

第五十六章-EXECryptor v2.2.50.b 脱壳

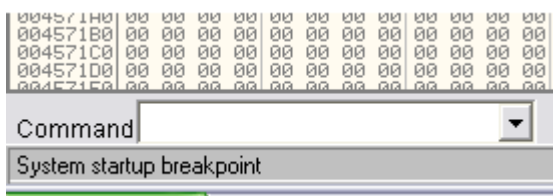
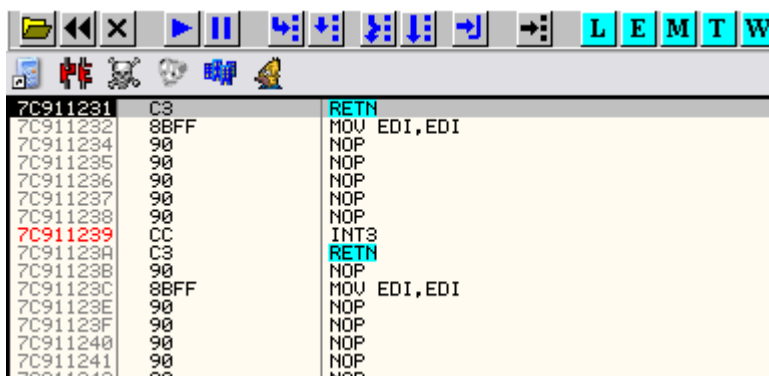
本章我们来看 UnPackMe_ExecCryptor2.2.50.b.exe 这个程序。

我们直接运行该程序,看看效果。

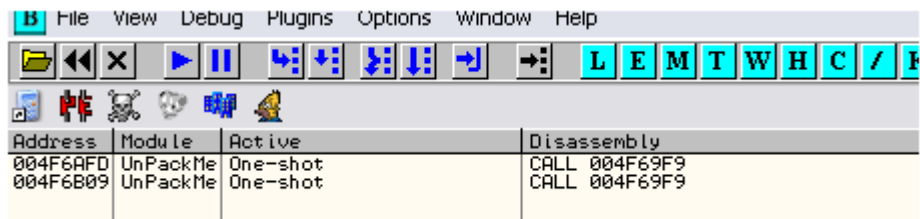


我们可以看到 ExeCryptor UnPackMe 的等级 B 稍微要难一点-Aggressive(译为:咄咄逼人的,这里译为强力的较为恰当)模式开启了,(PS:吓唬人吗?我好怕呀,嘿嘿,o(∩_∩)o)。

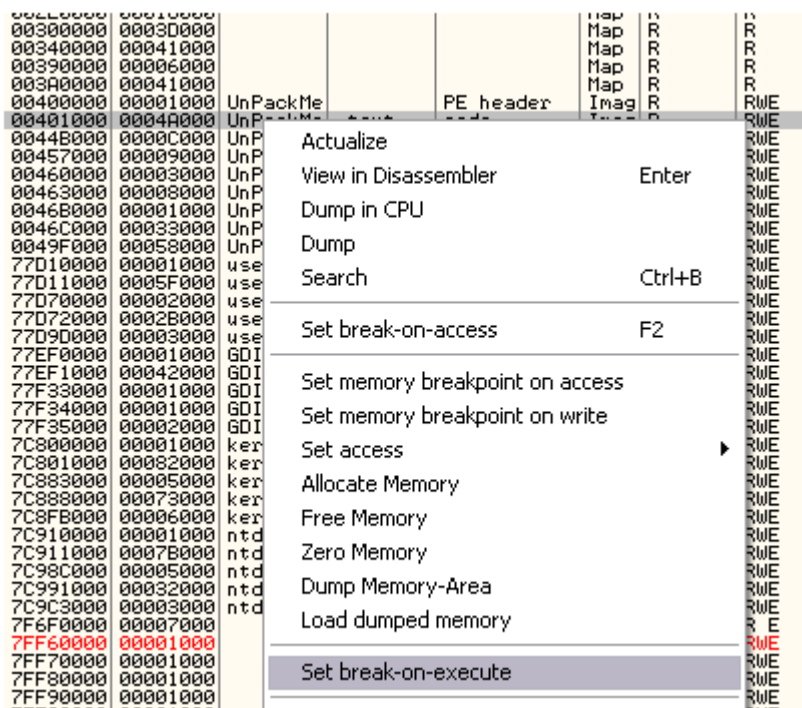
我们直接用上一章调试等级 A 的 OD(无需修改 OllyAdvanced 反反调试插件里面的选项)来加载这个 UnPackMe_ExecCryptor2.2.50.b.exe。



