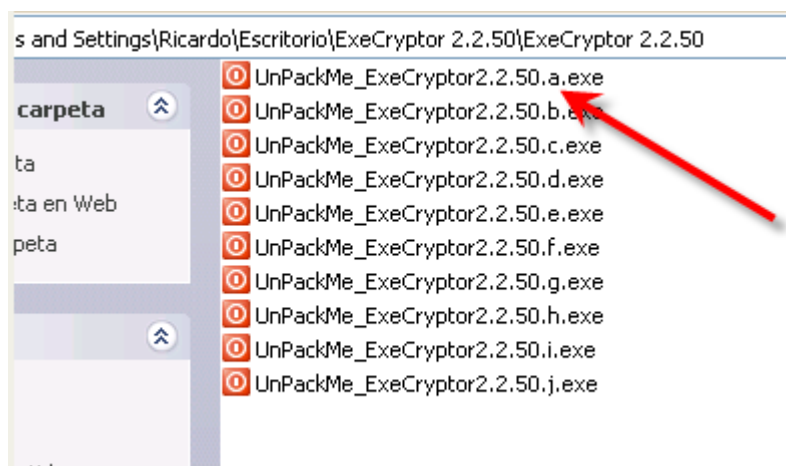


第五十四章-EXECryptor v2.2.50.a 脱壳-Part1

当我们遇到一款之前没有分析过的壳的时候,不要盲目的下手,我们最好先在网上搜一下该壳相关的 UnPackMe。如果有相关的 UnPackMe 的话,我们可以将 UnPackMe 与目标程序对照着来分析。如果实在找不到相关的 UnPackMe 的话,我们可以去下载一个该壳的加壳器,然后我们找一个比较简单的小程序(PS:或者自己编写一个小程序也可以,有源代码最好不过了)来作为加壳的对象。接着我们使用该壳的加壳器对我们的小程序进行加壳,我们逐一选择不同的加密强度,从最低保护强度到最高保护强度。接下来逐一分析该壳不同的加密选项都有什么不同,等我们把该壳的各个保护手段都研究透彻了以后,再来分析我们最初的目标程序,就会容易很多。就算以后该壳发布了新的版本,我们有了对其之前版本的深入理解,再来分析其最新版本应该也不会太困难,因为通常来说,新版本的改动不会很大。

我已经为大家准备好了 EXECryptor 的一系列 UnPackMe,虽然不是 EXECryptor 最新的版本,但是对于研究 EXECryptor 的保护机制已经足够了。(PS:EXECryptor 在当年来说算的上一款猛壳)

这里我们可以看到不同等级的 UnPackMe,难度逐一递增。

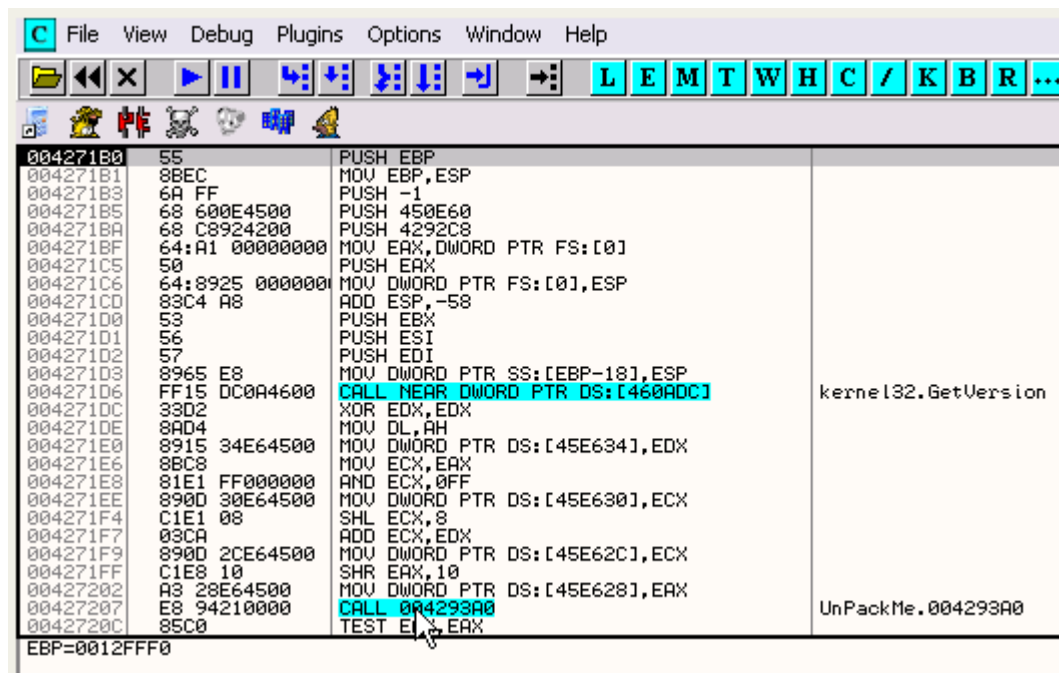


本章我们的目标程序是上图中的这个 UnPackMe_Execryptor2.2.50.a.exe。我们直接运行该程序,在弹出的对话框中可以看到保护措施。



我们可以看到这个等级的加密强度几乎为零,只是简单的压缩代码/数据/资源,跟 UPX 壳的做法很比较相似。这里如果我们将 UPX 的 UnPackMe 与该程序对照着来分析的话,就可以很容易的得知其 OEP 以及 IAT 的起始地址,大小。

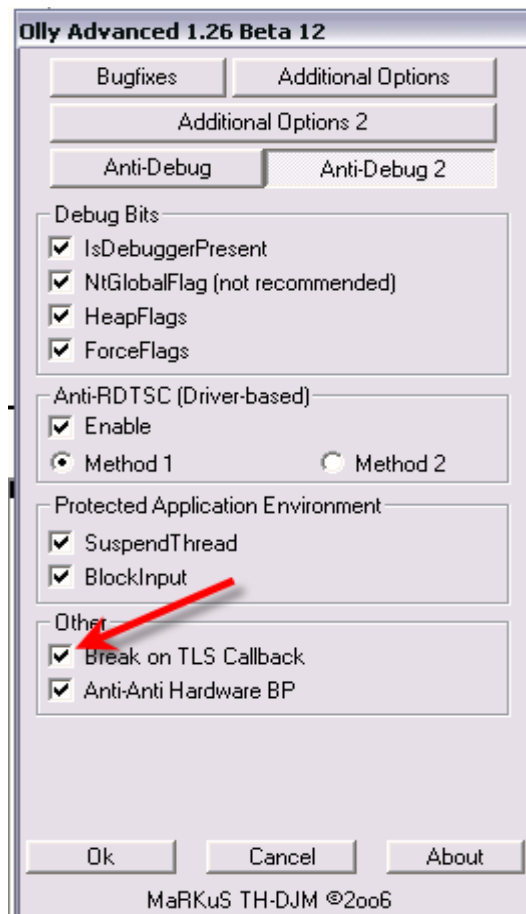
我们用 OD 加载 UPX 的 UnPackMe,断在了入口点处。



这个入口我们应该很熟悉了吧,可以说几乎本系列教程的每一个章节都可以看到。由于该 `UnPackMe_ExeCryptor2.2.50.a.exe` 与 `UnPackMe_UPX1.91.a.exe` 的原程序都是一样的,所以加壳以后,它们的 OEP,以及 IAT 也应该是一样的,它们调用的第一个 API 函数都是 `GetVersion`。我们可以得到这些基本的信息。

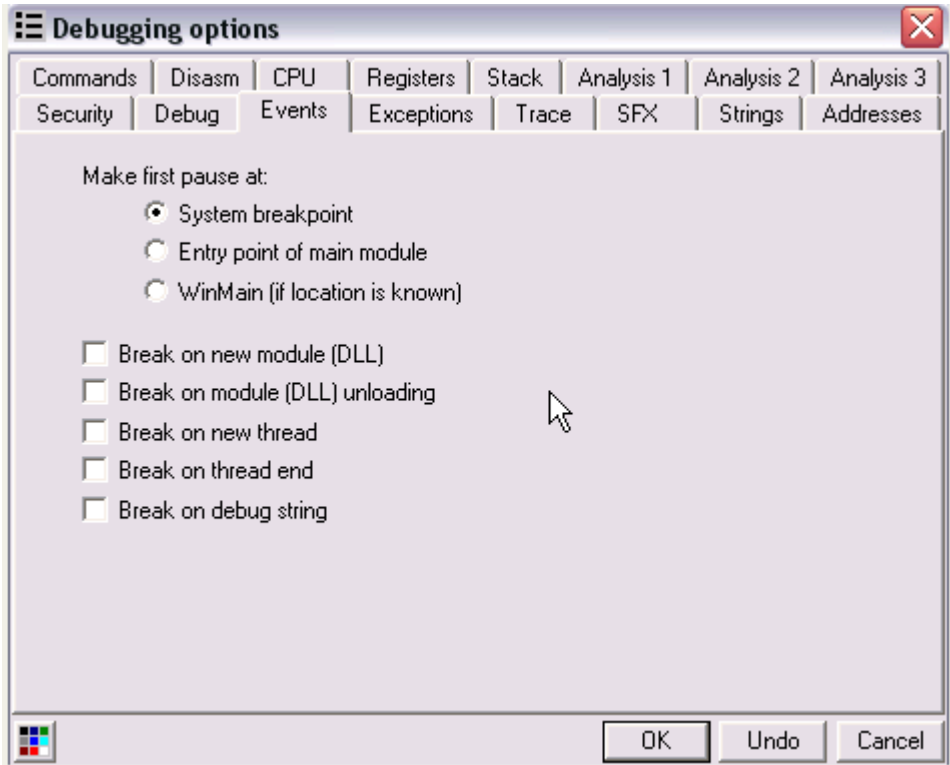
好,下面我们来分析 `UnPackMe_ExeCryptor2.2.50.a.exe`。

首先配置好 OllyAdvanced 这个反反调试插件。



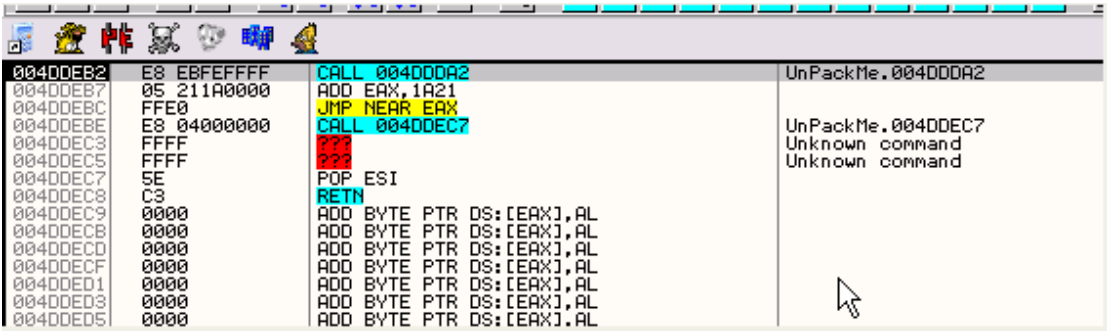
这里我用 Patched 4 这款 OD(我一般都是用这个版本),OllyAdvanced 这款插件里面有很多选项可供我们选择,其他的选项勾不勾选无所谓,但是 Break on TLS Callback 这个选项这里我们一定要记得勾选。

如果大家用 OD 加载 UnPackMe_ExeCryptor2.2.50.a.exe 的话,会发现还没有达到入口点程序就退出了。这是因为 ExeCryptor 利用了 TLS CALLBACK 这一特性在入口点之前执行代码-检测是否正在被调试,如果是,则退出进程。TLS CALLBACK 可以在入口点之前执行代码这个特性最初是由一个病毒作者发现的,后来被 ExeCryptor 的作者利用来进行反调试。

