 NOP

7C811688 90 NOP

7C811689 90 NOP

7C81168A 90 NOP

7C81168B 90 NOP

7C81168C 90 NOP

7C81168D 90 NOP

7C81168E 90 NOP

7C81168F 90 NOP

7C811690 90 NOP

7C811691 90 NOP

7C811692 90 NOP

7C811693 90 NOP

7C811694 90 NOP

7C811695 90 NOP

7C811696 90 NOP

7C811697 90 NOP

7C811698 90 NOP

7C811699 90 NOP

7C81169A 90 NOP

7C81169B 90 NOP

7C81169C 90 NOP

7C81169D 90 NOP

7C81169E 90 NOP

7C81169F 90 NOP

7C8116A0 90 NOP

7C8116A1 90 NOP

7C8116A2 90 NOP

7C8116A3 90 NOP

7C8116A4 90 NOP

7C8116A5 90 NOP

7C8116A6 90 NOP

7C8116A7 90 NOP

7C8116A8 90 NOP

7C8116A9 90 NOP

7C8116AA 90 NOP

7C8116AB 90 NOP

7C8116AC 90 NOP

7C8116AD 90 NOP

7C8116AE 90 NOP

7C8116AF 90 NOP

7C8116B0 90 NOP

7C8116B1 90 NOP

7C8116B2 90 NOP

7C8116B3 90 NOP

7C8116B4 90 NOP

7C8116B5 90 NOP

7C8116B6 90 NOP

7C8116B7 90 NOP

7C8116B8 90 NOP

7C8116B9 90 NOP

7C8116BA 90 NOP

7C8116BB 90 NOP

7C8116BC 90 NOP

7C8116BD 90 NOP

7C8116BE 90 NOP

7C8116BF 90 NOP

7C8116C0 90 NOP

7C8116C1 90 NOP

7C8116C2 90 NOP

7C8116C3 90 NOP

7C8116C4 90 NOP

7C8116C5 90 NOP

7C8116C6 90 NOP

7C8116C7 90 NOP

7C8116C8 90 NOP

7C8116C9 90 NOP

7C8116CA 90 NOP

7C8116CB 90 NOP

7C8116CC 90 NOP

7C8116CD 90 NOP

7C8116CE 90 NOP

7C8116CF 90 NOP

7C8116D0 90 NOP

7C8116D1 90 NOP

7C8116D2 90 NOP

7C8116D3 90 NOP

7C8116D4 90 NOP

7C8116D5 90 NOP

7C8116D6 90 NOP

7C8116D7 90 NOP

7C8116D8 90 NOP

7C8116D9 90 NOP

7C8116DA 90 NOP

7C8116DB 90 NOP

7C8116DC 90 NOP

7C8116DD 90 NOP

7C8116DE 90 NOP

7C8116DF 90 NOP

7C8116E0 90 NOP

7C8116E1 90 NOP

7C8116E2 90 NOP

7C8116E3 90 NOP

7C8116E4 90 NOP

7C8116E5 90 NOP

7C8116E6 90 NOP

7C8116E7 90 NOP

7C8116E8 90 NOP

7C8116E9 90 NOP

7C8116EA 90 NOP

7C8116EB 90 NOP

7C8116EC 90 NOP

7C8116ED 90 NOP

7C8116EE 90 NOP

7C8116EF 90 NOP

7C8116F0 90 NOP

7C8116F1 90 NOP

7C8116F2 90 NOP

7C8116F3 90 NOP

7C8116F4 90 NOP

7C8116F5 90 NOP

7C8116F6 90 NOP

7C8116F7 90 NOP

7C8116F8 90 NOP

7C8116F9 90 NOP

7C8116FA 90 NOP

7C8116FB 90 NOP

7C8116FC 90 NOP

7C8116FD 90 NOP

7C8116FE 90 NOP

7C8116FF 90 NOP

7C811700 90 NOP

7C811701 90 NOP

7C811702 90 NOP

7C811703 90 NOP

7C811704 90 NOP

7C811705 90 NOP

7C811706 90 NOP

7C811707 90 NOP

7C811708 90 NOP

7C811709 90 NOP

7C81170A 90 NOP

7C81170B 90 NOP

7C81170C 90 NOP

7C81170D 90 NOP

7C81170E 90 NOP

<

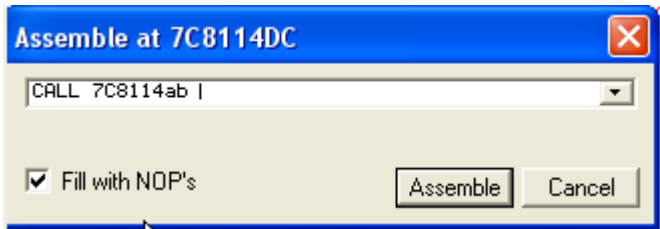
这里我们可以看到 OD 中 EIP 寄存器显示了该 API 函数的名称,当前 EIP 的值显示为红色,表示 EIP 寄存器被修改了,如果我们按减号键的话,EIP 会变为前一条执行过的指令的地址。

有些壳会去模拟执行 API 函数前几条指令,然后再去指令 API 函数后面的代码,这样 OD 也不会提示该 API 函数的名称。

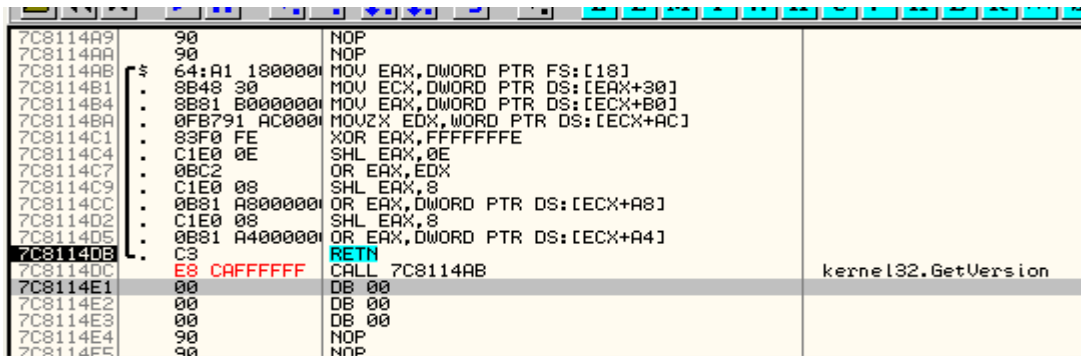
我们来看看如何应对这种情况:

我们在 API 函数返回指令 RET 的下一行指令空白处单击空格键,CALL 我们怀疑的 API 函数入口地址,这里我们输入:

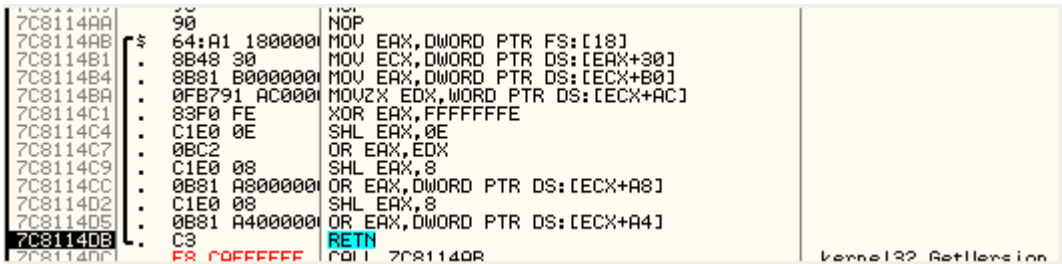
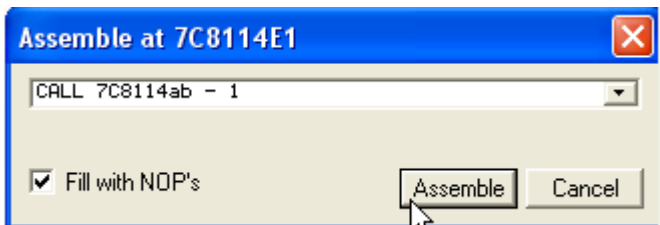
CALL 7C8114AB。



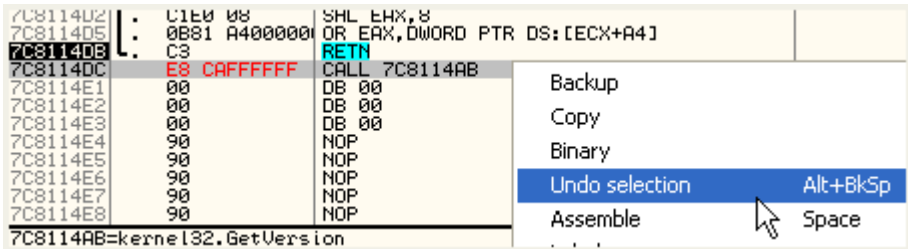
我们可以看到此时 OD 提示 API 函数的名称为 Kernel32.GetVersion。



如果我们 CALL 不是 API 函数的入口地址的话,OD 就不会显示函数名称了。

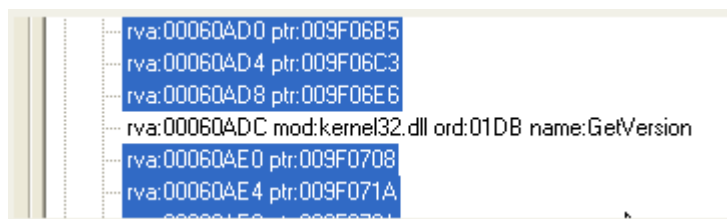
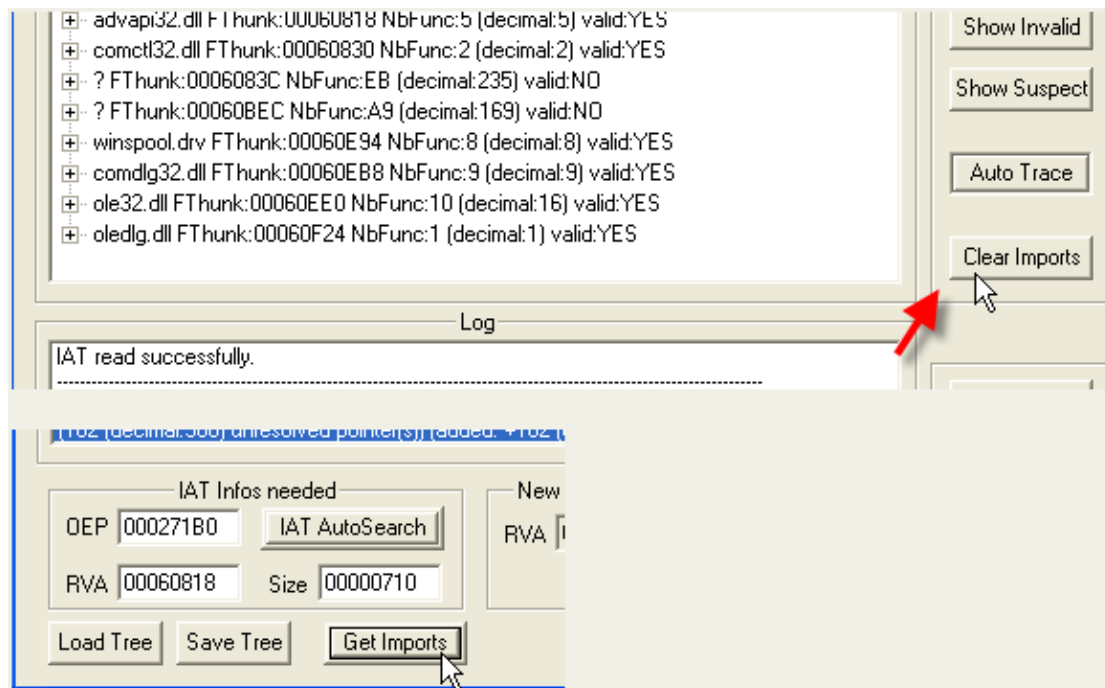