这里我们可以看到断在了系统断点处,我们打开断点列表窗口,删除里面的一次性断点。

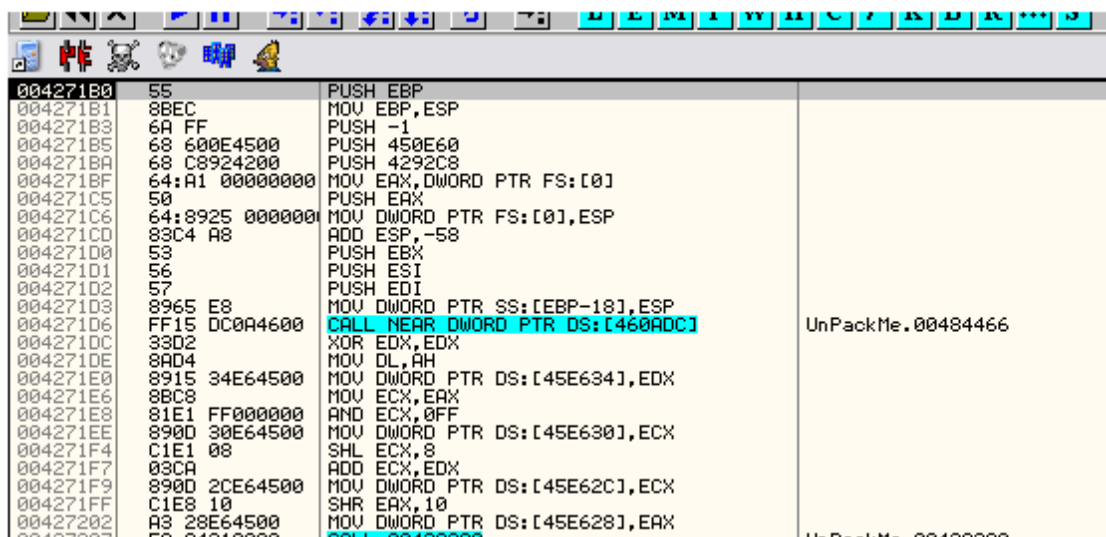


这里我们删除掉这两个一次性断点,接着跟上一章一样对代码段设置 break-on-execute 断点。(PS:我的 OllyBone 插件不好使,我上一章中已经介绍了定位 OEP 的方法,这里就不再赘述了,详情请查阅上一章)



这个 Set break-on-execute 右键菜单项是 OllyBone 插件的一个选项,大家应该还记得吧。

现在我们运行起来。



到这里为止,基本上跟上一章的步骤没什么区别,现在我们删除掉 break-on-execute 断点(PS:我定位 OEP 的方法并没有用到 OllyBone 插件,所以并不需要删除 break-on-execute 断点),继续。

如果大家足够细心的话,就会发现与上一章中的 UnPackMe_ExeCryptor2.2.50.a.exe 相比,这次我们断在 OEP 处时,多出了很多线程。

File View Debug Plugins Options Window Help								
L E M T W H C / K B R ... S								
Ident	Entry	Data block	Last error	Status	Priority	User time	System time	
000000AC	7C810659	7FFD9000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s	
000001C8	7C810659	7FFAC000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s	
000002EC	7C810659	7FFD7000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s	
000003E0	7C810659	7FFAE000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s	
00000410	7C810659	7FFAD000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s	
00000674	7C810659	7FFAA000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s	
000006FC	7C810659	7FFDD000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s	
000009F0	7C810659	7FFDA000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s	
000009FC	7C810659	7FFD6000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s	
00000B38	7C810659	7FFD4000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s	
00000C30	7C810659	7FFDB000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s	
00000C88	7C810659	7FFD5000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s	
00000D3C	7C810659	7FFAB000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s	
00000D44	00270000	7FFDE000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s	
00000DE8	7C810659	7FFAF000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s	
00000E0C	004F6AFD	7FFDF000	ERROR_SUCCESS (00)	Active	32 + 0	0.0468 s	0.0781 s	
00000F0C	7C810659	7FFDC000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s	