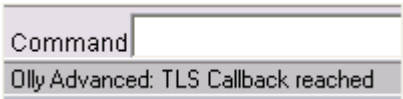


这里我们将首次中断的地方切换为 System breakpoint 处。

我们一运行起来就会断在 TLS CALLBACK 处。



我们可以看到 OD 状态栏上的提示。

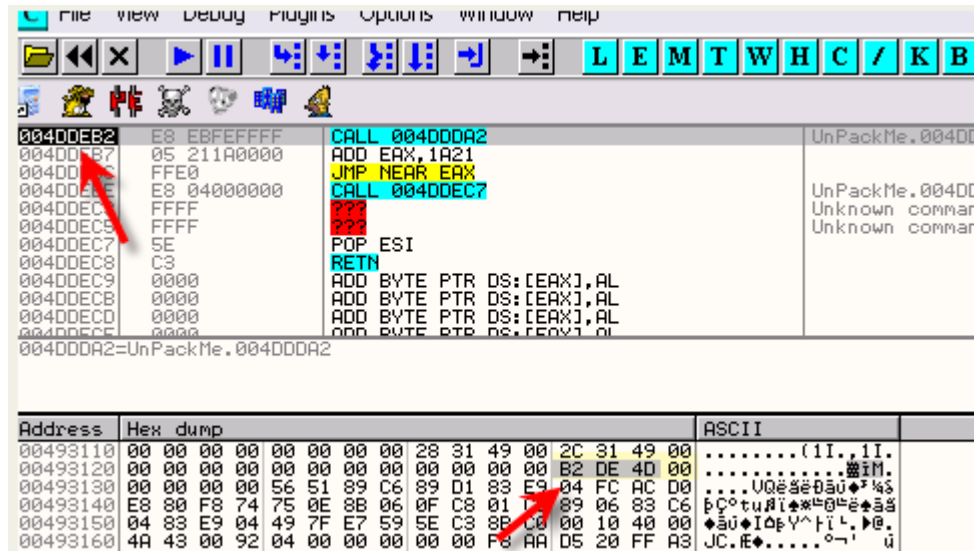


这是 OllyAdvanced 这个插件帮我们定位到的 TLS CALLBACK 回调函数的入口地址。下面我们来看看如何手工定位 TLS CALLBACK 回调函数的入口地址。

我们通过在数据窗口中按 CTRL+G 输入 40000 定位到 PE 头,然后单击鼠标右键选择-Special-PE header 将数据窗口的显示模式切换为 PE 解析模式,往下拉。

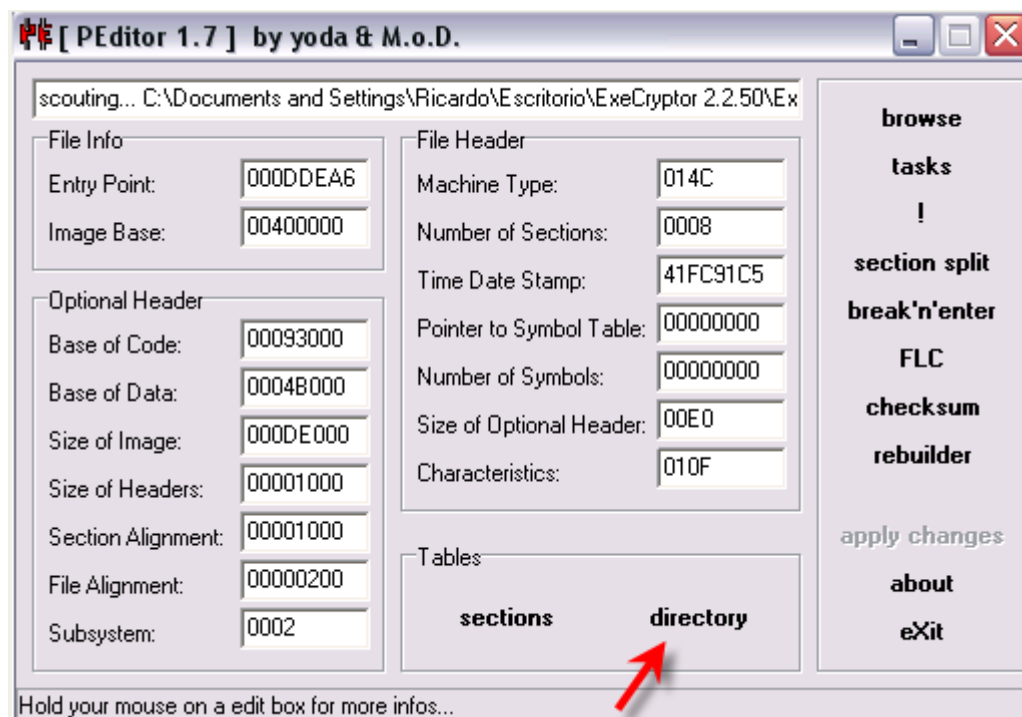
00400148	00000000	DD 00000000	Global Ptr address = 0
0040014C	00000000	DD 00000000	Must be 0
00400150	10310900	DD 00093110	TLS Table address = 93110
00400154	18000000	DD 00000018	TLS Table size = 18 (24.)
00400158	00000000	DD 00000000	Load Config Table address = 0
0040015C	00000000	DD 00000000	Load Config Table size = 0

这里我们可以看到 TLS Table address 为 93110(RVA),加上映像基址 400000 就得到了 493110,即 TLS TABLE 的起始地址。我们在数据窗口中定位到该地址,我们往下看就可以找到 TLS 回调函数的入口地址。

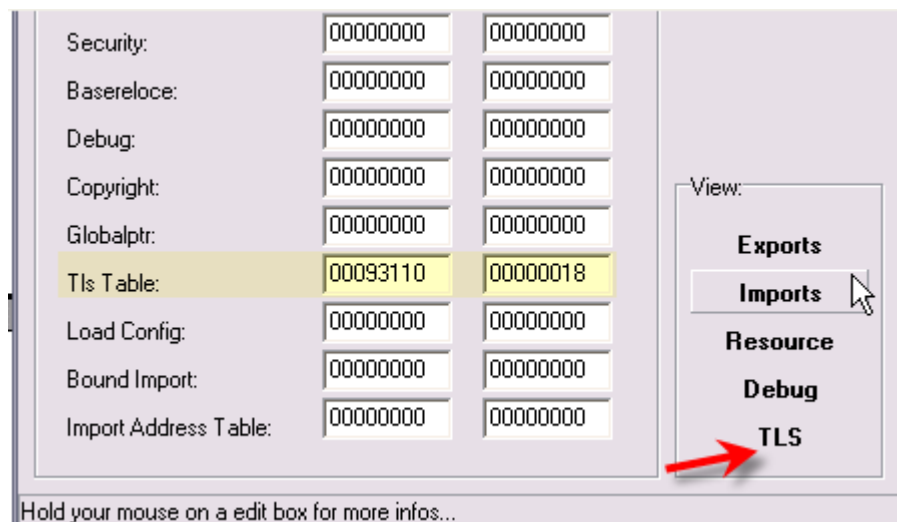


这里我们就定位到了 ExeCryptor 在到达入口点之前要执行代码的起始地址了,使用 OllyAdvanced 插件的话,它可以帮助我们直接定位到这个地址。

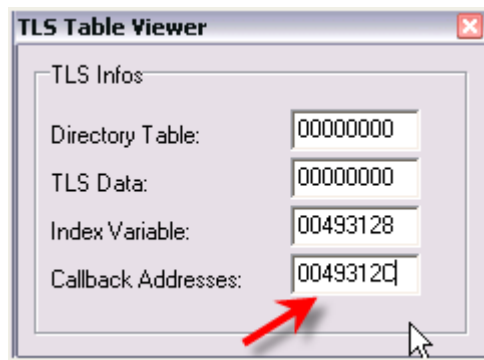
接下来我们来看看如何使用 PE 文件编辑器来定位 TLS 回调函数的入口地址。



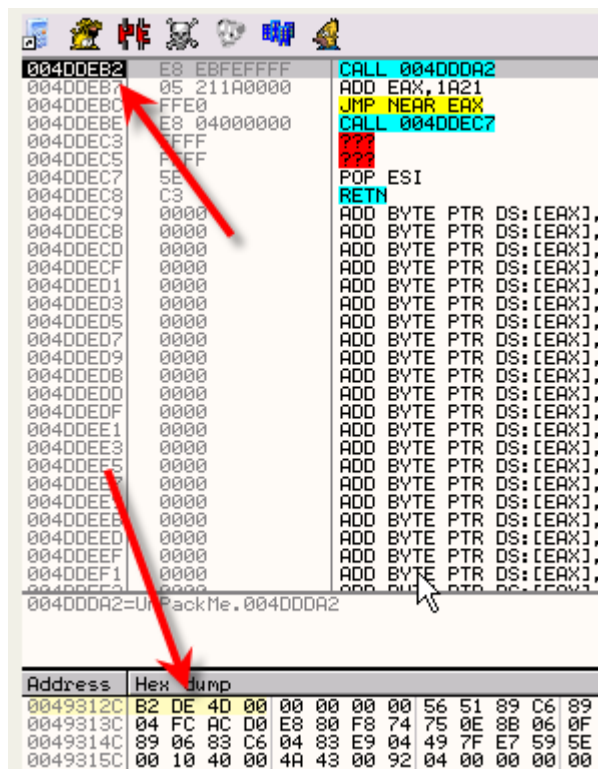
这里我们单击 directory 按钮查看数据目录。



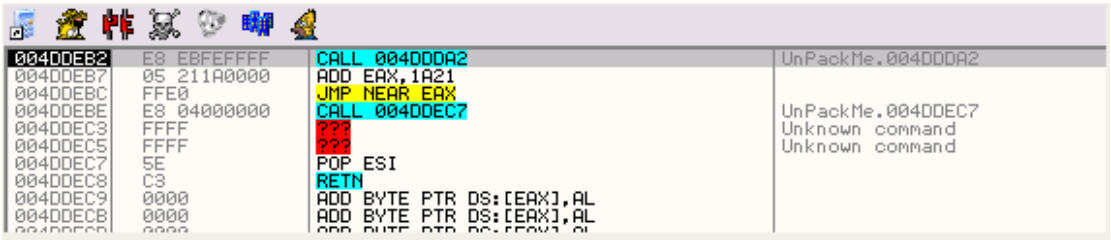
我们可以看到 TLS Table 起始地址的 RVA 为 93110,大小为 18。这右边还有个 TLS 按钮,单击该按钮我们就可以精确的查看 TLS 的回调函数入口地址的指针等信息。



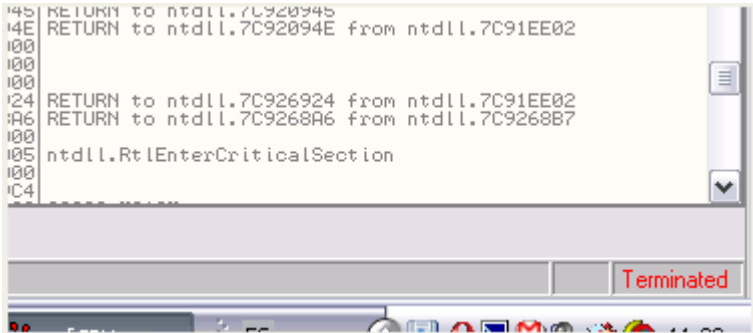
我们可以看到 TLS 回调函数的入口地址存放在 49312C 中,我们在数据窗口中定位到该地址。