我们单击鼠标右键选择-undo selection(撤销刚刚所做的修改)。



好了,我们现在知道如何手工定位 API 函数的名称了,打开 IMP REC,我们需要对重定位过的 IAT 项进行修复,在 60ADC 这个无效

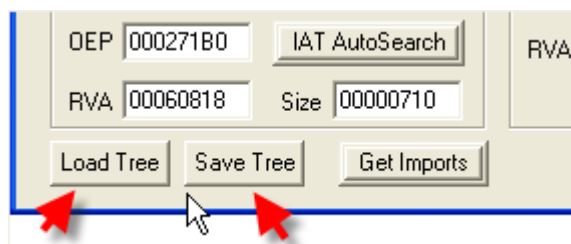
了有效。



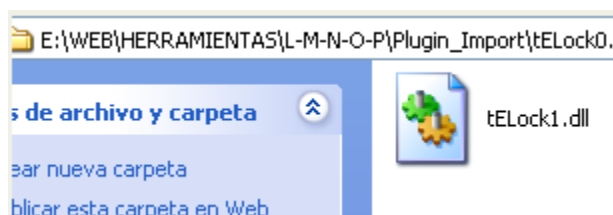
对于其他的重定向过的 IAT 项我们也可以按照上述方法来进行修复,接着就可以修复 dump 文件了。这种方法显得比较枯燥,纯属体力活。

(PS:大家想干这种体力活吗,不管你们想不想,反正我不想,嘿嘿)

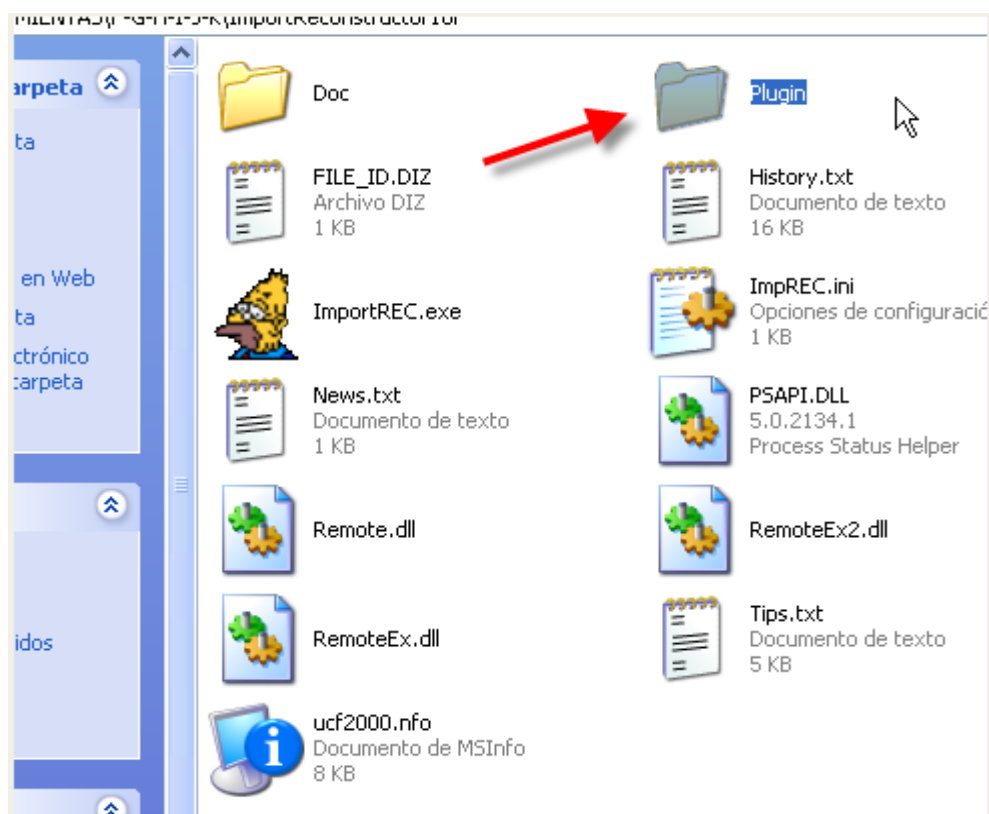
IMP REC 提示了一个功能可以协助我们完成这个手工修复工作。通过单击 Save Tree 按钮可以保存我们修复过的 IAT 项,当我们下次想继续修复其他项的时候可以通过单击 Load Tree 按钮导入我们之前保存的 IAT 信息,这样就可以继续之前没有完成的修复工作了。



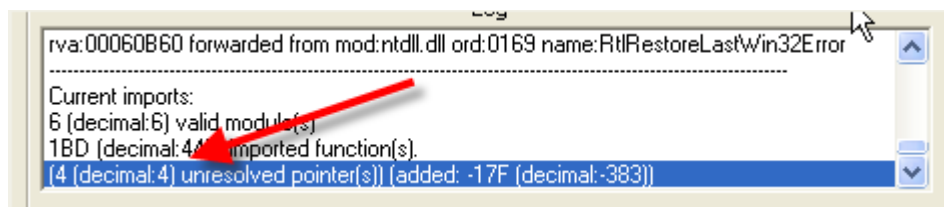
接下来的一种修复重定向的 IAT 项的方法就是借助于 IMP REC 的插件 tELock1,我们将其放到 IMP REC 目录下的 Plugin 文件下。下面我们来看看这个插件怎么用。



将插件的 DLL 拷贝到 IMP REC 中 Plugin 文件夹中。

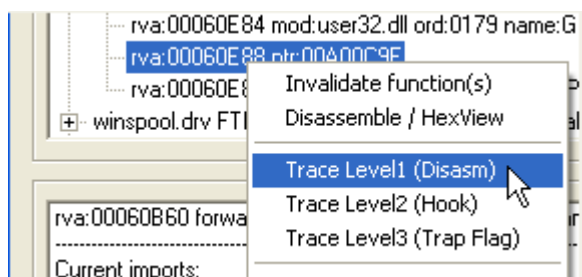


现在我们重启 IMP REC,输入 OEP,RVA,SIZE 等数据,单击 Get Imports,接着单击 Show Invalid,在无效的项上面单击鼠标右键选择 -Plugin Tracers-tElock1。这样该插件就会帮我们修复这些重定向的 IAT 项,我们可以看到日志窗口提示除了 4 项以外,其他重定向的 IAT 项都被修复了,这个插件不是很爽,帮我们完成大部分的重定向项的修复工作,剩下少数几个重定向 IAT 项我们可以自己手工修复。

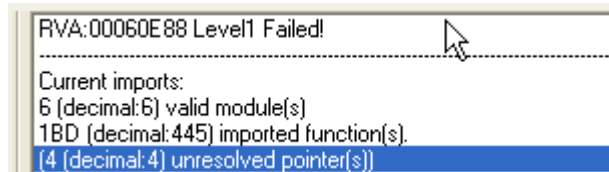


这里 IMP REC 提示 4 项未修复的,大家可以按照前面介绍的方法进行手工修复,这种插件的修复方式有明显的局限性,只能针对于特定的壳。

IMP REC 还自带有常规的 Tracers,我们可以试试看其能不能修复这些重定向过的 IAT 项,大多数情况,通常它也可以替代我们完成手工修复任务。



这里我们在第一个修复失败的项上面单击鼠标右键,我们可以看到 3 个等级,这里我们选择-Trace Level1(Disasm),看看等级 1 能不能起作用。



失败了,我们继续尝试 Level2,Level3,我们会发现这两项也不起作用,IMP REC 死掉了,看来这几个选项只能对比较简单的壳起作用。

接下来的这种修复重定向的方法叫做关键跳法,这种方法在很多脱壳教程中都有用到。

这种方法就是定位壳填充 IAT 的时机,看看何时填充正确 IAT 项,何时填充重定向过的 IAT 项。

举个例子:就拿 GetVersion 这一项来说吧,我们重启 UnPackMe-tElock0.98。

此时我们还没有到达 OEP 处,460ADC 中的值为 3D830870,在壳的解密例程执行过程中会将重定向过的值 9F06F7 填充到 460ADC 这个内存单元中。

Address	Hex dump	ASCII
00460ADC	70 08 83 3D F9 F2 67 B0 9B 85 EB 16 2C 9B 64 A2	p3=...g...dó
00460AEC	92 38 60 04 80 F4 74 C9 77 C7 C3 A5 A9 F1 32 42	E8'EÇtFw&H@±2B
00460AFC	89 DC B4 7A 7C C9 6E CA 89 9E AA F2 A2 89 81	ë-!ðz!fñ±ë×-óëü
00460B0C	2B 99 E9 53 60 42 69 A7 E1 3B 2E C9 7A BF 0D F4	+0US'B!ðp:..Fzγ..ŋ
00460B1C	0D F2 1C 6A FC 98 50 66 0C 02 03 91 0C BA 65 96	!..k*üPf==ë±..llëü

Address	Hex dump	ASCII
00460ADC	F7 06 9F 00 08 07 9F 00 1A 07 9F 00 2A 07 9F 00	..f..f..+..f..*..f..
00460AEC	3B 07 9F 00 5A 07 9F 00 74 07 9F 00 85 07 9F 00	..f..Z..f..t..f..ä..f..
00460AFC	96 07 9F 00 A7 07 9F 00 85 07 9F 00 C4 07 9F 00	ü..f..Q..f..Ä..f..-..f..
00460B0C	03 07 9F 00 F3 07 9F 00 01 08 9F 00 24 08 9F 00	ë..f..%..f..6..f..\$..f..
00460B1C	35 08 9F 00 46 08 9F 00 58 08 9F 00 68 08 9F 00	5..f..F..f..X..f..h..f..
00460B2C	79 08 9F 00 98 08 9F 00 B2 08 9F 00 C3 08 9F 00	y..f..ÿ..f..ÿ..f..h..f..
00460B3C	04 08 9F 00 E5 08 9F 00 F3 08 9F 00 02 09 9F 00	ë..f..ö..f..%..f..ö..f..