很显然,这些线程是用来做检测用的,如果检测到自己正在被调试,就直接退出进程。

如果我们直接运行起来的话,会发现程序直接退出了,这里我们将这些线程都挂起(除了红色标记的主线程以外)。

Ident	Entry	Data block	Last error	Status	Priority	User time	System time	
000000AC	7C810659	7FFD9000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s	
000001C8	7C810659	7FFAC000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s	
000002EC	7C810659	7FFD7000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s	
000003E0	7C810659	7FFAE000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s	
00000410	7C810659	7FFAD000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s	
00000674	7C810659	7FFAA000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s	
000006FC	7C810659	7FFDD000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s	
000009F0	7C810659	7FFDA000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s	
000009FC	7C810659	7FFD6000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s	
00000B38	7C810659	7FFD4000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s	
00000C30	7C810659	7FFDB000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s	
00000C88	7C810659	7FFD5000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s	
00000D3C	7C810659	7FFAB000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s	
00000D44	00270000	7FFDE000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s	
00000DE8	7C810659	7FFAF000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s	
00000E0C	004F6AFD	7FFDF000	ERROR_SUCCESS (00)	Active	32 + 0	0.0468 s	0.0781 s	
00000F0C	7C810659	7FFDC000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s	

这里我们依次在每个线程上单击鼠标右键选择 Suspend(挂起)(除了以红色标记的主线程以外)。

我们直接运行起来,会发现 OD 右下角的状态已经变成了 Running,说明程序已经运行起来了,但是程序的主界面并没有弹出来。

Address	Stack	Procedure	Called from	Frame
0012F8D8	7C91E9C0	Includes ntdll.KiFastSystemCallRet	ntdll.7C91E9BE	0012F960
0012F8DC	7C92901B	ntdll.ZwWaitForSingleObject	ntdll.7C929016	0012F960
0012F964	7C91104B	ntdll.RtlpWaitForCriticalSection	ntdll.7C911046	0012F960
0012F96C	7C924859	ntdll.RtlEnterCriticalSection	ntdll.7C924854	0012F9A4
0012F9A8	7C9266D3	? ntdll.LdrLockLoaderLock	ntdll.7C9266CE	0012F9A4
0012FA1C	7C92659E	? ntdll.LdrGetDllHandleEx	ntdll.7C926599	0012FA18
0012FA38	7C80E494	? <JMP.&ntdll.LdrGetDllHandle>	kernel32.7C80E48F	0012FA34
0012FA88	7C80E5A8	? kernel32.7C80E477	kernel32.7C80E5A6	0012FA84
0012FF0C	7C80E45C	? kernel32.7C80E4B9	kernel32.7C80E457	0012FF08
0012FF24	7C80B6C0	? kernel32.GetModuleHandleW	kernel32.7C80B6BB	0012FF20

好,现在我们在挂起的线程中任意挑选一个出来,查看一下其调用堆栈,我们可以看到调用了 ntdll.dll 中的 ZwWaitForSingleObject,也就说上层实际调用了 WaitForSingleObject 这个 API 函数,说明该线程在等待某个信号量,当前不会继续往下执行了。

下面我们任选一个挂起的线程,这里我选择了第一个挂起的线程,单击鼠标右键选择 Resume,让其恢复运行。


Ident	Entry	Data block	Last error	Status	Priority	User time
000000AC	7C810659	7FFD9000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000
000001C8	7C810659	7FFAC000	ERROR_SUCCESS (00)	Su		
000002EC	7C810659	7FFD7000	ERROR_SUCCESS (00)	Su		
000003E0	7C810659	7FFAE000	ERROR_SUCCESS (00)	Su		
00000410	7C810659	7FFAD000	ERROR_SUCCESS (00)	Su		
00000674	7C810659	7FFAA000	ERROR_SUCCESS (00)	Su		

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
000000AC	7C810659	7FFD9000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
000001C8	7C810659	7FFAC000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
000002EC	7C810659	7FFD7000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
000003E0	7C810659	7FFAE000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000410	7C810659	7FFAD000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000674	7C810659	7FFAA000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s
000006FC	7C810659	7FFD0000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
000009F0	7C810659	7FFDA000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s
000009FC	7C810659	7FFD6000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000B38	7C810659	7FFD4000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000C30	7C810659	7FFDB000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000C88	7C810659	7FFD5000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D3C	7C810659	7FFAB000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D44	00270000	7FFDE000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s
00000DE8	7C810659	7FFAF000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000E0C	004F6AFD	7FFDF000	ERROR_SUCCESS (00)	Active	32 + 0	0.0468 s	0.0781 s
00000F0C	7C810659	7FFDC000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s