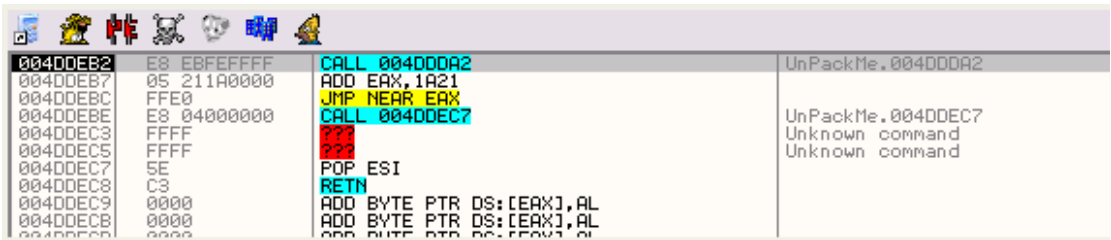
我们可以看到的的确是 TLS 回调函数的入口地址,嘿嘿。好了,现在我们就知道如何用 OD,PE 文件编辑器以及 OllyAdvanced 插件来定位 TLS 回调函数的入口地址了。使用 OllyAdvanced 插件的话,我们直接就可以断在 TLS 回调函数的入口地址处,如果是手工的话,我们首先要断在系统断点处,然后在 TLS 回调函数的入口处设置一个断点,然后运行起来,就可以断在 TLS 回调函数的入口处了。



我们运行起来。



这里我们可以看到壳已经检测到自己正在被调试,退出了进程。OllyAdvanced 插件里面的选项我们都勾选上了还被检测到,那我们



这里我们打开断点窗口查看一下,尽管我们之前并没有设置断点,我们可以看到这里有一个一次性断点。

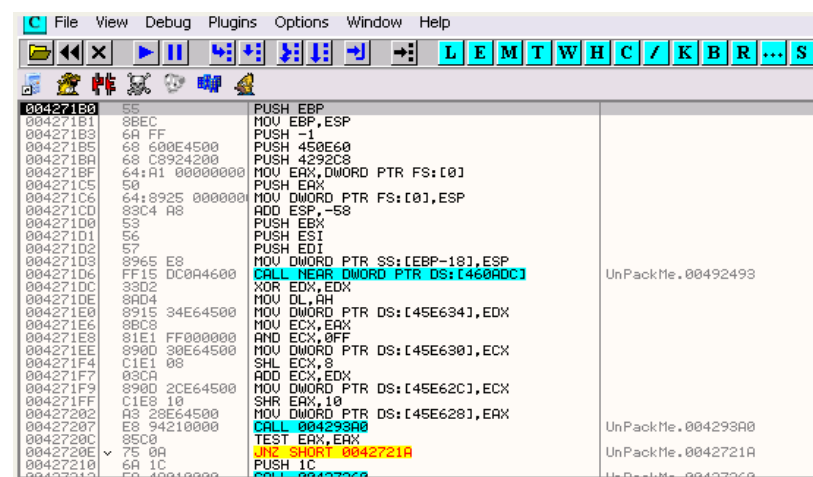
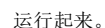
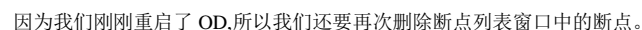
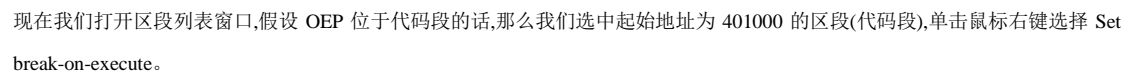
Address	Module	Active	Disassembly
0040DDEA6	UnPackMe	One-shot	CALL 0040DDA2

而此时壳在 TLS 回调函数中的检测代码还没有执行,当检测代码执行的时候,就会发现内存中有指令被替换成 CC,也就说明正在被调试,所以这里我们删除掉这个断点,然后运行起来,看看还会不会被检测到。



这里我们可以看到程序正常运行起来了,并没有退出。也就是说的确是这个断点被壳检测到了,才导致退出的。我们直接手动将该断点删除即可。如果遇到有的情况,删除了这个断点,还是退出的话,那么就是说除了 OllyAdvanced 插件里面的反反调试选项以

好了,现在问题我们已经解决了,我们重启 OD。



(PS:这里利用 OllyBone 这个插件的 Set break-on-execute 选项,我依然是怎么断也断不下来,无语球了,哈哈。等明年有时间我自己写个 break-on-execute 的插件吧!这里的话我就给大家介绍一下我断 OEP 的方法吧,嘿嘿。首先最后一次异常法大家就不要想了,因为压根就没有异常,哈哈。我呢,是用 OD 自带的 Set break-on-access 这个选项来定位 OEP 的,由于是访问断点,所以读取,写入,执行的时候都会断下来,所以肯定是没有 OllyBone 的 break-on-execute 快的,但是 OllyBone 不好用,我也没有办法。我们重启 OD,断在了 TLS CALLBACK 回调函数的入口处。

004DDEB2	E8 EBFEBFFF	CALL 004DDDA2	UnPackMe.004DDDA2
004DDEB7	05 211A0000	ADD EAX,1A21	
004DDEBC	FFEB	JMP NEAR EAX	
004DDEBE	E8 04000000	CALL 004DDEC7	UnPackMe.004DDEC7
004DDEC3	FFFF	???	Unknown command
004DDEC5	FFFF	???	Unknown command
004DDEC7	5E	POP ESI	
004DDEC8	C3	RETN	
004DDEC9	0000	ADD BYTE PTR DS:[EAX],AL	
004DDECB	0000	ADD BYTE PTR DS:[EAX],AL	
004DDECD	0000	ADD BYTE PTR DS:[EAX],AL	
004DDECF	0000	ADD BYTE PTR DS:[EAX],AL	

老规矩,删除掉断点列表窗口中的一次性断点。

Address	Module	Active	Disassembly	Comment
004DDEA6	UnPackMe	One-shot	CALL 004DDDA2	

Remove

Del

Follow in Disassembler Enter

Copy to clipboard

Appearance

接着给代码段设置 break-on-access 断点。

003B0000	00008000				Priv	RWE	RWE	
003B0000	00008000				Priv	RW	RW	
003C0000	00001000				Priv	RW	RW	
003D0000	00001000				Priv	RW	RW	
00400000	00001000	UnPackMe	PE header		Imag	R	RWE	
00401000	0004A000	UnPackMe	.text	code	Imag	R	RWE	
0044B000	0000C000	UnPackMe	.rdata	Actualize		R	RWE	
00457000	00009000	UnPackMe	.data	View in Disassembler	Enter		RWE	
00460000	00003000	UnPackMe	dyun	Dump in CPU		R	RWE	
00463000	00008000	UnPackMe	.rsrc	Dump		R	RWE	
0046B000	00001000	UnPackMe	xd4a	Search	CtrlR	R	RWE	
0046C000	00027000	UnPackMe	uwxw			R	RWE	
00493000	0004B000	UnPackMe	mlna	Set break-on-access	F2	R	RWE	
004E0000	00003000				R	E	R	E
005A0000	00002000				R	E	R	E
005B0000	00103000				R		R	
006C0000	00067000				R	E	R	E
62C20000	00001000	UnPackMe	LPK			R	RWE	

Set break-on-access F2

Set memory breakpoint on access

Set memory breakpoint on write

Set access

这里的大家要记住,break-on-access 是一次性断点,断下来了就没了,下次要用的话,还要设置一次。

003D0000	00001000				Priv	RW	RW	
00400000	00001000	UnPackMe		PE header	Imag	R	RWE	
00401000	0004A000	UnPackMe	.text	code	Imag	R	RWE	
0044B000	0000C000	UnPackMe	.rdata	code	Imag	R	RWE	
00457000	00009000	UnPackMe	.data	data	Imag	R	RWE	
00460000	00003000	UnPackMe	dyuntioj		Imag	R	RWE	
00463000	00008000	UnPackMe	.rsrc	resources	Imag	R	RWE	
0046B000	00001000	UnPackMe	xd4ambdu		Imag	R	RWE	
0046C000	00027000	UnPackMe	uwxwxip		Imag	R	RWE	
00493000	0004B000	UnPackMe	mlnaijnz	SFX,imports	Imag	R	RWE	
004E0000	00003000				Map	R	E	R
004E0000	00003000				Map	R	E	R

这里我们就设置完毕 break-on-access 断点了,我们可以看到该区段的起始地址被标注为红色了。

我们运行起来。

004DB303	AA	STOS BYTE PTR ES:[EDI]	
004DB304	EB E9	JMP SHORT 004DB2EF	UnPackMe.004DB2EF
004DB306	E8 FC000000	CALL 004DB407	UnPackMe.004DB407
004DB30B	0F82 97000000	JB 004DB3A8	UnPackMe.004DB3A8
004DB311	E8 F1000000	CALL 004DB407	UnPackMe.004DB407
004DB316	73 5B	JNB SHORT 004DB373	UnPackMe.004DB373
004DB318	B9 04000000	MOU ECX,4	
004DB31D	E8 FD000000	CALL 004DB41F	UnPackMe.004DB41F
004DB322	48	DEC EAX	
004DB323	74 DE	JE SHORT 004DB303	UnPackMe.004DB303
004DB325	0F89 C7000000	JNS 004DB3F2	UnPackMe.004DB3F2
004DB32B	E8 D7000000	CALL 004DB407	UnPackMe.004DB407
004DB330	73 1B	JNB SHORT 004DB34D	UnPackMe.004DB34D

断在了这里,从 OD 状态栏中的提示信息,我们可以知道这是由于写入导致的中断。

00457100	00 00 00 00	00 00 00 00	00 00 00 00	00 00 0
00457110	00 00 00 00	00 00 00 00	00 00 00 00	00 00 0
00457120	00 00 00 00	00 00 00 00	00 00 00 00	00 00 0
Command : <input type="text"/>				
Break-on-access when writing to [00401000]				

我们要的是执行导致的中断,而不是读取或者写入导致的中断。

这里大家不要盲目的再次设置 break-on-access 断点,然后按 F9 键直接运行起来。

如果基础好的童鞋的话,一眼就可以看出这里是一个循环,大家看出来没有?我的天!有童鞋说木有看出来!

我们用鼠标选中接下来的 4DB304 这个地址处的跳转指令。

004DB2EE	43	INC EBX	
004DB2EF	31C0	XOR EAX,EAX	
004DB2F1	E8 11010000	CALL 004DB407	
004DB2F6	73 0E	JNB SHORT 004DB306	
004DB2F8	8B4D F0	MOU ECX,DWORD PTR SS:[EBP-10]	
004DB2FB	E8 1F010000	CALL 004DB41F	
004DB300	0245 EF	ADD AL,BYTE PTR SS:[EBP-11]	
004DB303	AA	STOS BYTE PTR ES:[EDI]	
004DB304	EB E9	JMP SHORT 004DB2EF	
004DB306	E8 FC000000	CALL 004DB407	

看到没,出现了向上指的红色箭头,这款代码不是循环操作是什么?嘿嘿。那么怎么样跳过循环呢?很简单,直接对下一条语句即

4DB306 处设置一个断点,然后运行起来,就可以跳过这个循环了。但是大家不要慌,将代码往下拉,看看这不是一个嵌套循环。

004DB2E9	BA 00000000	MOU EDX,80000000	
004DB2EE	43	INC EBX	
004DB2EF	31C0	XOR EAX,EAX	
004DB2F1	E8 11010000	CALL 004DB407	UnPackMe.004DB407
004DB2F6	73 0E	JNB SHORT 004DB306	UnPackMe.004DB306
004DB2F8	8B4D F0	MOU ECX,DWORD PTR SS:[EBP-10]	
004DB2FB	E8 1F010000	CALL 004DB41F	UnPackMe.004DB41F
004DB300	0245 EF	ADD AL,BYTE PTR SS:[EBP-11]	
004DB303	AA	STOS BYTE PTR ES:[EDI]	
004DB304	EB E9	JMP SHORT 004DB2EF	UnPackMe.004DB2EF
004DB306	E8 FC000000	CALL 004DB407	UnPackMe.004DB407
004DB30B	0F82 97000000	JB 004DB3A8	UnPackMe.004DB3A8
004DB311	E8 F1000000	CALL 004DB407	UnPackMe.004DB407
004DB316	73 5B	JNB SHORT 004DB373	UnPackMe.004DB373
004DB318	B9 04000000	MOU ECX,4	
004DB31D	E8 FD000000	CALL 004DB41F	UnPackMe.004DB41F
004DB322	48	DEC EAX	
004DB323	74 DE	JE SHORT 004DB303	UnPackMe.004DB303
004DB325	0F89 C7000000	JNS 004DB3F2	UnPackMe.004DB3F2
004DB32B	E8 D7000000	CALL 004DB407	UnPackMe.004DB407
004DB330	73 1B	JNB SHORT 004DB34D	UnPackMe.004DB34D
004DB332	55	PUSH EBP	
004DB333	BD 00010000	MOU EBP,100	
004DB338	E8 D7000000	CALL 004DB414	UnPackMe.004DB414
004DB33D	8807	MOU BYTE PTR DS:[EDI],AL	
004DB33F	47	INC EDI	
004DB340	4D	DEC EBP	
004DB341	75 F5	JNZ SHORT 004DB338	UnPackMe.004DB338
004DB343	E8 BF000000	CALL 004DB407	UnPackMe.004DB407
004DB348	72 E9	JB SHORT 004DB333	UnPackMe.004DB333
004DB34A	5D	POP EBP	
004DB34B	EB A2	JMP SHORT 004DB2EF	UnPackMe.004DB2EF
004DB34D	B9 01000000	MOU ECX,1	
004DB352	E8 C8000000	CALL 004DB41F	UnPackMe.004DB41F
004DB357	83C0 07	ADD EAX,7	
004DB35A	8945 F0	MOU DWORD PTR SS:[EBP-10],EAX	
004DB35D	C645 EF 00	MOU BYTE PTR SS:[EBP-11],0	
004DB361	83F8 08	CMP EAX,8	
004DB364	74 89	JE SHORT 004DB2EF	UnPackMe.004DB2EF
004DB366	E8 A9000000	CALL 004DB414	UnPackMe.004DB414
004DB36A	8A45 FF	MOU BYTE PTR SS:[EBP-11],AL	