为了能够定位何时该 IAT 项被写入,我们可以对 460ADC 设置硬件写入断点,但是由于该壳会检测硬件断点,所以这里我们设置内存写入断点,让壳在此处写入重定向过的 IAT 值的时候断下来。

Address	Hex dump	ASCII
00460ADC	70 08 83 3D F9 F2 67 B0 9B 85 EB 16 2C 9B 64 A2	p3=...g...dó
00460AEC	92 38 60 04 80 F4 74 C9 77 C7 C3 A5 A9 F1 32 42	E8'EÇtFw&H@±2B
00460AFC	89 DC B4 00 7A 7C C9 6E CA 89 9E AA F2 A2 89 81	ë-!ðz!fñ±ë×-óëü
00460B0C	2B 99 E9 53 60 42 69 A7 E1 3B 2E C9 7A BF 0D F4	+0US'B!ðp:..Fzγ..ŋ
00460B1C	0D F2 1C 6A FC 98 50 66 0C 02 03 91 0C BA 65 96	!..k*üPf==ë±..llëü
00460B2C	00 12 AB 33 7B B1 74 33 55 F5 B5 40 20 11 04 00	..f..K..9\$3öY8+.re
00460B3C	7B B1 74 33 55 F5 B5 40 20 11 04 00 91 42 14 40	00 EF 00t0..l..= >ðë±ÿ\$
00460B4C	55 F5 B5 40 20 11 04 00 91 42 14 40 53 20 84 30	04 15 US&L+±ëŋü△F-]÷ð\$
00460B5C	58 B0 B2 00 20 11 04 00 91 42 14 40 53 20 84 30	
00460B6C	20 11 04 00 91 42 14 40 53 20 84 30 B9 C7 A0 70	
00460B7C	91 42 14 40 53 20 84 30 B9 C7 A0 70 ED 23 61 80	
00460B8C	53 20 84 30 B9 C7 A0 70 ED 23 61 80	
00460B9C	B9 C7 A0 70 ED 23 61 80	
00460BAC	ED 23 61 80	

运行起来,我们可以看到断在了这里。

Address	Hex dump	Instruction	Comment
00465D91	AA	STOS BYTE PTR ES:[EDI]	
00465D92	69D2 A5B0CD4B	IMUL EDX,EDX,4BCDB0A5	
00465D98	F9	STC	
00465D99	72 02	JB SHORT 00465D9D	
00465D9B	CD 20	INT 20	
00465D9D	D1C2	ROL EDX,1	
00465D9F	69D2 A5B0CD4B	IMUL EDX,EDX,4BCDB0A5	

这里并不是我们要定位的点,因为我们按 F8 键执行这一行会发现重定向过 IAT 值并没有被写进来,我们继续运行。

我们多运行几次,断在了这里:


```

00465FC2  8807      MOV BYTE PTR DS:[EDI],AL
00465FC4  D2C8      ROR AL,CL
00465FC6  32C3      XOR AL,BL
00465FC8  021F      ADD BL,BYTE PTR DS:[EDI]
00465FCA  12D9      ADC BL,CL
00465FCC  F6C1 01   TEST CL,1
00465FCF  75 0F      JNZ SHORT 00465FE0
00465FD1  D1EB      SHR EBX,1
00465FD3  F7C3 00000000  TEST EBX,0

```

这里也不是写入重定向过的 IAT 值,我们继续运行。

```

00465FDB  D3C3      RUL EBX,CL
00465FDD  8D1CDB    LEA EBX,DWORD PTR DS:[EBX]
00465FE0  AA        STOS BYTE PTR ES:[EDI]
00465FE1  49        DEC ECX
00465FE2  7F D3     JG SHORT 00465FB7
00465FE4  5F        POP EDI
00465FE5  5E        POP ESI

```

我们按 F8 键执行这行指令,发现也不是保存重定向过的 IAT 值,继续运行。

```

00466128  F3:A4     REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
0046612A  5E        POP ESI
0046612B  EB 8E     JMP SHORT 004660BB
0046612D  02D2     ADD DL,DL
0046612F  75 05     JNZ SHORT 00466136

```

断在了这里,这里将写入一个无效的值,但是并不是我们要定位的重定向过的 IAT 值 9F06F7。

Address	Hex dump	ASCII
00460ADC	A0 12 06 00 96 12 06 00 82 12 06 00 6E 12 06 00	@@@.Q@@.e@@.n@@.
00460AEC	60 12 06 00 54 12 06 00 3A 12 06 00 2A 12 06 00	'@@.T@@.:@@.@@.
00460AFC	1A 12 06 00 02 12 06 00 EE 11 06 00 08 11 06 00	+@@.0@@.'@@.i@@.
00460B0C	C8 11 06 00 B4 11 06 00 AC 11 06 00 0E 11 06 00	@@@.J@@.%@@.A@@.
00460B1C	70 11 06 00 52 11 06 00 36 11 06 00 24 11 06 00	p@@.R@@.6@@.S@@.
00460B2C	12 11 06 00 FE 10 06 00 EA 10 06 00 00 10 06 00	@@@.m@@.U@@.S@@.
00460B3C	C0 10 06 00 AE 10 06 00 A0 10 06 00 90 10 06 00	L@@.<@@.a@@.E@@.
00460B4C	80 10 06 00 72 10 06 00 62 10 06 00 50 10 06 00	C@@.r@@.b@@.P@@.
00460B5C	40 10 06 00 30 10 06 00 24 10 06 00 16 10 06 00	@@@.0@@.S@@.@@.
00460B6C	08 10 06 00 F8 0F 06 00 EC 0F 06 00 DC 0F 06 00	@@@.o@@.y@@.@@.
00460B7C	CC 0F 06 00 BC 0F 06 00 AE 0F 06 00 A0 0F 06 00	f@@.u@@.<@@.a@@.

我们执行 REP MOVS 指令,该无效的值被写入到该 IAT 项中,我们继续运行。

```

004664E5  8908      MOV DWORD PTR DS:[EAX],ECX
004664E7  58        POP EAX
004664E8  83C0 09   ADD EAX,9
004664EB  0185 56D44000  ADD DWORD PTR SS:[EBP+40D456],EAX
004664F1  EB 08     JMP SHORT 004664FB
004664F3  838D 52D44000  OR DWORD PTR SS:[EBP+40D452],FFFFFFFF
004664FA  61        POPAD
004664FB  03BD 4ED34000  ADD EDI,DWORD PTR SS:[EBP+40D34E]
00466501  85DB      TEST EBX,EBX
00466503  0F84 C7000000  JE 004665D0

```

我们断在了这里,我们可以看到当前 ECX 寄存器的值正好等于重定向过的 IAT 值 9F06F7。

Registers (FPU)	
EAX	00460ADC UnPackMe.00460ADC
ECX	009F06F7
EDX	00400000 UnPackMe.00400000
EBX	000612A0
ESP	0012FFA0
EBP	0005995E
ESI	00460014 UnPackMe.00460014
EDI	009F166C
EIP	004664E5 UnPackMe.004664E5
C 0	ES 0023 32bit 0(FFFFFFFF)
P 0	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
Z 0	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 003B 32bit 7FDF000(FFF)
T 0	GS 0000 NULL

ECX = 9F06F7,将被保存到 EAX = 460ADC 指向的 IAT 中,这里就是我们要定位的地方。

Address	Hex dump	ASCII
004600DC	F7 06 9F 00 96 12 06 00 82 12 06 00 6E 12 06 00	~*f.û*.é*.n*.
004600EC	60 12 06 00 54 12 06 00 3A 12 06 00 2A 12 06 00	'*.T*.:*.**.
004600FC	1A 12 06 00 02 12 06 00 EE 11 06 00 D8 11 06 00	+*.0*.~*.i*.
0046000C	C8 11 06 00 B4 11 06 00 AC 11 06 00 8E 11 06 00	~*.~*.k*.A*.

我们按下 F8 键,9F06F7 就会被保存到 460ADC 中,这只是第一步,接下来我们需要定位关键跳转。

File View Debug Plugins Options Window Help		
L E M T W H C / K B R ... S		
004664C0	8B80 52D44000	MOV EDI, DWORD PTR SS:[EBP+40D452]
004664D3	8B85 5AD44000	MOV EAX, DWORD PTR SS:[EBP+40D45A]
004664D9	0385 4ED34000	ADD EAX, DWORD PTR SS:[EBP+40D34E]
004664DF	8B80 56D44000	MOV ECX, DWORD PTR DS:[EAX], ECX
004664E5	9308	MOV DWORD PTR DS:[EAX], ECX
004664E7	58	POP EAX
004664E8	83C0 09	ADD EAX, 9
004664EB	0185 56D44000	ADD DWORD PTR SS:[EBP+40D456], EAX
004664F1	EB 08	JMP SHORT 004664FB
004664F3	8380 52D44000	OR DWORD PTR SS:[EBP+40D452], FFFFFFFF
004664F8	61	POPAD
004664FB	038D 4ED34000	ADD EDI, DWORD PTR SS:[EBP+40D34E]
00466501	850B	TEST EBX, EBX
00466503	0F84 C7000000	JE 0046650B
00466509	F7C3 00000000	TEST EBX, 00000000
0046650F	5A 00	PUSH 0
00466511	75 0F	JNZ SHORT 00466522
00466513	0D5C13 02	LEA EBX, DWORD PTR DS:[EBX+EDX+2]
00466517	803B 00	CMP BYTE PTR DS:[EBX], 0
0046651A	0F84 3B000000	JE 00466525
00466520	EB 45	JMP SHORT 00466567
00466522	FF0424	INC DWORD PTR SS:[ESP]
00466525	66180B	TEST BX, BX
00466528	0F84 85000000	JE 00466533
0046652E	8B85 4AD34000	MOV EAX, DWORD PTR SS:[EBP+40D34A]

我们往下跟几步,达到了 46651A 处 JE 指令处,我们看看寄存器窗口中 EBX 寄存器的值,当前 EBX 正好指向了 GetVersion 的函数名称字符串。