我们可以看到 OD 右下角状态仍然是 Running,但是程序的主界面还是没有弹出来,所以我们再次将该线程挂起,继续恢复下一个线程,如果下一个线程被恢复后,程序的主界面还是没有弹出来,我们继续将其挂起,继续恢复下一项,直到某挂起的线程被恢复以后,程序的主界面弹出来为止。

00000C88	7C810659	7FFD5000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D3C	7C810659	7FFAB000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D44	00270000	7FFDE000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s
00000DE8	7C810659	7FFAF000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000E0C	004F6AFD	7FFDF000	ERROR_SUCCESS (00)	Active	32 + 0	0.0781 s	0.1250 s
00000F0C	7C810659	7FFDC000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s

Teddy Rogers / SnD Team 2005



This is an ExeCryptor 2.1.20 UnPackME
(Compressed Resources/Code/Data/Maximum Compression)

Aggressive Mode Enabled
If you unpack it write a tutorial... :)

Aceptar

我这里当恢复 Entry 的值为 270000 的这个线程(不同的机器上这个地址可能不一样)的时候,程序的主界面弹出来了。

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
000000AC	7C810659	7FFD9000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
000001C8	7C810659	7FFAC000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
000002EC	7C810659	7FFD7000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
000003E0	7C810659	7FFAE000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000410	7C810659	7FFAD000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000674	7C810659	7FFAA000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
000006FC	7C810659	7FFD0000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
000009F0	7C810659	7FFDA000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
000009FC	7C810659	7FFD6000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000B38	7C810659	7FFD4000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000C30	7C810659	7FFDB000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000C88	7C810659	7FFD5000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D3C	7C810659	7FFAB000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D44	00270000	7FFDE000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s
00000DE8	7C810659	7FFAF000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000E0C	004F6AFD	7FFDF000	ERROR_SUCCESS (00)	Active	32 + 0	0.0937 s	0.2031 s
00000F0C	7C810659	7FFDC000	ERROR_SUCCESS (00)	Suspended	32 + 0	0.0000 s	0.0000 s

我们定位到 270000 地址处看看。

00270000	55	PUSH EBP
00270001	56	PUSH ESI
00270002	57	PUSH EDI
00270003	53	PUSH EBX
00270004	E8 00000000	CALL 00270009
00270009	58	POP EAX
0027000A	83C0 1D	ADD EAX, 1D
0027000D	50	PUSH EAX
0027000E	64:FF35 00000000	PUSH DWORD PTR FS:[0]
00270015	64:8925 00000000	MOV DWORD PTR FS:[0], ESP
0027001C	53	PUSH EBX
0027001D	FFD7	CALL NEAR EDI
0027001F	EB 12	JMP SHORT 00270033
00270021	8B6424 08	MOV ESP, DWORD PTR SS:[ESP+8]
00270025	64:8F05 00000000	POP DWORD PTR FS:[0]
0027002C	B8 010000C0	MOV EAX, C0000001
00270031	EB 07	JMP SHORT 0027003A
00270033	64:8F05 00000000	POP DWORD PTR FS:[0]
0027003A	83C4 04	ADD ESP, 4
0027003D	5B	POP EBX
0027003E	5F	POP EDI
0027003F	5E	POP ESI
00270040	5D	POP EBP
00270041	50	PUSH EAX
00270042	FFD6	CALL NEAR ESI
00270044	C3	RETN

我们对起始地址为 270000 的这个区段设置一个内存访问断点。

00250000	00006000	Priv	RW	RW
00260000	00001000	Priv	RW	RW
00270000	00001000	Priv	FE	RW
00280000	00001000			RW
002CE000	00002000			RW
002D0000	00003000			RW
002E0000	00016000			R
00300000	0003D000			R
00340000	00041000			R
00390000	00006000			R
003A0000	00041000			R
003F0000	00001000			RWE
00400000	00001000	UnPackMe		RWE
00401000	0004A000	UnPackMe		RWE
00448000	0000C000	UnPackMe		RWE
00457000	0000C000	UnPackMe		RWE
00460000	00003000	UnPackMe		RWE
00463000	00003000	UnPackMe		RWE