我们会发现 4DB364 处也是一个向上的跳转,说明这块代码是一个双层嵌套循环。

好,那么我们直接对 4DB364 这个地址的下一条语句处即 4DB336 地址处设置一个断点。

004DB2E9	BA 00000000	MOV EDX,80000000	
004DB2EE	43	INC EBX	
004DB2EF	31C0	XOR EAX,EAX	
004DB2F1	E8 11010000	CALL 004DB407	UnPackt
004DB2F6	73 0E	JNB SHORT 004DB306	UnPackt
004DB2F8	8B4D F0	MOV ECX,DWORD PTR SS:[EBP-10]	
004DB2FB	E8 1F010000	CALL 004DB41F	UnPackt
004DB300	0245 EF	ADD AL,BYTE PTR SS:[EBP-11]	
004DB303	AA	STOS BYTE PTR ES:[EDI]	
004DB304	EB E9	JMP SHORT 004DB2EF	UnPackt
004DB306	E8 FC000000	CALL 004DB407	UnPackt
004DB30B	0F82 97000000	JB 004DB3A8	UnPackt
004DB311	E8 F1000000	CALL 004DB407	UnPackt
004DB316	73 5B	JNB SHORT 004DB373	UnPackt
004DB318	B9 04000000	MOV ECX,4	
004DB31D	E8 FD000000	CALL 004DB41F	UnPackt
004DB322	48	DEC EAX	
004DB323	74 DE	JE SHORT 004DB303	UnPackt
004DB325	0F89 C7000000	JNS 004DB3F2	UnPackt
004DB32B	E8 D7000000	CALL 004DB407	UnPackt
004DB330	73 1B	JNB SHORT 004DB34D	UnPackt
004DB332	55	PUSH EBP	
004DB333	BD 00010000	MOV EBP,100	
004DB338	E8 D7000000	CALL 004DB414	UnPackt
004DB33D	8B07	MOV BYTE PTR DS:[EDI],AL	
004DB33F	47	INC EDI	
004DB340	4D	DEC EBP	
004DB341	75 F5	JNZ SHORT 004DB338	UnPackt
004DB343	E8 BF000000	CALL 004DB407	UnPackt
004DB348	72 E9	JB SHORT 004DB333	UnPackt
004DB34A	5D	POP EBP	
004DB34B	EB A2	JMP SHORT 004DB2EF	UnPackt
004DB34D	B9 01000000	MOV ECX,1	
004DB352	E8 C8000000	CALL 004DB41F	UnPackt
004DB357	83C0 07	ADD EAX,7	
004DB35A	8945 F0	MOV DWORD PTR SS:[EBP-10],EAX	
004DB35D	C645 EF 00	MOV BYTE PTR SS:[EBP-11],0	
004DB361	83F8 08	CMP EAX,8	
004DB364	74 89	JE SHORT 004DB2EF	UnPackt
004DB366	E8 A9000000	CALL 004DB414	UnPackt
004DB36D	8945 EF	MOV DWORD PTR SS:[EBP-10],EAX	

运行起来。

004DB357	83C0 07	ADD EAX,7	
004DB35A	8945 F0	MOV DWORD PTR SS:[EBP-10],EAX	
004DB35D	C645 EF 00	MOV BYTE PTR SS:[EBP-11],0	
004DB361	83F8 08	CMP EAX,8	
004DB364	74 89	JE SHORT 004DB2EF	UnPackt
004DB366	E8 A9000000	CALL 004DB414	UnPackt
004DB36B	8B45 EF	MOV BYTE PTR SS:[EBP-11],AL	
004DB36E	E9 7CFFFFFF	JMP 004DB2EF	UnPackt
004DB373	B9 07000000	MOV ECX,7	
004DB378	E8 A2000000	CALL 004DB41F	UnPackt
004DB37D	50	PUSH EAX	

我们可以看到断在了 4DB366 地址处,也就是我们跳过了这个双层嵌套循环。

接下来我们删除掉 4DB366 地址处的断点,然后依然是对代码段设置 break-on-access 断点。

003B0000	00008000				Priv	RW	RW
003C0000	00001000				Priv	RW	RW
003D0000	00001000				Priv	RW	RW
00400000	00001000	UnPackMe		PE header	Imag	R	RWE
00401000	0000A000	UnPackMe	.text	code	Imag	R	RWE
0044B000	0000C000	UnPackMe	.rdata	code	Imag	R	RWE
00457000	00009000	UnPackMe	.data	data	Imag	R	RWE
00460000	00003000	UnPackMe	dyuntioj		Imag	R	RWE
00463000	00008000	UnPackMe	.rsrc	resources	Imag	R	RWE
0046B000	00001000	UnPackMe	xd4ambdu		Imag	R	RWE
0046C000	00027000	UnPackMe	uvxwxip		Imag	R	RWE
00493000	0004B000	UnPackMe	nlmaiinz	SFX,imports	Imag	R	RWE
004E0000	00003000				Map	R E	R E
005A0000	00002000				Map	R E	R E
005B0000	00103000				Map	R	R
006C0000	00067000				Map	R E	R E

运行起来。

004271AE	70	NOP	
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 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 NEAR DWORD PTR DS:[460ADC]	UnP
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	
EBP=0012FFF0			
Address	Hex dump	ASCII	
Command :			
Break-on-access when executing [004271B0]			