00466567	81E3 FFFFFFFF	AND EBX, FFFFFFFF
0046656D	53	PUSH EBX
0046656F	FFB5 4AD34000	PUSH DWORD PTR SS:[EBP+40D34A]
00466574	FF95 E08A0000	CALL DWORD PTR SS:[EBP+408AE0]
0046657A	48	INC EAX
0046657B	48	DEC EAX
0046657C	75 33	JNZ SHORT 004665B1
0046657E	58	POP EAX
0046657F	F9	STC
00466580	0F82 61FDFFFF	JB 0046662E7
00466586	47	INC EDI
00466587	44	INC ESP
00466588	49	DEC ECX

继续往下我们可以看到会调用 GetProcAddress 获取 GetVersion 这个 API 函数的地址。

0012FF94	0046657A	CALL to GetProcAddress from UnPackMe.00466574
0012FF98	7C800000	hModule = 7C800000 (kernel32)
0012FF9C	004612A2	ProcNameOrOrdinal = "GetVersion"
0012FFA0	00000000	
0012FFA4	7C920738	ntdll.7C920738

我们可以在堆栈窗口中看到参数情况,按 F8 键执行这个 CALL。

Registers (FPU)		
EAX	7C8114AB	kernel32.GetVersion
ECX	7C929AEB	ntdll.7C929AEB
EDX	7C98C0D8	ntdll.7C98C0D8
EBX	004612A2	ASCII "GetVersion"
ESP	0012FF94	ASCII "zeF"
EBP	0005995E	
ESI	00460014	UnPackMe.00460014
EDI	009F17D4	
EIP	7C80AC87	kernel32.7C80AC87
C 0	ES 0023 32bit 0(FFFFFFFF)	

我们可以看到此时 EAX 保存了 GetVersion 这个 API 函数的地址,我们继续往下跟。

0046657C	75 33	JNZ SHORT 004665B1
0046657E	58	POP EAX
0046657F	F9	STC
00466580	0F82 61FDFFFF	JB 0046662E7
00466586	47	INC EDI
00466587	44	INC ESP
00466588	49	DEC ECX
00466589	3332	XOR ESI, DWORD PTR DS:[EDX]
0046658B	2F 44	TNC

这里有一个条件跳转,这个条件跳转是成立的,我们不跟过去,直接单击鼠标右键选择-Follow。

004665B1	8907	MOV DWORD PTR DS:[EDI],EAX	
004665B3	58	POP EAX	
004665B4	48	DEC EAX	
004665B5	74 0D	JE SHORT 004665C4	UnPackMe.004665C4
004665B7	40	INC EAX	
004665B8	F8	CLC	
004665B9	66:8943 FE	MOV WORD PTR DS:[EBX-2],AX	
004665BD	8803	MOV BYTE PTR DS:[EBX],AL	
004665BF	43	INC EBX	
004665C0	3803	CMP BYTE PTR DS:[EBX],AL	
004665C2	75 F9	JNZ SHORT 004665BD	UnPackMe.004665BD
004665C4	8385 4ED34000	ADD DWORD PTR SS:[EBP+40D34E],4	
004665C8	E9 BAFDFFFF	JMP 0046638A	UnPackMe.0046638A

这里我们可以看到 GetVersion 的函数地址并不是被填充到 IAT 中,而是被写入到 EDI 指向的内存单元中,当前 EDI 为:

Registers (FPU)		
EAX	7C81140B	kernel32.GetVersion
ECX	7C929AEB	ntdll.7C929AEB
EDX	7C98C0D8	ntdll.7C98C0D8
EBX	004612A2	ASCII "GetVersion"
ESP	0012FFA0	
EBP	0005995E	
ESI	00460014	UnPackMe.00460014
EDI	009F17D4	
EIP	0046657C	UnPackMe.0046657C

保存了该 API 函数的地址以后,紧接着我们到了 JMP 指令处。

004665B1	8907	MOV DWORD PTR DS:[EDI],EAX	
004665B3	58	POP EAX	
004665B4	48	DEC EAX	
004665B5	74 0D	JE SHORT 004665C4	
004665B7	40	INC EAX	
004665B8	F8	CLC	
004665B9	66:8943 FE	MOV WORD PTR DS:[EBX-2],AX	
004665BD	8803	MOV BYTE PTR DS:[EBX],AL	
004665BF	43	INC EBX	
004665C0	3803	CMP BYTE PTR DS:[EBX],AL	
004665C2	75 F9	JNZ SHORT 004665BD	
004665C4	8385 4ED34000	ADD DWORD PTR SS:[EBP+40D34E],4	
004665C8	E9 BAFDFFFF	JMP 0046638A	
004665D0	83C6 14	ADD ESI,14	
004665D3	8B95 62D34000	MOV EDX,DWORD PTR SS:[EBP+40D362]	
004665D9	E9 48FCFFFF	JMP 00466226	
004665DE	61	POPAD	
004665DF	C3	RETN	

这里通过这个 JMP 我们又将回到这个过程的开始处,这里是一个循环,我们单击鼠标右键选择-Follow。

0046638A	8B95 62D34000	MOV EDX,DWORD PTR SS:[EBP+40D362]	
00466390	8B06	MOV EAX,DWORD PTR DS:[ESI]	
00466392	85C0	TEST EAX,EAX	
00466394	75 0B	JNZ SHORT 004663A1	UnPackMe.004663A1
00466396	8B46 10	MOV EAX,DWORD PTR DS:[ESI+10]	
00466399	85C0	TEST EAX,EAX	
0046639B	0F84 46FFFFFF	JE 004662E7	UnPackMe.004662E7
004663A1	03C2	ADD EAX,EDX	
004663A3	0385 4ED34000	ADD EAX,DWORD PTR SS:[EBP+40D34E]	
004663A9	8B18	MOV EBX,DWORD PTR DS:[EAX]	
004663AB	F7C3 00000000	TEST EBX,00000000	
004663B1	74 06	JE SHORT 004663B9	UnPackMe.004663B9

这里接下来又要将第二个重定向过的 IAT 值填充的下一个 IAT 项中,然后通过 GetProcAddress 获取对应的 API 函数地址,并将获取到的 API 函数地址保存到处,接着又是第三个重定向过的 IAT 值,循环往复,直到所有 IAT 项都被处理完毕为止。

由于壳只是对一部分 IAT 项进行重定向,所以下面我们来看看壳对 IAT 项重定向以及不重定向流程的差异。

IAT 是如何被重定向的

0046638A	8B95 62D34000	MOV EDX,DWORD PTR SS:[EBP+40D362]	
00466390	8B06	MOV EAX,DWORD PTR DS:[ESI]	
00466392	85C0	TEST EAX,EAX	
00466394	75 0B	JNZ SHORT 004663A1	UnPackMe.004663A1
00466396	8B46 10	MOV EAX,DWORD PTR DS:[ESI+10]	
00466399	85C0	TEST EAX,EAX	
0046639B	0F84 46FFFFFF	JE 004662E7	UnPackMe.004662E7
004663A1	03C2	ADD EAX,EDX	
004663A3	0385 4ED34000	ADD EAX,DWORD PTR SS:[EBP+40D34E]	

0046638A	8B95 62D34000	MOV EDX,DWORD PTR SS:[EBP+40D362]	
00466390	8B06	MOV EAX,DWORD PTR DS:[ESI]	
00466392	85C0	TEST EAX,EAX	
00466394	75 0B	JNZ SHORT 004663A1	UnPackMe.004663A1
00466396	8B46 10	MOV EAX,DWORD PTR DS:[ESI+10]	
00466399	85C0	TEST EAX,EAX	
0046639B	0F84 46FFFFFF	JE 004662E7	UnPackMe.004662E7
004663A1	03C2	ADD EAX,EDX	
004663A3	0385 4ED34000	ADD EAX,DWORD PTR SS:[EBP+40D34E]	
004663A9	8B18	MOV EBX,DWORD PTR DS:[EAX]	
004663AB	F7C3 00000000	TEST EBX,80000000	
004663B1	74 06	JE SHORT 004663B9	UnPackMe.004663B9
004663B3	8120 00000000	AND DWORD PTR DS:[EAX],80000000	
004663B9	8B7E 10	MOV EDI,DWORD PTR DS:[ESI+10]	
004663BC	03FA	ADD EDI,EDX	
004663BE	80A5 D6CC4000	AND BYTE PTR SS:[EBP+40CCD6],0FF	
004663C5	0F84 30010000	JE 004664FB	UnPackMe.004664FB
004663CB	80A5 D7CC4000	AND BYTE PTR SS:[EBP+40CCD7],0FF	
004663D2	0F84 23010000	JE 004664FB	UnPackMe.004664FB
004663D8	89BD 5AD44000	MOV DWORD PTR SS:[EBP+40D45A],EDI	
004663DE	8B85 52D44000	MOV EAX,DWORD PTR SS:[EBP+40D452]	
004663E4	40	INC EAX	
004663E5	0F84 10010000	JE 004664FB	UnPackMe.004664FB
004663EB	48	DEC EAX	
004663EC	0F85 B2000000	JNZ 004664A4	UnPackMe.004664A4
004663F2	60	PUSHAD	
004663F3	8BF7	MOV ESI,EDI	

0046638A	8B95 62D34000	MOV EDX,DWORD PTR SS:[EBP+40D362]	
00466390	8B06	MOV EAX,DWORD PTR DS:[ESI]	
00466392	85C0	TEST EAX,EAX	
00466394	75 0B	JNZ SHORT 004663A1	UnPackMe.004663A1
00466396	8B46 10	MOV EAX,DWORD PTR DS:[ESI+10]	
00466399	85C0	TEST EAX,EAX	
0046639B	0F84 46FFFFFF	JE 004662E7	UnPackMe.004662E7
004663A1	03C2	ADD EAX,EDX	
004663A3	0385 4ED34000	ADD EAX,DWORD PTR SS:[EBP+40D34E]	
004663A9	8B18	MOV EBX,DWORD PTR DS:[EAX]	
004663AB	F7C3 00000000	TEST EBX,80000000	
004663B1	74 06	JE SHORT 004663B9	UnPackMe.004663B9
004663B3	8120 00000000	AND DWORD PTR DS:[EAX],80000000	
004663B9	8B7E 10	MOV EDI,DWORD PTR DS:[ESI+10]	
004663BC	03FA	ADD EDI,EDX	
004663BE	80A5 D6CC4000	AND BYTE PTR SS:[EBP+40CCD6],0FF	
004663C5	0F84 30010000	JE 004664FB	UnPackMe.004664FB
004663CB	80A5 D7CC4000	AND BYTE PTR SS:[EBP+40CCD7],0FF	
004663D2	0F84 23010000	JE 004664FB	UnPackMe.004664FB
004663D8	89BD 5AD44000	MOV DWORD PTR SS:[EBP+40D45A],EDI	
004663DE	8B85 52D44000	MOV EAX,DWORD PTR SS:[EBP+40D452]	