接着运行起来,发现并没有断下来,所以我们需要改变一下策略了。

我们没有必要再深入探究这个线程的细节了,我们的主要目的还是为了修复 IAT,既然断不下来,说明该线程涉及到修复 IAT 的可能性比较小。现在我们重启 OD,再次断到 OEP 处。

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
0000002E0	7C810659	7FFD0000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
000000304	7C810659	7FFAE000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
000000478	7C810659	7FFD8000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
000000480	7C810659	7FFAA000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
000000A98	7C810659	7FFAC000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
000000BC4	00270000	7FFDE000	ERROR_SUCCESS (000)	Active	32 + 0	0.0000 s	0.0000 s
000000C14	7C810659	7FFD5000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
000000C48	7C810659	7FFD4000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
000000D10	7C810659	7FFDC000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
000000D64	7C810659	7FFD6000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
000000D70	7C810659	7FFAF000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
000000E64	7C810659	7FFAB000	ERROR_SUCCESS (000)	Active	32 - 15	0.0000 s	0.0000 s
000000F20	7C810659	7FFDB000	ERROR_SUCCESS (000)	Active	32 - 15	0.0156 s	0.0000 s
000000F90	004F6AFD	7FFDF000	ERROR_SUCCESS (000)	Active	32 + 0	0.0468 s	0.0625 s
000000F94	7C810659	7FFAD000	ERROR_SUCCESS (000)	Active	32 + 0	0.0000 s	0.0000 s
000000FC8	7C810659	7FFD9000	ERROR_SUCCESS (000)	Active	32 + 0	0.0000 s	0.0000 s
000000FF4	7C810659	7FFD7000	ERROR_SUCCESS (000)	Active	32 + 0	0.0000 s	0.0000 s

下面我们来验证一下是不是仅仅只需要主线程以及线程函数入口地址为 270000 的这两个线程该程序就能正常运行起来了,如果是的话,那就最好不过了,如果不是的话,我们就还需要定位让程序正常运行必需的第三个或者更多的线程。

好,下面我们将这两个线程以外的其他线程都挂起。

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
000002E0	7C810659	7FFD0000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000304	7C810659	7FFAE000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000478	7C810659	7FFD8000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000480	7C810659	7FFA0000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000A08	7C810659	7FFAC000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000BC4	00270000	7FFDE000	ERROR_SUCCESS (000)	Active	32 + 0	0.0000 s	0.0000 s
00000C14	7C810659	7FFD5000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000C48	7C810659	7FFD4000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D10	7C810659	7FFDC000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D64	7C810659	7FFD6000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000D70	7C810659	7FFAF000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000E64	7C810659	7FFAB000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0000 s	0.0000 s
00000F20	7C810659	7FFD0000	ERROR_SUCCESS (000)	Suspended	32 - 15	0.0156 s	0.0000 s
00000F90	004F6AFD	7FFD0000	ERROR_SUCCESS (000)	Active	32 + 0	0.0468 s	0.0625 s
00000F94	7C810659	7FFAD000	ERROR_SUCCESS (000)	Suspended	32 + 0	0.0000 s	0.0000 s
00000FC8	7C810659	7FFD9000	ERROR_SUCCESS (000)	Suspended	32 + 0	0.0000 s	0.0000 s
00000FF4	7C810659	7FFD7000	ERROR_SUCCESS (000)	Suspended	32 + 0	0.0000 s	0.0000 s

另外,我们注意一下现在这两个线程的优先级,我们会发现线程函数入口地址为 270000 的这个线程的优先级与主线程的优先级是一样的,而其他线程的优先级比它们两个都低。

我们运行起来看看会发生什么。



也就是说该程序要想正常运行,只需要这两个线程即可。

这里我给大家一些小小的建议:在脱一些强壳的时候,大家没有必要按照一些教程的步骤来生搬硬套,大家要学会灵活变通。

有时候,看到网上的一些教程的脱壳步骤(譬如:按 5 次 F8,3 次 F7 就可以到达 OEP 处了+_),说实话这种教程有点滑稽可笑,不免有误人子弟之嫌。

这里就拿我当前的这个脱壳方案来说吧,说不定换了一台机器就行不通了也说不定。不可控的因素太多了,有时候可能同一款壳在不同的机器上,反反调试插件选项的配置上可能也会不同,琳琳种种的这些东西可能会花费我们大量的时间,所以大家在平时脱壳的过程中要善于总结经验,活学活用。