这样我们就断在了 OEP 处,注意到 OD 状态栏中的提示信息没有?该中断是由执行导致的。

以上就是我定位 OEP 的方法。

不知道有木有童鞋的 OllyBone 插件能够断下来,如果你们中有人能断下来,请分享一下经验,谢谢,反正我是一次都没有断下来。

~~~~(>\_<)~~~~

)

这里我们可以看到断在了我们熟悉的 OEP-4271B0 处了,这个等级的保护并不存在 stolen bytes,我们直接就可以定位到 OEP。

|          |                  |                                 |                   |
|----------|------------------|---------------------------------|-------------------|
| 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 NEAR DWORD PTR DS:[460ADC] | UnPackMe.00492493 |
| 004271DC | 33D2             | XOR EDX,EDX                     |                   |
| 004271DE | 8AD4             | MOV DL,AH                       |                   |
| 004271E0 | 8915 34E64500    | MOV DWORD PTR DS:[45E634],EDX   |                   |

我们对比着 UPX 的 UnPackMe 来看,它到达 OEP 后以后,下面的 GetVersion 的调用处并没有被重定向,而我们这里被重定向了,下面我们来修复 IAT。

首先我们来定位 IAT 的起始地址和结束位置。

| Address  | Hex dump                                        | ASCII             |
|----------|-------------------------------------------------|-------------------|
| 004607F8 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....             |
| 00460808 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....             |
| 00460818 | A8 FC 47 00 41 6F 48 00 98 00 48 00 F9 A4 47 00 | zG.AoH.y.H.-xG.   |
| 00460828 | 81 D9 48 00 00 00 00 00 00 15 C5 58 2E B0 C3 58 | u'H.....!stX.c!X  |
| 00460838 | 00 00 00 00 04 6A EF 77 66 95 EF 77 89 6A EF 77 | ...Ej'wfo'wej'w   |
| 00460848 | F3 AD EF 77 ED D9 EF 77 99 8B EF 77 C0 B5 EF 77 | %i'wY'w0i'w!A'w   |
| 00460858 | 2A 7D EF 77 B2 7C EF 77 77 53 F2 77 1E C9 F1 77 | *)'wW!'wWS=w!F'w  |
| 00460868 | 0C BC EF 77 52 D4 EF 77 FA 8D EF 77 F1 D0 EF 77 | .d'wRE'w.i'wz!w   |
| 00460878 | 51 B2 EF 77 26 D5 EF 77 2A E3 EF 77 5F 39 F2 77 | Cw'w&'w*0'w_9=w   |
| 00460888 | 71 B4 EF 77 2E AD EF 77 E1 61 EF 77 B8 85 EF 77 | qI'w.i'wPa'w0a'w  |
| 00460898 | CC D2 EF 77 43 78 EF 77 FB EA F0 77 12 83 EF 77 | lFe'wCd'w!0-w#3'w |
| 00460908 | 01 72 F0 77 A9 34 F0 77 05 93 EF 77 68 EF EF 77 | 0r-w04-w!0'wh'w   |
| 00460918 | AA D2 EF 77 B2 6F EF 77 3F 38 F2 77 D6 E8 EF 77 | -e'wW'o'w?8=w!P'w |
| 00460928 | 68 E0 EF 77 00 60 EF 77 90 58 EF 77 6D AC EF 77 | h0'w.'wE!wM%w     |
| 00460938 | 94 6C F0 77 22 8D EF 77 3D C8 F1 77 3D 6D F0 77 | 3l-w'!w=wz=wM-w   |
| 00460948 | 6F C0 EF 77 85 78 EF 77 26 D9 EF 77 FB 5E EF 77 | 5l'wac'w&'w!^w    |
| 00460958 | 36 8A EF 77 FC 8A EF 77 0F 62 EF 77 49 5E EF 77 | 6e'w?e'w*b'wI'w   |
| 00460968 | 97 5D EF 77 1A 9A EF 77 68 FA EF 77 78 C9 F0 77 | uJ'w+u'wk'w!F'w   |
| 00460978 | DA 98 F2 77 1A 40 F2 77 55 EA EF 77 C5 61 EF 77 | rY=w+@=wU0'w+a'w  |
| 00460988 | 70 E6 EF 77 F0 81 EF 77 2D 6C EF 77 98 6E EF 77 | pu'w-u'w-l'wYn'w  |
| 00460998 | 4F 83 EF 77 09 ED EF 77 EB AA EF 77 26 69 F0 77 | 0a'w.y'wu-w&i-w   |
| 004609A8 | B1 95 EF 77 6F B0 EF 77 8A 5A EF 77 E9 49 F2 77 | 88'woc'weZ'wU!w   |
| 004609B8 | 26 F1 F0 77 C9 D0 F0 77 51 E0 F0 77 33 8C EF 77 | &z-w!f!-wQ0-w3I'w |
| 004609C8 | 6C EC EF 77 29 94 EF 77 00 00 00 00 B6 89 48 00 | (y'w)0'w....AeH.  |
| 004609D8 | A9 58 48 00 2F D8 48 00 8C C2 46 00 D0 89 47 00 | 0XH./iH.iF.sëG.   |
| 004609E8 | 00 04 48 00 16 01 49 00 B7 14 49 00 29 F6 46 00 | .eH..0I.A0I.)+.   |
| 004609F8 | 34 E6 47 00 F3 C5 47 00 08 23 48 00 86 30 48 00 | 4uG.%+G.#H.30H.   |
| 00460A08 | 45 0D 48 00 69 7A 47 00 78 D1 48 00 69 24 49 00 | E.H.i2G.(0H.i3I.  |
| 00460A18 | E9 C1 48 00 C8 0E 48 00 02 41 47 00 C7 4D 48 00 | u+H.50H.0AG.3MH.  |
| 00460A28 | C2 9E 48 00 52 6D 47 00 2E 43 47 00 14 58 48 00 | t+H.R'G.CG.0XH.   |
| 00460A38 | 48 29 48 00 20 4A 47 00 88 FC 46 00 91 36 47 00 | H*H..jG.e*F.#6G.  |

这里我们可以看到 IAT 的起始地址为 460818,跟 UPX 的 UnPackMe 一样。我们继续往下定位 IAT 的结束位置。

| Address  | Hex dump                                        | ASCII              |
|----------|-------------------------------------------------|--------------------|
| 00460DB8 | 40 64 48 00 37 62 48 00 D6 B8 48 00 2C EE 48 00 | @dH.7bH.i0H..-H.   |
| 00460DC8 | E1 0F 47 00 42 98 48 00 64 24 48 00 7D 44 47 00 | %*G.B0H.d3H.)0G.   |
| 00460DD8 | D2 C3 46 00 42 D4 46 00 0A 6E 48 00 53 C8 46 00 | e!F.BeF..nH.SeF.   |
| 00460DE8 | CE 29 48 00 68 E0 47 00 B4 A3 47 00 B2 69 48 00 | te)H.k0G.-u0G.0iH. |
| 00460DF8 | 13 7A 48 00 C0 08 47 00 47 F5 46 00 D3 F1 48 00 | !!zH.'0G.G3F.ezH.  |
| 00460E08 | 07 FC 47 00 47 43 48 00 9A 54 47 00 D0 19 47 00 | .*G.GCH.0TG.i+G.   |
| 00460E18 | E7 F9 46 00 15 45 47 00 E8 CF 46 00 CE C6 46 00 | f-F.SEG.p0F.teSF.  |
| 00460E28 | 92 AD 47 00 77 91 48 00 19 68 47 00 E6 77 47 00 | te+G.waH.+kG.pwG.  |
| 00460E38 | 3A 64 48 00 71 76 47 00 B9 F9 47 00 E3 F3 48 00 | .dH.qvG.i!-G.0%H.  |
| 00460E48 | 8D C5 46 00 06 17 47 00 A8 38 47 00 7D C0 47 00 | i+F.#+G.28G.)!G.   |
| 00460E58 | 64 2A 48 00 F6 CD 48 00 2A 92 47 00 77 08 47 00 | d*H.+H.*eG.w0G.    |
| 00460E68 | 40 CE 47 00 05 29 48 00 41 D2 48 00 31 58 47 00 | 0eF.G.#)H.AeH.1XG. |
| 00460E78 | 0D 92 48 00 54 DA 46 00 76 0F 47 00 49 F7 47 00 | .eH.TrF.v*G.I+G.   |
| 00460E88 | 00 00 00 00 F7 A8 B1 76 00 00 00 00 C8 74 F8 72 | ....c0v....!t0r    |
| 00460E98 | 73 66 F9 72 87 72 F8 72 43 80 F8 72 67 37 F9 72 | sf-r0r0rC0rg7-r    |
| 00460EA8 | F8 A1 F9 72 67 83 F8 72 90 53 F8 72 00 00 00 00 | 'A-r0a0reS0r....   |
| 00460EB8 | CE 00 37 76 7C 86 37 76 B0 86 37 76 33 25 36 76 | te.7v!37v37v3%6v   |
| 00460EC8 | 1E 31 36 76 D8 7C 37 76 89 C2 37 76 CD 46 38 76 | !16vi!7v0t7vF8v    |
| 00460ED8 | CE EE 36 76 00 00 00 48 D0 4C 77 9C C8 4D 77    | te'6v....H3Lw0rMw  |
| 00460EE8 | CC 42 4F 77 2C D0 4C 77 DA F6 4C 77 73 33 50 77 | !FB0w.\$Lwr+Lws3Pw |
| 00460EF8 | 10 64 4D 77 03 0E 52 77 33 0F 52 77 40 A6 54 77 | !dMw00Rw3*Rw03Tw   |
| 00460F08 | F1 A7 54 77 92 9C 4F 77 6F 57 52 77 99 33 4E 77 | z0Tw!0w0wRw03Nw    |
| 00460F18 | B2 5D 4E 77 90 C0 5A 77 00 00 00 00 F3 F0 CC 74 | WJNwe'Zw....%-!ft  |
| 00460F28 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    | .....              |
| 00460F38 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    | .....              |

这里我们可以看到 IAT 的结束地址为 460F28。我们来计算一下 IAT 的长度。

004610F4

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Command

? 460f28-460818

▼

HEX: 710 - C

计算得出 IAT 的长度为 710。

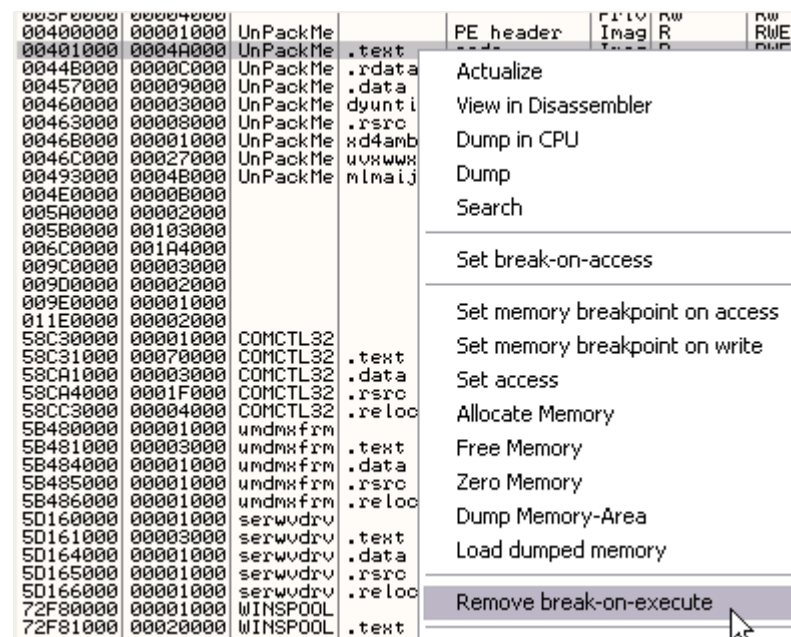
OEP(RVA):271B0

IAT 起始地址(RVA):60818

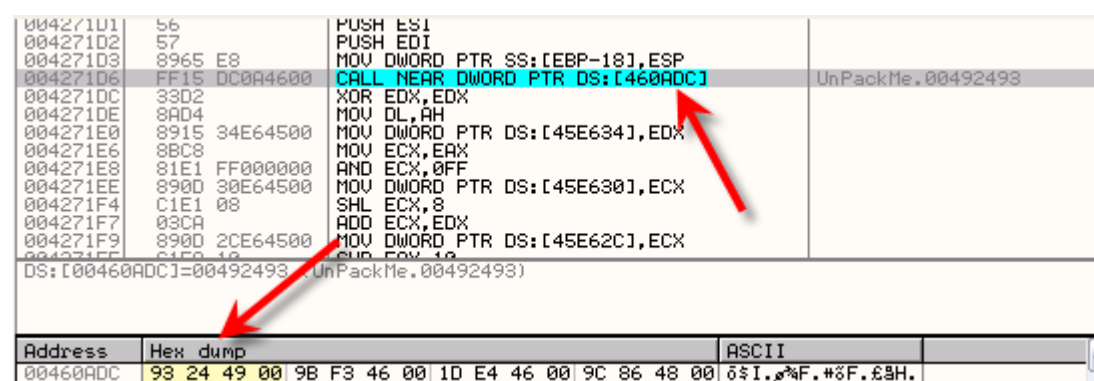
IAT 的大小:710

这里 IMP REC 重建 IAT 所需要的数据我们都有了。

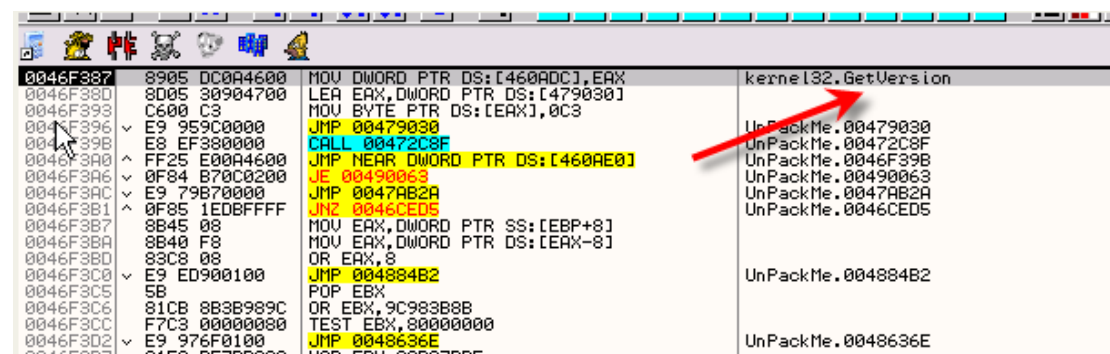
下面我们的任务就是来修复 IAT, ExeCryptor 并不存在我们前面章节介绍过的关键跳, 它是怎么做呢? 它会在特定的时候将正确的 API 函数地址填充到对应的 IAT 项中, 所以这里我们给 GetVersion 所在的 IAT 项设置内存写入断点。



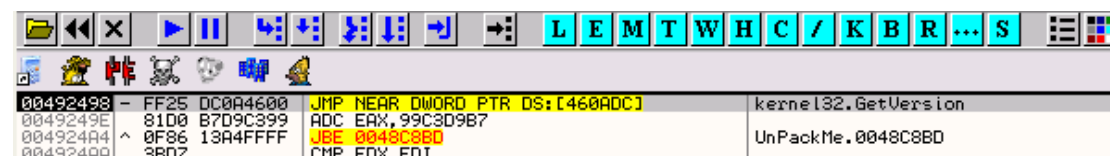
这里我们首先要选择 Remove break-on-execute 将 break-on-execute 断点删除掉, 以免出错。



这里我们给将要调用的第一个 API 函数所在的 IAT 项设置了内存写入断点, 运行起来, 看看会发生什么。



断在了这里, 我们可以看到这一条指令是将正确的 API 函数地址保存到对应的 IAT 项中。



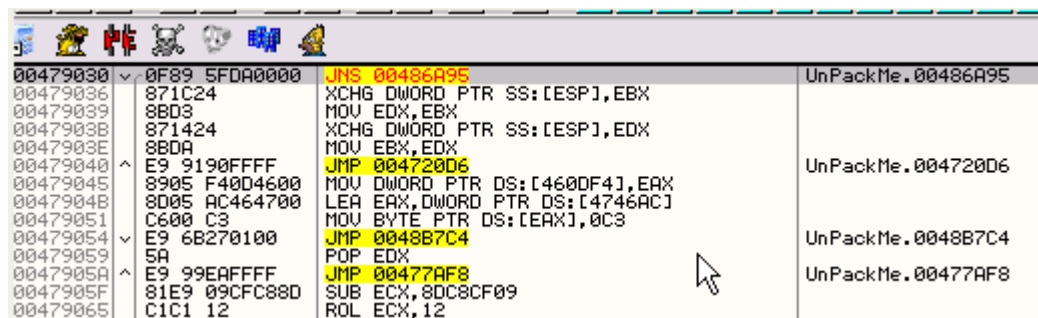


下面几行,我们会发现其会将 479030 处的代码修改为 C3,即 RET 指令,然后利用 RET 指令返回到某地址处再去调用实际的要调用的 API 函数,下面我们来详细跟踪一下这个流程。



我们可以看到在正确的 API 地址被保存到对应的 IAT 项中后,下面会向 479030 地址处写入一个 C3,我们看看 479030 地址处之前是什么内容。

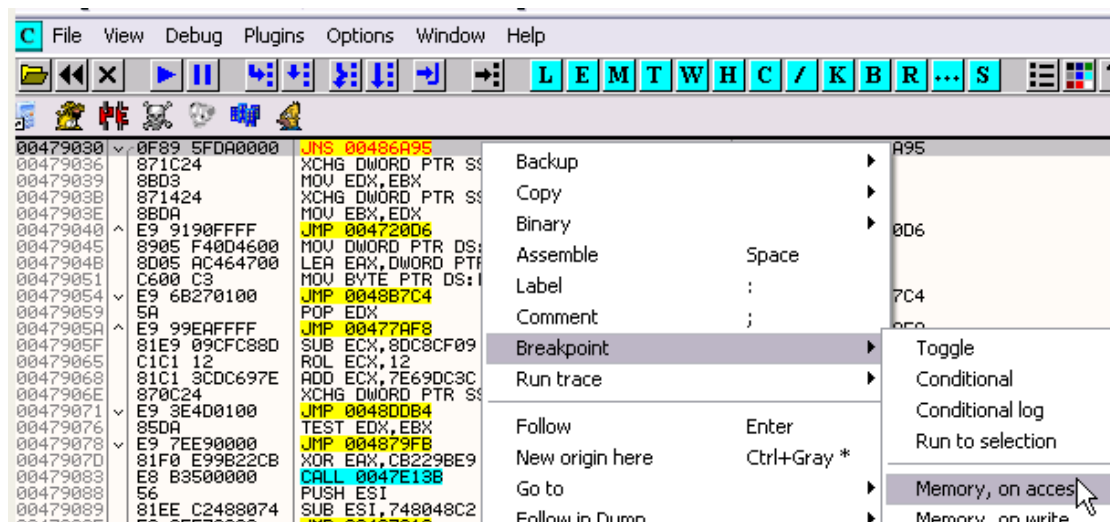
479030 之前是这样的:



写入 C3 后变成了:



这里我们可以看到被修改为了 RET 指令,相当于该壳在修复 IAT 项以后再进行自修改(修改自身代码)。但是该壳的代码并不位于第一个区段,所以这里我们重启 OD,达到 OEP 以后对 479030 地址处的指令设置内存访问断点。



首先我们到达第一个 API 函数调用处。

|          |               |                                 |                   |
|----------|---------------|---------------------------------|-------------------|
| 00427101 | 56            | PUSH ESI                        |                   |
| 00427102 | 57            | PUSH EDI                        |                   |
| 00427103 | 8965 E8       | MOV DWORD PTR SS:[EBP-18],ESP   |                   |
| 00427106 | FF15 DC0A4600 | CALL NEAR DWORD PTR DS:[460ADC] | UnPackMe.00492493 |
| 0042710C | 33D2          | XOR EDX,EDX                     |                   |
| 0042710E | 8AD4          | MOV DL,AH                       |                   |
| 0042710A | 8915 34F64500 | MOV DWORD PTR DS:[45F634],EDX   |                   |

下面我们给返回地址 4271DC 处设置一个断点。

|          |               |                                 |                   |
|----------|---------------|---------------------------------|-------------------|
| 00427102 | 57            | PUSH EDI                        |                   |
| 00427103 | 8965 E8       | MOV DWORD PTR SS:[EBP-18],ESP   |                   |
| 00427106 | FF15 DC0A4600 | CALL NEAR DWORD PTR DS:[460ADC] | UnPackMe.00492493 |
| 0042710C | 33D2          | XOR EDX,EDX                     |                   |
| 0042710E | 8AD4          | MOV DL,AH                       |                   |
| 0042710A | 8915 34E64500 | MOV DWORD PTR DS:[45E634],EDX   |                   |

现在我们运行起来。

|          |               |                               |                   |
|----------|---------------|-------------------------------|-------------------|
| 00479030 | 0F89 5FDA0000 | JNS 00486A95                  | UnPackMe.00486A95 |
| 00479036 | 871C24        | XCHG DWORD PTR SS:[ESP],EBX   |                   |
| 00479039 | 8BD3          | MOV EDX,EBX                   |                   |
| 0047903B | 871424        | XCHG DWORD PTR SS:[ESP],EDX   |                   |
| 0047903E | 8BD3          | MOV EBX,EDX                   |                   |