004663BC	03FA	ADD EDI,EDX	
004663BE	80A5 D6CC4000	AND BYTE PTR SS:[EBP+40CCD6],0FF	
004663C5	0F84 30010000	JE 004664FB	UnPackMe.004664FB
004663CB	80A5 D7CC4000	AND BYTE PTR SS:[EBP+40CCD7],0FF	
004663D2	0F84 23010000	JE 004664FB	UnPackMe.004664FB
004663D8	89BD 5AD44000	MOV DWORD PTR SS:[EBP+40D45A],EDI	
004663DE	8B85 52D44000	MOV EAX,DWORD PTR SS:[EBP+40D452]	
004663E4	40	INC EAX	

004663DE	8B85 52D44000	MOV EAX,DWORD PTR SS:[EBP+40D452]	
004663E4	40	INC EAX	
004663E5	0F84 10010000	JE 004664FB	UnPackMe.004664FB
004663EB	48	DEC EAX	
004663EC	0F85 B2000000	JNZ 004664A4	UnPackMe.004664A4
004663F2	60	PUSHAD	
004663F3	8BF7	MOV ESI,EDI	

004663DE	8B85 52D44000	MOV EAX,DWORD PTR SS:[EBP+40D452]	
004663E4	40	INC EAX	
004663E5	0F84 10010000	JE 004664FB	UnPackMe.004664FB
004663EB	48	DEC EAX	
004663EC	0F85 B2000000	JNZ 004664A4	UnPackMe.004664A4
004663F2	60	PUSHAD	
004663F3	8BF7	MOV ESI,EDI	

004664A4	8D85 14BB4000	LEA EAX,DWORD PTR SS:[EBP+40BB14]	
004664AA	FFB5 FBCA4000	PUSH DWORD PTR SS:[EBP+40CAFB]	
004664B0	0FB608	MOVZX ECX,BYTE PTR DS:[EAX]	
004664B3	FF0C24	DEC DWORD PTR SS:[ESP]	
004664B6	7E 05	JLE SHORT 004664B0	UnPackMe.004664B0
004664B8	40	INC EAX	
004664B9	03C1	ADD EAX,ECX	
004664BB	EB F3	JMP SHORT 004664B0	UnPackMe.004664B0
004664BD	890C24	MOV DWORD PTR SS:[ESP],ECX	
004664C0	FF85 FBCA4000	INC DWORD PTR SS:[EBP+40CAFB]	
004664C2	8B85 52D44000	MOV EAX,DWORD PTR SS:[EBP+40D452]	

这里是一个小循环,继续往下:

004664FB	03BD 4ED34000	ADD EDI,DWORD PTR SS:[EBP+40D34E]	
00466501	85DB	TEST EBX,EBX	
00466503	0F84 C7000000	JE 004665D0	UnPackMe.004665D0
00466509	F7C3 00000000	TEST EBX,80000000	
0046650F	6A 00	PUSH 0	
00466511	75 0F	JNZ SHORT 00466522	UnPackMe.00466522
00466513	8D5C13 02	LEA EBX,DWORD PTR DS:[EBX+EDX+2]	
00466517	803B 00	CMP BYTE PTR DS:[EBX],0	
0046651A	0F84 93000000	JE 004665B3	UnPackMe.004665B3
00466520	EB 45	JMP SHORT 00466567	UnPackMe.00466567
00466522	FF0424	INC DWORD PTR SS:[ESP]	
00466525	66:85DB	TEST BX,BX	
00466528	0F84 85000000	JE 004665B3	UnPackMe.004665B3
0046652E	8B85 4AD34000	MOV EAX,DWORD PTR SS:[EBP+40D34A]	
00466534	3B85 42D44000	CMP EAX,DWORD PTR SS:[EBP+40D442]	
0046653A	75 2B	JNZ SHORT 00466567	UnPackMe.00466567
0046653C	81E3 FFFFFFFF	AND EBX,7FFFFFFF	
00466542	8BD3	MOV EDX,EBX	
00466544	8D1495 FCFFFFFF	LEA EDX,DWORD PTR DS:[EDX*4-4]	
0046654B	8B9D 4AD34000	MOV EBX,DWORD PTR SS:[EBP+40D34A]	
00466551	8B43 3C	MOV EAX,DWORD PTR DS:[EBX+3C]	
00466554	8B4418 78	MOV EAX,DWORD PTR DS:[EAX+EBX+78]	
00466558	035C18 1C	ADD EBX,DWORD PTR DS:[EAX+EBX+1C]	
0046655C	8B041A	MOV EAX,DWORD PTR DS:[EDX+EBX]	
0046655F	0385 4AD34000	ADD EAX,DWORD PTR SS:[EBP+40D34A]	
00466565	EB 13	JMP SHORT 0046657A	UnPackMe.0046657A
00466567	81E3 FFFFFFFF	AND EBX,7FFFFFFF	
0046656D	53	PUSH EBX	
0046656E	FFB5 4AD34000	PUSH DWORD PTR SS:[EBP+40D34A]	
00466574	FF95 E0BA4000	CALL DWORD PTR SS:[EBP+40BAE0]	
0046657D	40	INC EAX	

004664FB	03BD 4ED34000	ADD EDI,DWORD PTR SS:[EBP+40D34E]	
00466501	85DB	TEST EBX,EBX	
00466503	0F84 C7000000	JE 004665D0	UnPackMe.004665D0
00466509	F7C3 00000000	TEST EBX,80000000	
0046650F	6A 00	PUSH 0	
00466511	75 0F	JNZ SHORT 00466522	UnPackMe.00466522
00466513	8D5C13 02	LEA EBX,DWORD PTR DS:[EBX+EDX+2]	
00466517	803B 00	CMP BYTE PTR DS:[EBX],0	
0046651A	0F84 93000000	JE 004665B3	UnPackMe.004665B3
00466520	EB 45	JMP SHORT 00466567	UnPackMe.00466567
00466522	FF0424	INC DWORD PTR SS:[ESP]	
00466525	66:85DB	TEST BX,BX	

004664FB	03BD 4ED34000	ADD EDI,DWORD PTR SS:[EBP+40D34E]	
00466501	85DB	TEST EBX,EBX	
00466503	0F84 C7000000	JE 004665D0	UnPackMe.004665D0
00466509	F7C3 00000000	TEST EBX,80000000	
0046650F	6A 00	PUSH 0	
00466511	75 0F	JNZ SHORT 00466522	UnPackMe.00466522
00466513	8D5C13 02	LEA EBX,DWORD PTR DS:[EBX+EDX+2]	
00466517	803B 00	CMP BYTE PTR DS:[EBX],0	
0046651A	0F84 93000000	JE 004665B3	UnPackMe.004665B3
00466520	EB 45	JMP SHORT 00466567	UnPackMe.00466567
00466522	FF0424	INC DWORD PTR SS:[ESP]	
00466525	66:85DB	TEST BX,BX	
00466528	0F84 85000000	JE 004665B3	UnPackMe.004665B3
0046652E	8B85 4AD34000	MOV EAX,DWORD PTR SS:[EBP+40D34A]	
00466534	3B85 42D44000	CMP EAX,DWORD PTR SS:[EBP+40D442]	
0046653A	75 2B	JNZ SHORT 00466567	UnPackMe.00466567
0046653C	81E3 FFFFFFFF	AND EBX,7FFFFFFF	

0046651A	0F84 93000000	JE 004665B3	UnPackMe.004665B3
00466520	EB 45	JMP SHORT 00466567	UnPackMe.00466567
00466522	FF0424	INC DWORD PTR SS:[ESP]	
00466525	66:85DB	TEST BX,BX	
00466528	0F84 85000000	JE 004665B3	UnPackMe.004665B3
0046652E	8B85 4AD34000	MOV EAX,DWORD PTR SS:[EBP+40D34A]	
00466534	3B85 42D44000	CMP EAX,DWORD PTR SS:[EBP+40D442]	
0046653A	75 2B	JNZ SHORT 00466567	UnPackMe.00466567
0046653C	81E3 FFFFFFFF	AND EBX,7FFFFFFF	
00466542	8BD3	MOV EDX,EBX	
00466544	8D1495 FCFFFFFF	LEA EDX,DWORD PTR DS:[EDX*4-4]	
0046654B	8B9D 4AD34000	MOV EBX,DWORD PTR SS:[EBP+40D34A]	
00466551	8B43 3C	MOV EAX,DWORD PTR DS:[EBX+3C]	
00466554	8B4418 78	MOV EAX,DWORD PTR DS:[EAX+EBX+78]	
00466558	035C18 1C	ADD EBX,DWORD PTR DS:[EAX+EBX+1C]	
0046655C	8B041A	MOV EAX,DWORD PTR DS:[EDX+EBX]	
0046655F	0385 4AD34000	ADD EAX,DWORD PTR SS:[EBP+40D34A]	
00466565	EB 13	JMP SHORT 0046657A	UnPackMe.0046657A
00466567	81E3 FFFFFFFF	AND EBX,7FFFFFFF	
0046656D	53	PUSH EBX	
0046656E	FFB5 4AD34000	PUSH DWORD PTR SS:[EBP+40D34A]	
00466574	FF95 E0BA4000	CALL DWORD PTR SS:[EBP+40BAE0]	kernel32.GetProcAddress
0046657A	40	INC EAX	
0046657B	48	DEC EAX	
0046657D	7E 00	JNZ SHORT 00466584	UnPackMe.00466584

这里是调用 GetProcAddress,接着就会到保存 GetProcAddress 获取到的 API 函数地址处。

00466567	81E3 FFFFFFFF	AND EBX,7FFFFFFF	
0046656D	53	PUSH EBX	
0046656E	FFB5 4AD34000	PUSH DWORD PTR SS:[EBP+40D34A]	
00466574	FF95 E0BA4000	CALL DWORD PTR SS:[EBP+40BAE0]	
0046657A	40	INC EAX	
0046657B	48	DEC EAX	
0046657C	75 33	JNZ SHORT 004665B1	UnPackMe.004665B1
0046657E	58	POP EAX	
0046657F	F9	STC	
00466580	0F82 61FDFFFF	JB 004662E7	UnPackMe.004662E7
00466586	47	INC EDI	

004665B1	8907	MOV DWORD PTR DS:[EDI],EAX	kernel32.CreateDirectoryA
004665B3	58	POP EAX	
004665B4	48	DEC EAX	
004665B5	74 0D	JE SHORT 004665C4	UnPackMe.004665C4
004665B7	40	INC EAX	
004665B8	F8	CLC	
004665B9	66:8943 FE	MOV WORD PTR DS:[EBX-2],AX	
004665BD	8803	MOV BYTE PTR DS:[EBX],AL	
004665BF	43	INC EBX	
004665C0	3803	CMP BYTE PTR DS:[EBX],AL	
004665C2	75 F9	JNZ SHORT 004665BD	UnPackMe.004665BD
004665C4	8385 4ED34000	ADD DWORD PTR SS:[EBP+40D34E],4	
004665C6	E9 BAFDFFFF	JMP 0046638A	UnPackMe.0046638A
004665D0	83C6 14	ADD ESI,14	