好了,废话不多说,我们继续。

上一章中的 UnPackMe_ExeCryptor2.2.50.a.exe 这个程序,我们是用内存写入断点来定位修复 IAT 项的指令的,同理,这里我们还是来尝试对 OEP 下面调用的第一个 API 函数对应的 IAT 项设置内存写入断点,接着利用 OD 的 Trace into 功能来进行自动跟踪,自动跟踪的时间大约要花费 5 分钟左右,大家耐心等待一下吧(自动跟踪的记录文本见附件)。

Address	Disassembly	Comment
00483FE3	MOV DWORD PTR DS:[460ADC],EAX	kernel32.GetVersion
00483FE9	LEA EAX,DWORD PTR DS:[494670]	
00483FEF	PUSH 488ADF	
00483FF4	JMP 00482FA9	UnPackMe.00482FA9
00483FF9	MOV EAX,DWORD PTR SS:[ESP]	
00483FFC	CALL 00493AB4	UnPackMe.00493AB4

大家应该还记得上一章中有一条所有待修复的 IAT 项都会执行的指令吧,这里我们在自动跟踪的指令序列中也可能找到它,只不过地址比上一章中那个地址稍微要高一点。

Step	Module	Address	Disassembly	Registers
8.	Main	UnPackMe.0048158C	POP ECX	ECX=7C81120A, ESP=0012FE08
7.	Main	UnPackMe.0048158D	POP EBP	ESP=0012FE0C, EBP=0012FF38
6.	Main	UnPackMe.0048158E	RETN	ESP=0012FE10
5.	Main	UnPackMe.00483420	MOV EAX,DWORD PTR SS:[EBP-C]	EAX=7C81110A
4.	Main	UnPackMe.00483423	MOV ESP,EBP	ESP=0012FF38
3.	Main	UnPackMe.00483425	POP EBP	ESP=0012FF3C, EBP=0012FFC0
2.	Main	UnPackMe.00483426	RETN	ESP=0012FF40
1.	Main	UnPackMe.00475937	RETN	ESP=0012FF44
0.	Main	UnPackMe.00483FE3	MOV DWORD PTR DS:[460ADC],EAX	

这次 MOV EAX,DWORD PTR SS:[EBP-C]这条指令的地址为 483420,好,下面我们对上一章中编写的那个脚本进行相应的修改,我们将要设置硬件执行断点的地址修改为 483423,当断到这条指令处时,EAX 中已经保存正确的 API 函数地址。

上一章中的脚本如下:

```
-----  
  
var table  
var content  
    mov table,460818  
  
start:  
    cmp table,460F28  
    ja final  
    cmp [table],50000000  
    ja ToSkip  
  
    mov content,[table]  
    cmp content,0  
    je ToSkip  
    log content  
    log table  
  
    mov eip,content  
    bphws 47691F,"x"  
    mov [47691F],0  
    mov [476920],0  
    cob ToRepair  
    run  
  
ToRepair:  
    cmp eip,7C91E88E  
    je ToSkip  
    log eax  
    mov [table],eax  
    run  
  
ToSkip:  
    add table,4  
    jmp start  
  
final:  
    ret
```

大家应该还记得上一章中 MOV EAX,DWORD PTR SS:[EBP -C]这条指令的地址吧,执行了这指令后,EAX 中将会保存待修复 IAT 项对应的 API 函数地址。

0047691C Main MOV EAX,DWORD PTR SS:[EBP-C] ; EAX=77DA6BF0

0047691F Main MOV ESP,EBP ; ESP=0012FFB0

而本章这里:

00483420 > 8B45 F4 MOV EAX,DWORD PTR SS:[EBP-C] ; kernel32.GetVersion

00483423 . |8BE5 MOV ESP,EBP

所以我们这里需要将该脚本中的 47691F 替换成 483423。

```
var table
var content
    mov table,460818

start:
    cmp table,460F28
    ja final
    cmp [table],50000000
    ja ToSkip

    mov content,[table]
    cmp content,0
    je ToSkip
    log content
    log table

    mov eip,content
    bphws 483423,"x"
    mov [483423],0
    mov [483424],0
    cob ToRepair
    run

ToRepair:
    cmp eip,7C91E88E
    je ToSkip
    log eax
    mov [table],eax
    run

ToSkip:
    add table,4
    jmp start

final:
    ret
```