| 00479040 | E9 9190FFFF   | JMP 004720D6                  | UnPackMe.004720D6 |
| 00479045 | 89A5 F4004600 | MOV DWORD PTR DS:[4600F4],EAX |                   |

我们可以看到断在了条件跳转处,当 API 函数地址被填充到对应 IAT 项中以后,这里就会被自修改为 RET 指令,壳的自修改是我们重点关注的,也就是说我们在到达 OEP 处以后,可以对壳修复 IAT 项代码所在的区段设置内存写入断点,当断下来时,就到了自修改的地方,

现在我们重启 OD,到达 OEP 处。

|          |                |                                 |                   |
|----------|----------------|---------------------------------|-------------------|
| 00427105 | 50             | PUSH EAX                        |                   |
| 00427106 | 64:8925 000000 | MOV DWORD PTR FS:[0],ESP        |                   |
| 0042710D | 83C4 A8        | ADD ESP,-58                     |                   |
| 00427100 | 53             | PUSH EBX                        |                   |
| 00427101 | 56             | PUSH ESI                        |                   |
| 00427102 | 57             | PUSH EDI                        |                   |
| 00427103 | 8965 E8        | MOV DWORD PTR SS:[EBP-18],ESP   |                   |
| 00427106 | FF15 DC0A4600  | CALL NEAR DWORD PTR DS:[460ADC] | UnPackMe.00492493 |
| 0042710C | 33D2           | XOR EDX,EDX                     |                   |
| 0042710E | 8AD4           | MOV DL,AH                       |                   |
| 00427100 | 8915 34E64500  | MOV DWORD PTR DS:[45E634],EDX   |                   |
| 00427106 | 8BC8           | MOV ECX,EAX                     |                   |
| 00427108 | 81E1 FF000000  | AND ECX,0FF                     |                   |

这里我们再次到达第一个 API 函数调用处,这里是 CALL 重定向后的地址 492493,当 479030 处被修改为了 RET 指令,重定向的 IAT 项已经被恢复为正常的 API 函数地址了,所以这里我们给壳所在的区段设置内存写入断点,接着运行起来,看看会发生什么。

|          |          |          |                                 |           |        |    |
|----------|----------|----------|---------------------------------|-----------|--------|----|
| 00460000 | 00003000 | UnPackMe | dyuntioj                        |           | Imag R | Rt |
| 00463000 | 00008000 | UnPackMe | .rsrce                          | resources | Imag R | Rt |
| 0046B000 | 00001000 | UnPackMe | xd4ambdu                        |           | Imag R | Rt |
| 0046C000 | 00027000 | UnPackMe |                                 |           | Imag R | Rt |
| 00493000 | 0004B000 | UnPackMe |                                 |           |        |    |
| 004E0000 | 00009000 |          | Actualize                       |           |        |    |
| 005A0000 | 00002000 |          | Dump in CPU                     |           |        |    |
| 005B0000 | 00103000 |          | Dump                            |           |        |    |
| 006C0000 | 00117000 |          | Search                          |           | Ctrl+B |    |
| 009D0000 | 00002000 |          |                                 |           |        |    |
| 009E0000 | 00001000 |          |                                 |           |        |    |
| 011E0000 | 00002000 |          |                                 |           |        |    |
| 58C30000 | 00001000 | COMCTL3  | Set break-on-access             |           | F2     |    |
| 58C31000 | 00070000 | COMCTL3  |                                 |           |        |    |
| 58CA1000 | 00003000 | COMCTL3  | Set memory breakpoint on access |           |        |    |
| 58CA4000 | 0001F000 | COMCTL3  |                                 |           |        |    |
| 58CC3000 | 00004000 | COMCTL3  | Set memory breakpoint on write  |           |        |    |
| 5B480000 | 00001000 | undmxf   | Set access                      |           |        |    |
| 5B481000 | 00003000 | undmxf   |                                 |           |        |    |
| 5B484000 | 00001000 | undmxf   |                                 |           |        |    |
| 5B485000 | 00001000 | undmxf   | Allocate Memory                 |           |        |    |

|          |               |                                 |                   |
|----------|---------------|---------------------------------|-------------------|
| 00427106 | FF15 DC0A4600 | CALL NEAR DWORD PTR DS:[460ADC] | UnPackMe.00492493 |
| 0042710C | 33D2          | XOR EDX,EDX                     |                   |
| 0042710E | 8AD4          | MOV DL,AH                       |                   |

接着我们还是对返回地址 4271DC 处设置一个断点,运行起来,看看会发生什么。



|          |               |                              |
|----------|---------------|------------------------------|
| 004806D1 | 8802          | MOV BYTE PTR DS:[EDX],AL     |
| 004806D3 | 8B45 F8       | MOV EAX,DWORD PTR SS:[EBP-8] |
| 004806D6 | 8A00          | MOV AL,BYTE PTR DS:[EAX]     |
| 004806D8 | 0045 F6       | ADD BYTE PTR SS:[EBP-A],AL   |
| 004806DB | 33C0          | XOR EAX,EAX                  |
| 004806DD | 8A45 F6       | MOV AL,BYTE PTR SS:[EBP-A]   |
| 004806E0 | ✓ E9 3E150000 | JMP 00481C23                 |
| 004806E5 | 81C1 F68EDF64 | ADD ECX,64DF8EF6             |
| 004806EB | 870C24        | XCHG DWORD PTR SS:[ESP],ECX  |
| 004806EE | ^ E9 9478FFFF | JMP 00477F87                 |

AL=4B ('K')  
DS:[0047A9E8]=B4

继续:

|          |               |                              |
|----------|---------------|------------------------------|
| 004806D1 | 8802          | MOV BYTE PTR DS:[EDX],AL     |
| 004806D3 | 8B45 F8       | MOV EAX,DWORD PTR SS:[EBP-8] |
| 004806D6 | 8A00          | MOV AL,BYTE PTR DS:[EAX]     |
| 004806D8 | 0045 F6       | ADD BYTE PTR SS:[EBP-A],AL   |
| 004806DB | 33C0          | XOR EAX,EAX                  |
| 004806DD | 8A45 F6       | MOV AL,BYTE PTR SS:[EBP-A]   |
| 004806E0 | ✓ E9 3E150000 | JMP 00481C23                 |
| 004806E5 | 81C1 F68EDF64 | ADD ECX,64DF8EF6             |
| 004806EB | 870C24        | XCHG DWORD PTR SS:[ESP],ECX  |
| 004806EE | ^ E9 9478FFFF | JMP 00477F87                 |

AL=45 ('E')  
DS:[0047A9E9]=45 ('E')

|          |               |                              |
|----------|---------------|------------------------------|
| 004806D1 | 8802          | MOV BYTE PTR DS:[EDX],AL     |
| 004806D3 | 8B45 F8       | MOV EAX,DWORD PTR SS:[EBP-8] |
| 004806D6 | 8A00          | MOV AL,BYTE PTR DS:[EAX]     |
| 004806D8 | 0045 F6       | ADD BYTE PTR SS:[EBP-A],AL   |
| 004806DB | 33C0          | XOR EAX,EAX                  |
| 004806DD | 8A45 F6       | MOV AL,BYTE PTR SS:[EBP-A]   |
| 004806E0 | ✓ E9 3E150000 | JMP 00481C23                 |
| 004806E5 | 81C1 F68EDF64 | ADD ECX,64DF8EF6             |
| 004806EB | 870C24        | XCHG DWORD PTR SS:[ESP],ECX  |
| 004806EE | ^ E9 9478FFFF | JMP 00477F87                 |

AL=52 ('R')  
DS:[0047A9EA]=94

| Address  | Hex dump                                        |
|----------|-------------------------------------------------|
| 0047A9E8 | 4B 45 52 4E 45 4C 33 32 2E 64 6C 6C 00 E9 A1 AF |
| 0047A9F8 | FF FF 0F 84 CF E7 FF FF 68 72 9D 07 4E 5A E9 4C |
| 0047AA08 | C2 00 00 0F 85 AF 96 00 00 C1 EB 03 F7 C3 B4 6F |

下一个:

|          |               |                              |
|----------|---------------|------------------------------|
| 004806D1 | 8802          | MOV BYTE PTR DS:[EDX],AL     |
| 004806D3 | 8B45 F8       | MOV EAX,DWORD PTR SS:[EBP-8] |
| 004806D6 | 8A00          | MOV AL,BYTE PTR DS:[EAX]     |
| 004806D8 | 0045 F6       | ADD BYTE PTR SS:[EBP-A],AL   |
| 004806DB | 33C0          | XOR EAX,EAX                  |
| 004806DD | 8A45 F6       | MOV AL,BYTE PTR SS:[EBP-A]   |
| 004806E0 | ✓ E9 3E150000 | JMP 00481C23                 |
| 004806E5 | 81C1 F68EDF64 | ADD ECX,64DF8EF6             |
| 004806EB | 870C24        | XCHG DWORD PTR SS:[ESP],ECX  |

AL=4E ('N')  
DS:[0047A9EB]=9C

这是实际上是在填充一个字符串,我们就不一个字母一个字母的看了,我们直接在数据窗口中查看填充完毕后的整个字符串是什么。

| Address  | Hex dump                                        | ASCII            |
|----------|-------------------------------------------------|------------------|
| 0047A9E8 | 4B 45 52 4E 45 4C 33 32 2E 64 6C 6C 00 E9 A1 AF | KERNEL32.dll.üi» |
| 0047A9F8 | FF FF 0F 84 CF E7 FF FF 68 72 9D 07 4E 5A E9 4C | *äð hr0•NZÜL     |
| 0047AA08 | C2 00 00 0F 85 AF 96 00 00 C1 EB 03 F7 C3 B4 6F | т..*ä»ü...±ü♥•Hо |

我们可以看到实际上是一个 DLL 的名称字符串,可能下面会被用来获取 API 函数的地址,我们继续。

|          |                 |                              |             |
|----------|-----------------|------------------------------|-------------|
| 0047F87E | C600 00         | MOV BYTE PTR DS:[EAX],0      |             |
| 0047F881 | 8BE5            | MOV ESP,EBP                  |             |
| 0047F883 | 5D              | POP EBP                      |             |
| 0047F884 | C3              | RETN                         |             |
| 0047F885 | 8B45 FC         | MOV EAX,DWORD PTR SS:[EBP-4] |             |
| 0047F888 | E8 DC79FFFF     | CALL 00477269                | UnPackMe.00 |
| 0047F88D | E9 0BC10000     | JMP 0048B99D                 | UnPackMe.00 |
| 0047F892 | 13D6            | ADC EDX,ESI                  |             |
| 0047F894 | 87DF            | XCHG EDI,EBX                 |             |
| 0047F896 | 8B45 FC         | MOV EAX,DWORD PTR SS:[EBP-4] |             |
| 0047F899 | 8038 F1         | CMP BYTE PTR DS:[EAX],0F1    |             |