以上就是跟踪一个 IAT 项被重定向的完成流程,其中有很多的条件跳转。现在我们跟到 OEP 处,定位到一个正确的 IAT 项,比较一下两者有什么差别。

这里我们选择 460BAC 这一项。重新启动程序。

Address	Hex dump	ASCII
00460BDC	F7 06 9F 00 08 07 9F 00 1A 07 9F 00 2A 07 9F 00	················
00460BEC	3B 07 9F 00 5A 07 9F 00 74 07 9F 00 85 07 9F 00	················
00460BFC	96 07 9F 00 A7 07 9F 00 85 07 9F 00 C4 07 9F 00	················
00460B0C	D3 07 9F 00 F3 07 9F 00 01 08 9F 00 24 08 9F 00	················
00460B1C	35 08 9F 00 46 08 9F 00 58 08 9F 00 68 08 9F 00	················
00460B2C	79 08 9F 00 98 08 9F 00 B2 08 9F 00 C3 08 9F 00	················
00460B3C	04 08 9F 00 E5 08 9F 00 F3 08 9F 00 02 09 9F 00	················
00460B4C	11 09 9F 00 31 09 9F 00 3F 09 9F 00 62 09 9F 00	················
00460B5C	73 09 9F 00 84 09 9F 00 96 09 9F 00 A6 09 9F 00	················
00460B6C	B7 09 9F 00 D6 09 9F 00 F0 09 9F 00 01 0A 9F 00	················
00460B7C	12 0A 9F 00 23 0A 9F 00 31 0A 9F 00 40 0A 9F 00	················
00460B8C	4F 0A 9F 00 6F 0A 9F 00 7D 0A 9F 00 A0 0A 9F 00	················
00460B9C	B1 0A 9F 00 C2 0A 9F 00 D4 0A 9F 00 C0 4B 0F 77	················
00460BAC	3B 4C 0F 77 94 A5 11 77 59 4B 0F 77 82 4E 0F 77	················
00460BCC	3F 50 0F 77 D9 66 0F 77 50 4B 0F 77 55 4C 0F 77	················
00460BDC	C2 4B 0F 77 95 D2 11 77 80 5D 15 77 00 00 00 00	················
00460BEC	00 00 A7 00 11 00 A7 00 22 00 A7 00 33 00 A7 00	················
00460BFC	00 00 A0 00 11 00 A0 00 22 00 A0 00 33 00 A0 00	················

00466C00	0000	ADD BYTE PTR DS:[EAX],AL	
00466C0F	0000	ADD BYTE PTR DS:[EAX],AL	
00466C11	0000	ADD BYTE PTR DS:[EAX],AL	

00465000=UnPackMe.00465000

Address	Hex dump	ASCII
00460BAC	ED 23 61 8C BB 96 AD 21 5F 45 86 73 E3 8D 73 81	Y#a!qû!_E\$sd!sü
00460BCC	82 0C 04 1D C9 6C E2 B3 BB 7D E9 F4 9F D8 C3 D7	ó·♦#P!0!q!ú!ñ!f!i!i
00460BCC	81 03 46 64 8F FD CB 4A BC 6F 01 BF CC EC 37 C8	ü♦FdA²rJ!o0h!rj7!e
00460BDC	86 B3 6B 2F A9 6B A3 2B 1A 47 19 6F 9C C1 62 DF	ä!k/0kû+G+o6!t!e
00460BEC	EE 0E 92 B6 66 EF BE F9 33 D3 FA 6E 21 B0 49 BC	"#Eaf'·#·3E·n!·I²

对 460BAC 这 4 个字节设置内存写入断点,运行起来,看看壳何时会将正确的 IAT 项填充进来。

004665B1	8907	MOV DWORD PTR DS:[EDI],EAX	
004665B3	58	POP EAX	
004665B4	48	DEC EAX	
004665B5	74 0D	JE SHORT 004665C4	UnPackMe.004665C4
004665B7	40	INC EAX	
004665B8	F8	CLC	
004665B9	66:8943 FE	MOV WORD PTR DS:[EBX-2],AX	
004665BD	8803	MOV BYTE PTR DS:[EBX],AL	
004665BF	43	INC EBX	

断在了这里,正确的 IAT 值将被写入。

Registers (FPU)				
EAX	770F4C3B			
ECX	7C929AEB	ntdll.7C929AEB		
EDX	7C98C0D8	ntdll.7C98C0D8		
EBX	00000007			
ESP	0012FFA0			
EBP	0005995E			
ESI	004600DC	UnPackMe.004600DC		
EDI	00460BAC	UnPackMe.00460BAC		
EIP	004665B1	UnPackMe.004665B1		
C	0	ES	0023	32bit 0(FFFFFFFF)
P	0	CS	001B	32bit 0(FFFFFFFF)
A	0	SS	0023	32bit 0(FFFFFFFF)

我们可以看到此时 EAX 寄存器中可能包含的是一个 API 函数的入口地址,虽然这里 OD 没有提示该 API 函数的名称,但是我们可以 Goto 过去看一看,我们 CTRL + G 输入 EAX。

Enter expression to follow

EAX

OK

770F4C39	90	NOP
770F4C3A	90	NOP
770F4C3B	8BFF	MOV EDI,EDI
770F4C3D	55	PUSH EBP
770F4C3E	8BEC	MOV EBP,ESP
770F4C40	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]
770F4C43	85C0	TEST EAX,EAX
770F4C45	74 05	JE SHORT 770F4C4C
770F4C47	8B40 FC	MOV EAX,DWORD PTR DS:[EAX-4]
770F4C4A	D1E8	SHR EAX,1
770F4C4C	5D	POP EBP
770F4C4D	C2 0400	RETN 4
770F4C50	90	NOP
770F4C51	90	NOP

我们到了这里,我们可以看到寄存器窗口中 EAX 显示处了该 API 函数的名称,嘿嘿。

Registers (FPU)				
EAX	770F4C3B	OLEAUT32.SysStringLen		
ECX	7C929AEB	ntdll.7C929AEB		
EDX	7C98C0D8	ntdll.7C98C0D8		
EBX	00000007			
ESP	0012FFA0			
EBP	0005995E			
ESI	004600DC	UnPackMe.004600DC		
EDI	00460BAC	UnPackMe.00460BAC		
EIP	004665B1	UnPackMe.004665B1		
C	0	ES	0023	32bit 0(FFFFFFFF)
P	0	CS	001B	32bit 0(FFFFFFFF)
A	0	SS	0023	32bit 0(FFFFFFFF)

接下来的任务就是需要将 IAT 项被重定向的流程修改为正确 IAT 项的处理流程。

让所有 IAT 项走正常流程

我们依然从循环的起始位置开始跟。

0046638A	8B95 62D34000	MOV EDX,DWORD PTR SS:[EBP+40D362]	UnPackMe.00400000
00466390	8B06	MOV EAX,DWORD PTR DS:[ESI]	
00466392	85C0	TEST EAX,EAX	
00466394	75 0B	JNZ SHORT 004663A1	UnPackMe.004663A1
00466396	8B46 10	MOV EAX,DWORD PTR DS:[ESI+10]	
00466399	85C0	TEST EAX,EAX	
0046639B	0F84 46FFFFFF	JE 004663E7	UnPackMe.004662E7
004663A1	03C2	ADD EAX,EDX	
004663A3	0385 4ED34000	ADD EAX,DWORD PTR SS:[EBP+40D34E]	
004663A9	8B18	MOV EBX,DWORD PTR DS:[EAX]	
004663AB	F7C3 00000000	TEST EBX,00000000	
004663B1	74 06	JE SHORT 004663B9	UnPackMe.004663B9
004663B3	8120 00000000	AND DWORD PTR DS:[EAX],00000000	
004663B9	8B7E 10	MOV EDI,DWORD PTR DS:[ESI+10]	
004663BC	03FA	ADD EDI,EDX	
004663BE	80A5 D6CC4000	AND BYTE PTR SS:[EBP+40CCD6],0FF	
004663C5	0F84 30010000	JE 004664FB	UnPackMe.004664FB
004663CB	80A5 D7CC4000	AND BYTE PTR SS:[EBP+40CCD7],0FF	
004663D2	0F84 20010000	JE 004664FB	UnPackMe.004664FB
004663D8	39BD 5AD44000	MOV DWORD PTR SS:[EBP+40D45A],EDI	
004663DE	8B85 52D44000	MOV EAX,DWORD PTR SS:[EBP+40D452]	
004663E4	40	INC EAX	
004663E5	0F84 10010000	JE 004664FB	UnPackMe.004664FB
004663EB	43	DEC EAX	
004663EC	0F85 B2000000	JNZ 004664A4	UnPackMe.004664A4
004663F2	60	PUSHAD	

0046638A	8B95 62D34000	MOV EDX,DWORD PTR SS:[EBP+40D362]	
00466390	8B06	MOV EAX,DWORD PTR DS:[ESI]	
00466392	85C0	TEST EAX,EAX	
00466394	75 0B	JNZ SHORT 004663A1	UnPackMe.004663A1
00466396	8B46 10	MOV EAX,DWORD PTR DS:[ESI+10]	
00466399	85C0	TEST EAX,EAX	
0046639B	0F84 46FFFFFF	JE 004662E7	UnPackMe.004662E7
004663A1	03C2	ADD EAX,EDX	
004663A3	0385 4ED34000	ADD EAX,DWORD PTR SS:[EBP+40D34E]	

这里跟 IAT 项重定向的流程是一致的。

004663A7	8B18	MOV EBX,DWORD PTR DS:[EAX]	
004663AB	F7C3 00000000	TEST EBX,00000000	
004663B1	74 06	JE SHORT 004663B9	UnPackMe.004663B9
004663B3	8120 00000000	AND DWORD PTR DS:[EAX],00000000	
004663B9	98B7E 10	MOV EDI,DWORD PTR DS:[ESI+10]	
004663BC	03FA	ADD EDI,EDX	
004663BE	80A5 D6CC4000	AND BYTE PTR SS:[EBP+40CCD6],0FF	

这里只是一个小跳转,我们继续跟。

004663B7	8B7E 10	MOV EDI,DWORD PTR DS:[ESI+10]	
004663BC	03FA	ADD EDI,EDX	
004663BE	80A5 D6CC4000	AND BYTE PTR SS:[EBP+40CCD6],0FF	
004663C5	0F84 30010000	JE 004664FB	
004663CB	80A5 D7CC4000	AND BYTE PTR SS:[EBP+40CCD7],0FF	
004663D2	0F84 23010000	JE 004664FB	
004663D8	89BD 5AD44000	MOV DWORD PTR SS:[EBP+40D45A],EDI	
004663DE	8B85 52D44000	MOV EAX,DWORD PTR SS:[EBP+40D452]	

这个跳转不成立。

004663D7	03FA 10	ADD EDI,EDX	
004663BC	03FA	ADD EDI,EDX	
004663BE	80A5 D6CC4000	AND BYTE PTR SS:[EBP+40CCD6],0FF	
004663C5	0F84 30010000	JE 004664FB	UnPackMe.004664FB
004663CB	80A5 D7CC4000	AND BYTE PTR SS:[EBP+40CCD7],0FF	
004663D2	0F84 23010000	JE 004664FB	UnPackMe.004664FB
004663D8	89BD 5AD44000	MOV DWORD PTR SS:[EBP+40D45A],EDI	
004663DE	8B85 52D44000	MOV EAX,DWORD PTR SS:[EBP+40D452]	
004663E4	40	INC EAX	
004663E5	0F84 10010000	JE 004664FB	UnPackMe.004664FB
004663EB	48	DEC EAX	
004663EC	0F85 B2000000	JNZ 004664A4	UnPackMe.004664A4
004663F2	60	PUSHAD	

这里跟之前 IAT 项被重定位的处理有明显区别,跳过的中间的大片代码。

004663BE	80A5 D6CC4000	AND BYTE PTR SS:[EBP+40CCD6],0FF	
004663C5	0F84 30010000	JE 004664FB	
004663CB	80A5 D7CC4000	AND BYTE PTR SS:[EBP+40CCD7],0FF	
004663D2	0F84 23010000	JE 004664FB	
004663D8	89BD 5AD44000	MOV DWORD PTR SS:[EBP+40D45A],EDI	
004663DE	8B85 52D44000	MOV EAX,DWORD PTR SS:[EBP+40D452]	
004663E4	40	INC EAX	
004663E5	0F84 10010000	JE 004664FB	
004663EB	48	DEC EAX	
004663EC	0F85 B2000000	JNZ 004664A4	
004663F2	60	PUSHAD	
004663F3	8BF7	MOV ESI,EDI	
004663F5	2BC0	SUB EAX,EAX	
004663F7	40	INC EAX	
004663F8	833F 00	CMP DWORD PTR DS:[EDI],0	
004663FB	8D7F 04	LEA EDI,DWORD PTR DS:[EDI+4]	
004663FE	75 F7	JNZ SHORT 004663F7	
00466400	48	DEC EAX	
00466401	0F84 EC000000	JE 004664F3	
00466407	8BD8	MOV EBX,EAX	
00466409	6BC0 31	IMUL EAX,EAX,31	
0046640C	6A 04	PUSH 4	
0046640E	68 00100000	PUSH 1000	
00466413	50	PUSH EAX	
00466414	6A 00	PUSH 0	
00466416	FF95 ECBA4000	CALL DWORD PTR SS:[EBP+40BAEC]	
0046641C	85C0	TEST EAX,EAX	
0046641E	0F84 CF000000	JE 004664F3	
00466424	8BFE	MOV EDI,ESI	
00466426	8BCB	MOV ECX,EBX	
00466428	8BF8	MOV EDI,EAX	
0046642A	8985 56D44000	MOV DWORD PTR SS:[EBP+40D456],EAX	
00466430	8BCB	MOV ECX,EBX	
00466432	6BD8 29	IMUL EBX,EBX,29	
00466435	03DF	ADD EBX,EDI	
00466437	891C24	MOV DWORD PTR SS:[ESP],EBX	
0046643A	B0 B8	MOV AL,0B8	
0046643C	6A 00	PUSH 0	
0046643E	50	PUSH EAX	
0046643F	53	PUSH EBX	
00466440	0FB74424 08	MOVZX EAX,WORD PTR SS:[ESP+8]	
00466445	50	PUSH EAX	
00466446	8D85 14BB4000	LEA EAX,DWORD PTR SS:[EBP+40BB14]	
0046644C	0FB618	MOVZX EBX,BYTE PTR DS:[EAX]	
0046644F	FF0C24	DEC DWORD PTR SS:[ESP]	
00466452	7E 09	JLE SHORT 0046645D	
00466454	40	INC EAX	

Jump is taken

我们可以看到这里是一个长跳转,跳转与否取决于 IAT 项是否被重定向,也就是说这里是决定是 IAT 项是否被重定向的关键跳,我们可以将这个条件跳转替换为 JMP。

但是这个跳转在程序一开始的时候并不存在,我们重启程序。

004663D2	E8 4278605E	CALL 5EA60C19	
004663D7	7E CF	JLE SHORT 004663A8	UnPackMe.004663A8
004663D9	3095 64EA3501	XOR BYTE PTR SS:[EBP+135EA641],DL	
004663DF	E6 D3	OUT 0D3,AL	I/O command
004663E1	9C	PUSHFD	
004663E2	BC 2F99553C	MOV ESP,3C55992F	
004663E3	83 05000000	MOV ESI,0	

我们可以看到在程序刚开始的时候,这里都是一些垃圾指令,壳会随后的某个时间点写入实际的功能代码。

004663D2	E8 4278605E	CALL 5EA60C19	
004663D7	7E CF	JLE Backup	UnPa
004663D9	3095 64EA3501	XOR	I/O
004663DF	E6 D3	OUT	
004663E1	9C	PUSH	
004663E2	BC 2F99553C	MOV	Binary
004663E3	A2 9E8CD06B	MOV	
004663E4	82E9 27	SUB	Assemble
004663E5	9D	POPF	Space
004663E6	8751 A5	XCHG	Label
004663E7	9D	POPF	:
004663E8	65:2F	DAS	Comment
004663E9	4D	DEC	;
004663EA	72 44	JB 3	Sup
004663EB	56	PUSH	UnPa
004663EC	2D 6183EFFE	SUB	
004663ED	5E	POP	
004663EE	07	POP	Mod i
004663EF	72 93	JB 3	UnPa
004663F0	8D55 C4	LEA	
004663F1	F8	CLC	
004663F2	D0F3	SAL	
004663F3	07	POP	
004663F4	C2 D39F	RET	

我们转到数据窗口,由于硬件断点会被检测,INT3 断点这里不能用,所以我们可以使用内存写入断点,我们对 460818~460f28 这片 IAT 区域设置内存写入断点。

00466BFF	0000	ADD	Label	:	
00466C01	004B 6C	ADD			
00466C04	06	PUSH	Breakpoint		Memory, on access
00466C05	0036	ADD			Memory, on write
00466C07	6C	INS	Search for		Hardware, on access
00466C08	06	PUSH	Find references	Ctrl+R	Hardware, on write
00466C09	0000	ADD	View executable file		Hardware, on execution
00465000=UnPackMe.00465000					

这里我们对所有的 IAT 项都设置上了内存写入断点,当第一个 IAT 项被写入的时候就会断下来。

00465D91	AA	STOS BYTE PTR ES:[EDI]	
00465D92	69D2 A5B0CD4B	IMUL EDX,EDX,4BCDB0A5	
00465D98	F9	STC	
00465D99	72 02	JB SHORT 00465D9D	UnP
00465D9B	CD 20	INT 20	
00465D9D	D1C2	ROL EDX,1	
00465D9F	69DB 701FEE6A	IMUL EBX,EBX,6AEE1F70	
00465DA5	8300	AND EBX,EBX	

断在了这里,这里的 STOS 指令会执行多次,直到整个 IAT 都被填充完毕为止,此时关键跳已经存在了。

004663D2	0F84 23010000	JE 004664FB	UnPackMe.004664FB
004663D8	89BD 5AD44000	MOV DWORD PTR SS:[EBP+40D45A],EDI	
004663DE	8B85 52D44000	MOV EAX,DWORD PTR SS:[EBP+40D452]	
004663E4	40	INC EAX	
004663E5	0F84 10010000	JE 004664FB	UnPackMe.004664FB
004663EB	48	DEC EAX	
004663EC	0F85 B2000000	JNZ 004664A4	UnPackMe.004664A4
004663F2	60	PUSHAD	
004663F3	8BF7	MOV ESI,EDI	
004663F5	2BC0	SUB EAX,EAX	
004663F7	40	INC EAX	
004663F8	833F 00	CMP DWORD PTR DS:[EDI],0	
004663FB	8D7F 04	LEA EDI,DWORD PTR DS:[EDI+4]	
004663FE	75 F7	JNZ SHORT 004663F7	UnPackMe.004663F7
00466400	48	DEC EAX	
00466401	0F84 EC000000	JE 004664F3	UnPackMe.004664F3
00466407	8BD8	MOV EBX,EAX	
00466409	6BC0 31	IMUL EAX,EAX,31	
0046640C	6A 04	PUSH 4	
0046640F	50	PUSH 0	

这里我们可以看到关键跳已经生成了,我们将该关键跳转 JE 替换成 JMP,接着清除之前设置的内存断点,接着跟踪到 OEP 处。

004663D2	E9 24010000	JMP 004664FB	UnPackMe.004664FB
004663D7	90	NOP	
004663D8	89BD 5AD44000	MOV DWORD PTR SS:[EBP+40D45A],EDI	
004663DE	8B85 52D44000	MOV EAX,DWORD PTR SS:[EBP+40D452]	
004663E4	40	INC EAX	