| 0047F89C | ^ 0F85 4A3CFFFF | JNZ 004734EC                 | UnPackMe.00 |
| 0047F8A2 | ^ E9 55E00000   | JMP 0048D8FC                 | UnPackMe.00 |
| 0047F8A3 | ^ 0F0F 2F300000 | JMP 00480156                 | UnPackMe.00 |

DS:[0047A9F4]=00

| Address  | Hex dump                                        | ASCII            |
|----------|-------------------------------------------------|------------------|
| 0047A9E8 | 4B 45 52 4E 45 4C 33 32 2E 64 6C 6C 00 E9 A1 AF | KERNEL32.dll.üi> |
| 0047A9F8 | FF FF 0F 84 CF E7 FF 68 72 9D 07 4E 5A E9 4C    | *äð hr0.NZÜL     |
| 0047AA08 | C2 00 00 0F 85 AF 96 00 C1 EB 03 F7 C3 B4 6F    | T...ä>ü..+Ü•H0   |

这里是添加字符串结束符'\0'。

|          |                 |                              |             |
|----------|-----------------|------------------------------|-------------|
| 004806D1 | 8802            | MOV BYTE PTR DS:[EDX],AL     |             |
| 004806D3 | 8B45 F8         | MOV EAX,DWORD PTR SS:[EBP-8] |             |
| 004806D6 | 8A00            | MOV AL,BYTE PTR DS:[EAX]     |             |
| 004806D8 | 0045 F6         | ADD BYTE PTR SS:[EBP-A],AL   |             |
| 004806DB | 33C0            | XOR EAX,EAX                  |             |
| 004806DD | 8A45 F6         | MOV AL,BYTE PTR SS:[EBP-A]   |             |
| 004806E0 | ^ E9 3E150000   | JMP 00481C23                 | UnPackMe.00 |
| 004806E5 | 81C1 F68EDF64   | ADD ECX,64DF8EF6             |             |
| 004806E8 | 870C24          | XCHG DWORD PTR SS:[ESP],ECX  |             |
| 004806EB | ^ E9 9478FFFF   | JMP 00477F87                 | UnPackMe.00 |
| 004806F3 | ^ 0F8C ECF7FFFF | JL 004806E5                  | UnPackMe.00 |
| 004806F9 | ^ E9 6258FFFF   | JMP 00475F60                 | UnPackMe.00 |
| 004806FE | 81E9 D1DF0F18   | SUB ECX,18DFDFD1             |             |
| 00480704 | 00              | CDB                          |             |

AL=B4  
DS:[0047A9E8]=4B ('K')

| Address  | Hex dump                                        | ASCII            |
|----------|-------------------------------------------------|------------------|
| 0047A9E8 | 4B 45 52 4E 45 4C 33 32 2E 64 6C 6C 00 E9 A1 AF | KERNEL32.dll.üi> |
| 0047A9F8 | FF FF 0F 84 CF E7 FF 68 72 9D 07 4E 5A E9 4C    | *äð hr0.NZÜL     |

这里又是重复上面的步骤,但是这次是填充字符'K'的小端存储方式,4B 反过来就是 B4,这么做的目的可能是为了隐藏字符串。

|          |                 |                               |                   |
|----------|-----------------|-------------------------------|-------------------|
| 0047493B | 8913            | MOV DWORD PTR DS:[EBX],EDX    | kernel32.7C800000 |
| 0047493D | 5B              | POP EBX                       |                   |
| 0047493E | 92              | XCHG EAX,EDX                  |                   |
| 0047493F | E8 B795FFFF     | CALL 0046DEFB                 | UnPackMe.0046DEFB |
| 00474944 | C3              | RETN                          |                   |
| 00474945 | ^ E9 79420100   | JMP 00488BC3                  | UnPackMe.00488BC3 |
| 0047494A | ^ E9 CCDE0000   | JMP 0048281B                  | UnPackMe.0048281B |
| 0047494F | ^ E9 8B600100   | JMP 0048AFDF                  | UnPackMe.0048AFDF |
| 00474954 | 0000            | ADD BYTE PTR DS:[EAX],AL      |                   |
| 00474956 | 0000            | ADD BYTE PTR DS:[EAX],AL      |                   |
| 00474958 | ^ 0F83 E11F0100 | JNB 0048693F                  | UnPackMe.0048693F |
| 0047495E | 8B15 48EE4700   | MOV EDX,DWORD PTR DS:[47EE48] |                   |
| 00474964 | 09D2            | OR EDX,EDX                    |                   |
| 00474966 | ^ 0F85 D4A00000 | JNZ 0047EE40                  | UnPackMe.0047EE40 |
| 0047496C | ^ E9 A1E2FFFF   | JMP 00472C12                  | UnPackMe.00472C12 |
| 00474971 | 03D5            | ADD EDX,EBP                   |                   |
| 00474973 | 81C2 ADBE4071   | ADD EDX,7140BEAD              |                   |
| 00474979 | 8B12            | MOV EDX,DWORD PTR DS:[EDX]    |                   |

EDX=7C800000 (kernel32.7C800000)  
DS:[00474954]=00000000

| Address  | Hex dump                                        | ASCII            |
|----------|-------------------------------------------------|------------------|
| 00474954 | 00 00 00 00 0F 83 E1 0F 01 00 8B 15 48 EE 47 00 | ....*äp70.TSH^G. |
| 00474964 | A9 02 0F 85 04 A4 00 00 F9 A1 F2 FF FF 03 05 A1 | .F*äpK...üi0 •üi |

接着到了这里,我们可以看到是保存 Kernel32.dll 的基地址,我这里是 7C800000。

| 0046F393         | C600 C3                                         | MOV BYTE PTR DS:[EAX],0C3      |
|------------------|-------------------------------------------------|--------------------------------|
| 0046F396         | E9 959C0000                                     | JMP 00479030                   |
| 0046F39B         | E8 EF380000                                     | CALL 00472C8F                  |
| 0046F3A0         | FF25 E00A4600                                   | JMP NEAR DWORD PTR DS:[460AE0] |
| 0046F3A6         | 0F84 B70C0200                                   | JE 00490063                    |
| 0046F3AC         | E9 79B70000                                     | JMP 0047AB2A                   |
| 0046F3B1         | 0F85 1EDBFFFF                                   | JNZ 0046CED5                   |
| 0046F3B7         | 8B45 08                                         | MOV EAX,DWORD PTR SS:[EBP+8]   |
| 0046F3BA         | 8B40 F8                                         | MOV EAX,DWORD PTR DS:[EAX-8]   |
| 0046F3BD         | 83C8 08                                         | OR EAX,8                       |
| 0046F3C0         | E9 ED900100                                     | JMP 004884B2                   |
| 0046F3C5         | 5B                                              | POP EBX                        |
| 0046F3C6         | 81CB 8B3B989C                                   | OR EBX,9C983B8B                |
| 0046F3CC         | F7C3 00000000                                   | TEST EBX,80000000              |
| 0046F3D2         | E9 976F0100                                     | JMP 0048636E                   |
| 0046F3D7         | 81F2 D57BD920                                   | XOR EDX,20D97BD5               |
| 0046F3DD         | 81EA 0E60598F                                   | SUB EDX,8F59600E               |
| 0046F3E3         | 81C2 16C01B80                                   | AND EDX,801BC016               |
| DS:[00479030]=0F |                                                 |                                |
| Address          | Hex dump                                        | ASCII                          |
| 00474954         | 00 00 80 7C 0F 83 E1 1F 01 00 8B 15 48 EE 47 00 | ..Ç!*                          |
| 00474964         | 09 02 0F 85 D4 A4 00 00 E9 A1 E2 FF FF 03 D5 81 | ..E*âE                         |

接着再次到了这里,填充 C3,此时对应 IAT 项中的值已经被修复为正常的 API 函数地址了。

|          |               |                                    |                     |
|----------|---------------|------------------------------------|---------------------|
| 004271C0 | 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 NEAR DWORD PTR DS:[460ADC]    | kernel32.GetVersion |
| 004271DC | 33D2          | XOR EDX,EDX                        |                     |
| 004271DE | 8AD4          | MOV DL,AH                          |                     |
| 004271E0 | 8915 34F64500 | MOV EAX,DWORD PTR DS:[45F6341] EAX |                     |

我们继续运行就断在了返回地址处,我们按 F7 单步继续往下跟踪到下一个 API 函数调用处。

|          |               |                                 |                   |
|----------|---------------|---------------------------------|-------------------|
| 004293A0 | 6A 00         | PUSH 0                          |                   |
| 004293A2 | 68 00100000   | PUSH 1000                       |                   |
| 004293A7 | 6A 00         | PUSH 0                          |                   |
| 004293A9 | FF15 A0094600 | CALL NEAR DWORD PTR DS:[4609A0] | UnPackMe.00482308 |
| 004293AF | 85C0          | TEST EAX,EAX                    |                   |
| 004293B1 | A3 C4EB4500   | MOV DWORD PTR DS:[45EBC4],EAX   |                   |
| 004293B6 | 75 01         | JNZ SHORT 004293B9              | UnPackMe.004293B9 |
| 004293B8 | C3            | RETN                            |                   |

给返回地址处设置一个断点,运行起来。

|          |               |                                 |                   |
|----------|---------------|---------------------------------|-------------------|
| 004293A2 | 68 00100000   | PUSH 1000                       |                   |
| 004293A7 | 6A 00         | PUSH 0                          |                   |
| 004293A9 | FF15 A0094600 | CALL NEAR DWORD PTR DS:[4609A0] | UnPackMe.00482308 |
| 004293AF | 85C0          | TEST EAX,EAX                    |                   |
| 004293B1 | A3 C4EB4500   | MOV DWORD PTR DS:[45EBC4],EAX   |                   |
| 004293B6 | 75 01         | JNZ SHORT 004293B9              | UnPackMe.004293B9 |

|                        |               |                               |
|------------------------|---------------|-------------------------------|