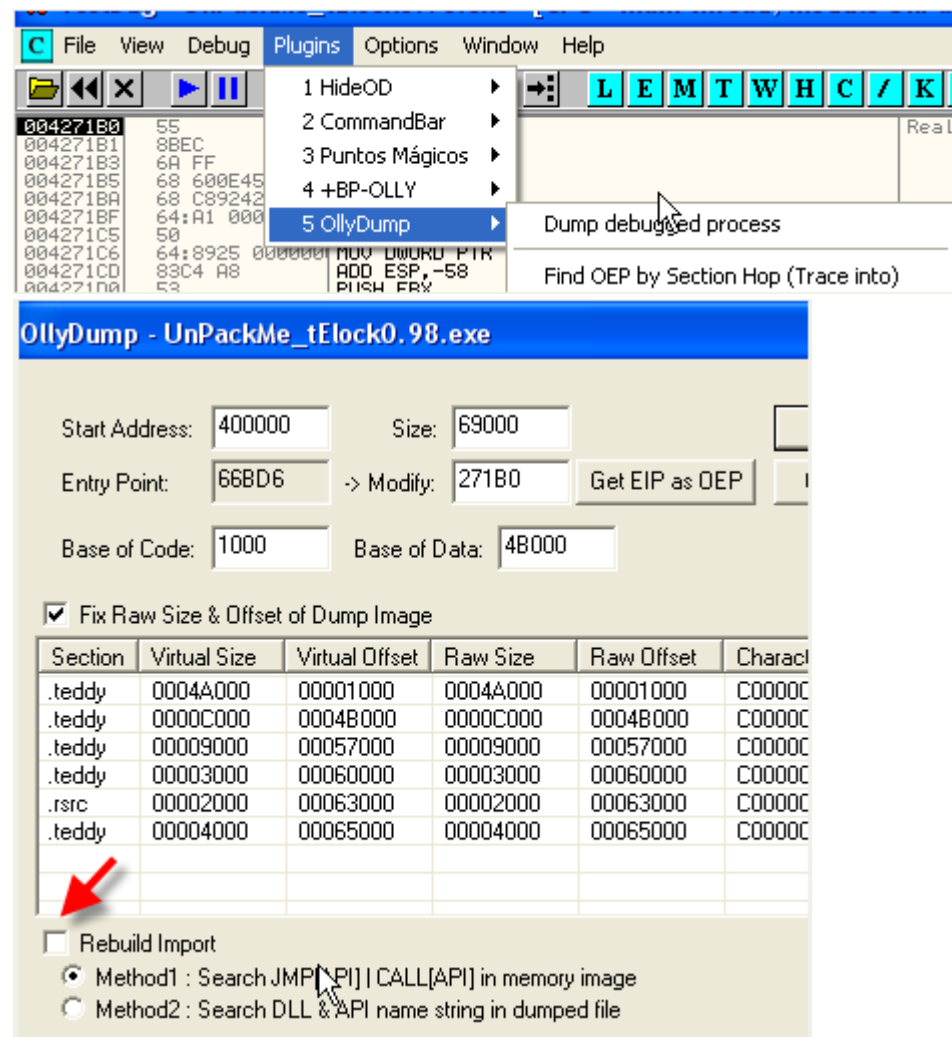
这里有两种可能,一种就是壳有自校验,运行起来会失败。另一种就是运行很正常,这里我们顺利的跟踪到了 OEP 处。

004271B0	55	PUSH EBP	Real entry point of SFX code
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 DC0A4600	CALL DWORD PTR DS:[460ADC]	kernel32.GetVersion
004271DC	33D2	XOR EDX,EDX	
004271DE	8AD4	MOV DL,AH	
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX	
004271F6	8BC8	MOV ECX,ECX	

现在到了 OEP 处,我们再来看看 IAT:

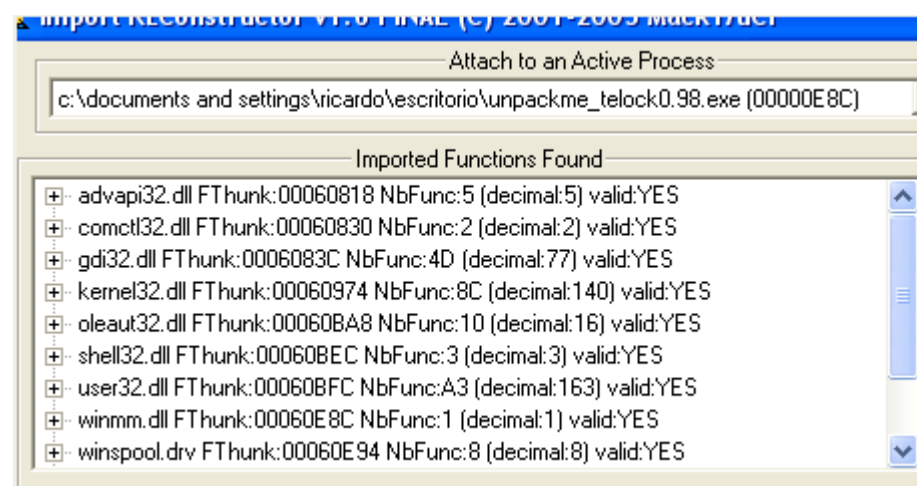
Address	Hex dump	ASCII
004607FC	78 2B 06 00 60 2B 06 00 4C 2B 06 00 2E 2B 06 00	x+*,*+,L+*,*+,
00460800	00 00 00 00 00 00 00 00 00 00 00 00 F0 6B DA 77C....-k rw
0046081C	1B 76 DA 77 F4 EA DA 77 E7 EB DA 77 33 78 DA 77	+Urw00 rw00 rwax rw
0046082C	00 00 00 00 00 15 C5 58 2E B0 C3 58 00 00 00 00	...!\$+X.c fX...
0046083C	04 6A EF 77 66 95 EF 77 89 6A EF 77 F3 AD EF 77	Ej'wfo'we'j'w'w'w
0046084C	ED D9 EF 77 99 88 EF 77 C8 B5 EF 77 2A 7D EF 77	~'w01'w'w'w'w'w
0046085C	B2 7C EF 77 53 82 EF 77 1E C9 F1 77 0C BC EF 77	81'wW5=AAf'w'w'w
0046086C	52 D4 EF 77 FA 80 EF 77 F1 D0 EF 77 51 B2 EF 77	8E'w'w'w'w'w'w'w
0046087C	26 D5 EF 77 2A E3 EF 77 5F 39 F2 77 71 B4 EF 77	8'w*0'w'w'w'w'w
0046088C	2E AD EF 77 E1 61 EF 77 B8 85 EF 77 CC D2 EF 77	.i'wpa'w0a'wfe'w
0046089C	43 70 EF 77 FB EA F0 77 12 83 EF 77 01 72 F0 77	Cp'w'0-w0a'w0x-w
004608AC	A9 34 F0 77 D5 93 EF 77 68 EF 77 AA D2 EF 77	04-w'0'wh'w'w'e'w
004608BC	B2 6F EF 77 3F 38 F2 77 D6 E8 EF 77 68 E0 EF 77	80'w?8=wi0'wh0'w
004608CC	00 60 EF 77 90 5B EF 77 6D AC EF 77 94 6C F0 77	..i'we['wn%w0l-w
004608DC	22 8D EF 77 3D C8 F1 77 3D 6D F0 77 6F C0 EF 77	'i'w=8=wn-w0l-w
004608EC	85 7B EF 77 26 D9 EF 77 FB 5E EF 77 36 8A EF 77	3c'w8-w'w'w'w'e'w
004608FC	FC 8A EF 77 0F 62 EF 77 49 5E EF 77 97 5D EF 77	2e'w*0'wI'w'w'w'w
0046090C	1A 9A EF 77 6B FA EF 77 7B C9 F0 77 DA 98 F2 77	+0'wk'w(f-w'w'w'w
0046091C	1A 40 F2 77 55 EA EF 77 C5 61 EF 77 70 E6 EF 77	+0=wU0'w'a'wpu'w
0046092C	F0 81 EF 77 2D 6C EF 77 38 6E EF 77 4F 83 EF 77	-u'w-l'w'w'n'w0a'w
0046093C	09 ED EF 77 EB AA EF 77 26 69 F0 77 B1 95 EF 77	~'w0-w'w0i-w00'w
0046094C	6F B0 EF 77 8A 5A EF 77 E9 49 F2 77 26 F1 F0 77	C0'we2'w0l-w0z-w
0046095C	C9 D0 F0 77 51 E0 F0 77 33 8C EF 77 6C EC EF 77	f'i-w00-w0l'w'ly'w
0046096C	29 94 EF 77 00 00 00 00 6B 17 80 7C C1 C9 80 7C	0'w....k0'w'f0'w
0046097C	69 10 81 7C EE 1E 80 7C 8D 2C 81 7C 40 7A 94 7C	0'u'w'w'w'w'w'w'w
0046098C	E1 EA 81 7C A2 CA 81 7C 16 1E 80 7C 43 99 80 7C	00u'0a'u'w'w'w'w'w
0046099C	10 11 81 7C 29 29 81 7C 14 9B 80 7C 81 9A 80 7C	0'u'w'w'w'w'w'w'w
004609AC	FB 2C 82 7C AE 94 83 7C 2B 2E 83 7C C4 CE 80 7C	'e'w'w'w'w'w'w'w
004609BC	8A 2B 86 7C 3F DC 81 7C 5F 48 81 7C 23 CC 81 7C	e+a'0a'u'w'w'w'w'w
004609CC	78 2C 81 7C 86 03 81 7C B9 8C 83 7C 80 A4 80 7C	x,u'0a'u'w'w'w'w'w
004609DC	57 BB 80 7C 7F D4 80 7C 72 17 81 7C 93 D2 80 7C	00C'w'w'w'w'w'w'w

我们可以看到所有的 IAT 项都是正常的,我们成功修复了 IAT,现在我们可以进行 dump 了。

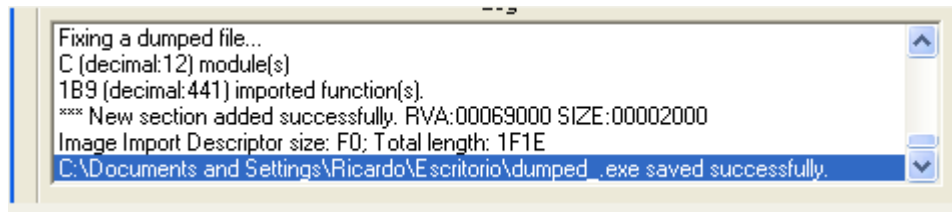


这里我们不勾选 ReBuild Import,dump 出来。

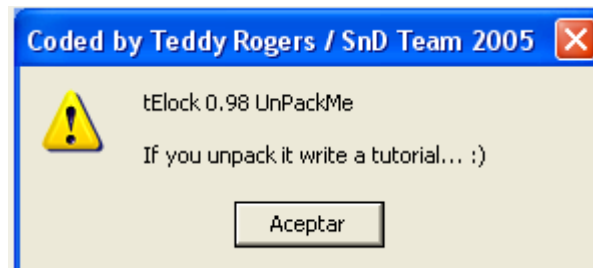
接着打开 IMP REC,填入 OEPRVA,SIZE 等数据,单击 Get Imports。



我们可以看到所有的项都是有效的,说明刚刚修改的关键跳起作用了,接着单击 Fix dump 修复刚刚的 dump 文件。



修复过的 dump 文件被重命名为了 dumped_.exe,我们运行一下看看效果。



嘿嘿,完美运行。

不同壳关键跳的地方也不一样,但是我们细心对比正常 IAT 项与重定向过的 IAT 项的处理流程差异,总能找到一点蛛丝马迹,好了,本章就讨论到这里,下一章我们继续讨论别的壳。