| 0048D8B7               | C600 C3       | MOV BYTE PTR DS:[EAX],0C3     |
| 0048D8BA               | E9 B851FFFF   | JMP 00482A77                  |
| 0048D8BF               | 8B0424        | MOV EAX,DWORD PTR SS:[ESP]    |
| 0048D8C2               | 50            | PUSH EAX                      |
| 0048D8C3               | E9 45F2FFFF   | JMP 0048CB00                  |
| 0048D8C8               | 8D05 67324700 | LEA EAX,DWORD PTR DS:[473267] |
| 0048D8CE               | C600 C3       | MOV BYTE PTR DS:[EAX],0C3     |
| 0048D8D1               | E9 2826FFFF   | JMP 0047FEFE                  |
| 0048D8D6               | 8B0424        | MOV EAX,DWORD PTR SS:[ESP]    |
| 0048D8D9               | 52            | PUSH EDX                      |
| 0048D8DA               | E8 51E8FEFF   | CALL 0047C130                 |
| 0048D8DF               | B8 FC862613   | MOV EAX,132686FC              |
| 0048D8E4               | 53            | PUSH EBX                      |
| 0048D8E5               | E8 8761FEFF   | CALL 00473A71                 |
| 0048D8EA               | E9 8B41FFFF   | JMP 00481A7A                  |
| 0048D8EF               | E9 BEFEFDFF   | JMP 0046D7B2                  |
| 0048D8F4               | 870C24        | XCHG DWORD PTR SS:[ESP],ECX   |
| 0048D8F7               | E9 FED0FEFF   | JMP 0047A9FA                  |
| 0048D8FC               | 0F84 E55BFEFF | JE 004734E7                   |
| DS:[0047A0BC]=5A ('Z') |               |                               |

这里是将 47A0BC 处的首字节修改为 RET 指令。

|          |                 |                                |                    |
|----------|-----------------|--------------------------------|--------------------|
| 0047A09E | C1C9 12         | ROR ECX,12                     |                    |
| 0047A0A1 | ^ 0F87 A554FFFF | JA 0046F54C                    | UnPackMe.0046F54C  |
| 0047A0A7 | ^ E9 BF1F0000   | JMP 0047C06B                   | UnPackMe.0047C06B  |
| 0047A0AC | - FF25 A0094600 | JMP NEAR DWORD PTR DS:[4609A0] | kernel32.HeapCreat |
| 0047A0B2 | ^ E9 9CE90000   | JMP 00488A53                   | UnPackMe.00488A53  |
| 0047A0B7 | ^ E9 413D0100   | JMP 0048DDFD                   | UnPackMe.0048DDFD  |
| 0047A0BC | C3              | RETN                           |                    |

我们单步往下跟踪到 47A0BC 的 RET 指令处,可以看到对应 IAT 项中的值已经被修改为正确的 API 函数地址了,该 API 函数是 Kernel32.dll 导出的。

我们继续看接下来要调用的这个 API 函数。

|          |                 |                               |                   |
|----------|-----------------|-------------------------------|-------------------|
| 00429409 | 85ED            | TEST EBP,EBP                  |                   |
| 0042940B | ^ 0F84 2B010000 | JE 0042953C                   | UnPackMe.0042953C |
| 00429411 | 8B3D A8094600   | MOV EDI,DWORD PTR DS:[4609A8] | UnPackMe.00480D45 |
| 00429417 | 6A 04           | PUSH 4                        |                   |
| 00429419 | 68 00200000     | PUSH 2000                     |                   |
| 0042941E | 68 00004000     | PUSH 400000                   |                   |
| 00429423 | 6A 00           | PUSH 0                        |                   |
| 00429425 | FFD7            | CALL NEAR EDI                 |                   |
| 00429427 | 8BF0            | MOV ESI,EAX                   |                   |
| 00429429 | 85F6            | TEST ESI,ESI                  |                   |

到了这里,我们对比着 UPX 的 UnPackMe 来看。

|          |                 |                               |                       |
|----------|-----------------|-------------------------------|-----------------------|
| 0042940B | ^ 0F84 2B010000 | JE 0042953C                   | UnPackMe.0042953C     |
| 00429411 | 8B3D A8094600   | MOV EDI,DWORD PTR DS:[4609A8] | kernel32.VirtualAlloc |
| 00429417 | 6A 04           | PUSH 4                        |                       |
| 00429419 | 68 00200000     | PUSH 2000                     |                       |
| 0042941E | 68 00004000     | PUSH 400000                   |                       |
| 00429423 | 6A 00           | PUSH 0                        |                       |
| 00429425 | FFD7            | CALL NEAR EDI                 |                       |
| 00429427 | 8BF0            | MOV ESI,EAX                   |                       |
| 00429429 | 85F6            | TEST ESI,ESI                  |                       |
| 0042942B | ^ 0F84 F4000000 | JE 00429525                   | UnPackMe.00429525     |
| 00429431 | 6A 04           | PUSH 4                        |                       |

我们可以看到这里实际要调用的 API 函数是 VirtualAlloc,我们还是在返回地址处设置一个断点,运行起来。

|          |               |                             |  |
|----------|---------------|-----------------------------|--|
| 0047E423 | C600 C3       | MOV BYTE PTR DS:[EAX],0C3   |  |
| 0047E426 | ^ E9 34290000 | JMP 00480D5F                |  |
| 0047E42B | 8B0424        | MOV EAX,DWORD PTR SS:[ESP]  |  |
| 0047E42E | 52            | PUSH EDX                    |  |
| 0047E42F | 51            | PUSH ECX                    |  |
| 0047E430 | ^ E9 70360100 | JMP 00491AA5                |  |
| 0047E435 | E8 87670000   | CALL 00484BC1               |  |
| 0047E43A | B8 0CC084C6   | MOV EAX,C684C00C            |  |
| 0047E43F | 53            | PUSH EBX                    |  |
| 0047E440 | 68 F4A32B27   | PUSH 272BA3F4               |  |
| 0047E445 | ^ E9 C62D0000 | JMP 00481210                |  |
| 0047E44A | 5A            | POP EDX                     |  |
| 0047E44B | 8B0424        | MOV EAX,DWORD PTR SS:[ESP]  |  |
| 0047E44E | 50            | PUSH EAX                    |  |
| 0047E44F | 8BC2          | MOV EAX,EDX                 |  |
| 0047E451 | 870424        | XCHG DWORD PTR SS:[ESP],EAX |  |
| 0047E454 | E8 016F0000   | CALL 004A535A               |  |
| 0047E459 | B8 50A8F865   | MOV EAX,65F8A850            |  |
| 0047E45E | 53            | PUSH EBX                    |  |

这里又是填充另一个 RET,我们来看看修改之前的代码是什么:

|          |                 |                                |                       |
|----------|-----------------|--------------------------------|-----------------------|
| 00480D4A | - FF25 A8094600 | JMP NEAR DWORD PTR DS:[4609A8] | kernel32.VirtualAlloc |
| 00480D50 | ^ E9 030E0000   | JMP 00481B58                   | UnPackMe.00481B58     |
| 00480D55 | ^ E9 608FFFFF   | JMP 00479CBA                   | UnPackMe.00479CBA     |
| 00480D5A | ^ E9 CC73FFFF   | JMP 0047812B                   | UnPackMe.0047812B     |
| 00480D5F | 5A              | POP EDX                        |                       |
| 00480D60 | ^ 0F82 C5D6FFFF | JB 0047E42B                    | UnPackMe.0047E42B     |

被修改为 RET 之前是 POP EDX,修改为 RET 以后,就会返回到 480D4A 处去调用实际要调用的 API 函数,即 VirtualAlloc。

|          |                 |                                |                       |
|----------|-----------------|--------------------------------|-----------------------|
| 00480D40 | ^ E9 3A14FFFF   | JMP 0047217F                   | UnPackMe.0047217F     |
| 00480D45 | ^ E8 15000000   | CALL 00480D5F                  | UnPackMe.00480D5F     |
| 00480D4A | - FF25 A8094600 | JMP NEAR DWORD PTR DS:[4609A8] | kernel32.VirtualAlloc |
| 00480D50 | ^ E9 030E0000   | JMP 00481B58                   | UnPackMe.00481B58     |
| 00480D55 | ^ E9 608FFFFF   | JMP 00479CBA                   | UnPackMe.00479CBA     |
| 00480D5A | ^ E9 CC73FFFF   | JMP 0047812B                   | UnPackMe.0047812B     |
| 00480D5F | C3              | RETN                           |                       |
| 00480D60 | ^ 0F82 C5D6FFFF | JB 0047E42B                    | UnPackMe.0047E42B     |

也就是说调用 API 函数的过程是,首先将对应 IAT 项中重定向的值修改为正确的 API 函数地址,然后通过自修改得到 RET 指令,

接着通过该 RET 指令返回到相应的地址处去调用实际要调用的 API 函数。

我们继续往下单步跟踪来验证一下：

|          |      |               |
|----------|------|---------------|
| 00429425 | FFD7 | CALL NEAR EDI |
| 00429427 | 8BF0 | MOV ESI,EAX   |
| 00429429 | 85F6 | TEST ESI,ESI  |

可以看到这里 480D5F 处的 POP EDX 被修改为了 RET,此时我们可以注意到 4609A8 这个 IAT 项中的值已经被修复为正常的 API 函数地址了,接着执行 RET 指令,就会返回到 480D45 处,调用实际要调用的 API 函数。

|          |                 |                                |                       |
|----------|-----------------|--------------------------------|-----------------------|
| 00480D40 | ^ E9 3A14FFFF   | JMP 0047217F                   | UnPackMe.0047217F     |
| 00480D45 | E8 15000000     | CALL 00480D5F                  | UnPackMe.00480D5F     |
| 00480D4A | - FF25 A8094600 | JMP NEAR DWORD PTR DS:[4609A8] | kernel32.VirtualAlloc |
| 00480D50 | ^ E9 030E0000   | JMP 00481B58                   | UnPackMe.00481B58     |
| 00480D55 | ^ E9 608FFFFF   | JMP 00479CBA                   | UnPackMe.00479CBA     |
| 00480D5A | ^ E9 CC73FFFF   | JMP 0047812B                   | UnPackMe.0047812B     |
| 00480D5F | C3              | RETN                           |                       |
| 00480D60 | ^ 0F82 C5D6FFFF | JB 0047E42B                    | UnPackMe.0047E42B     |

好了,下面我来给大家总结一下整个过程:首先获取相应模块的基地址(可能是调用的 LoadLibraryA),然后是获取对应的 API 函数地址(可能是调用 GetProcAddress),接着将 IAT 项中重定向的值修改为正确的 API 函数地址,然后修改自身区段的代码来达到调用实际要调用 API 函数的目的。

好了,本章就到这里,下一章我们来尝试编写脚本修改 ExeCryptor 的 IAT。