这里我对做了修改的地址用红色标注出来了,像 IAT 的起始地址以及结束地址这些我未做修改的地址以及常量值我用蓝色标注出

来了。50000000 这个常量值是用来判断 IAT 项是否被重定向的依据,而 7C91E88E 这个地址是我这里 ZwTerminateProcess 这个 API 函数的入口地址,大家应该还记得吧?我们继续。

好了,现在我们重启 OD,再次断到 OEP 处,别忘了删除掉 break-on-execute 断点,还有就是去掉忽略内存访问异常这个选项的对勾。

接下来,将主线程以及线程函数入口地址为 270000 的这两个线程以外的其他线程都挂起,然后执行脚本。

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	F0 68 DA 77 1B 76 DA 77 F4 EA DA 77 E7 EB DA 77	-k r w + v r w 7 U r w 5 U r w
00460828	83 78 DA 77 00 00 00 00 CF 65 C3 58 08 03 C4 58	3 k r w . . . 8 e t X i - X
00460838	00 00 00 00 04 6A EF 77 66 95 EF 77 89 6A EF 77 e j , w f o , w e j , w
00460848	F3 AD EF 77 ED 09 EF 77 99 8B EF 77 C0 B5 EF 77	% i , w y , w o i , w t a , w
00460858	2A 7C EF 77 B2 7C EF 77 77 53 F2 77 1E C9 F1 77	* j , w 8 i , w w S = w A F + w
00460868	0C BC EF 77 52 04 EF 77 FA 80 EF 77 F1 D0 EF 77	. , w R E , w . l , w t i , w
00460878	51 B2 EF 77 26 D5 EF 77 2A E3 EF 77 5F 39 F2 77	0 8 , w & , w * b , w _ 9 = w
00460888	71 B4 EF 77 2E AD EF 77 E1 61 EF 77 B8 85 EF 77	q i , w . + , w p a , w 0 a , w
00460898	CC D2 EF 77 43 70 EF 77 FB EA F0 77 12 83 EF 77	l f e , w C p , w i u - w + a , w
004608A8	01 72 F0 77 A9 34 F0 77 D5 93 EF 77 68 EF EF 77	0 r - w 0 4 - w i o , w h , w
004608B8	AA D2 EF 77 B2 6F EF 77 3F 38 F2 77 D6 E8 EF 77	- e , w 8 o , w ? 8 = w i f , w
004608C8	68 E0 EF 77 00 60 EF 77 90 5B EF 77 6D AC EF 77	h o , w . , w e l , w m 8 , w
004608D8	94 6C F0 77 22 80 EF 77 3D C8 F1 77 3D 6D F0 77	o l - w , l , w = e + w m = w
004608E8	6F C0 EF 77 85 78 EF 77 26 D9 EF 77 FB 5E EF 77	o l , w a c , w & , w i ^ , w
004608F8	36 8A EF 77 FC 8A EF 77 0F 62 EF 77 49 5E EF 77	6 e , w e , w * b , w i ^ , w
00460908	97 5D EF 77 1A 9A EF 77 6B FA EF 77 7B C9 F0 77	u j , w + u , w k , w c f r - w
00460918	DA 98 F2 77 1A 40 F2 77 55 EA EF 77 C5 61 EF 77	r y = w + 0 = w U U , w t + a , w
00460928	70 E6 EF 77 F0 81 EF 77 2D 6C EF 77 98 6E EF 77	p y , w - u , w - l , w y n , w
00460938	4F 83 EF 77 09 ED EF 77 EB AA EF 77 26 69 F0 77	0 a , w . y , w u - w & i - w
00460948	B1 95 EF 77 6F B0 EF 77 8A 5A EF 77 E9 49 F2 77	8 o , w c , w e z , w u l = w
00460958	26 F1 F0 77 C9 D0 F0 77 51 E0 F0 77 33 8C EF 77	& t - w f i - w Q 0 - w 3 i , w
00460968	6C EC EF 77 29 94 EF 77 00 00 00 00 6B 17 80 7C	l y , w i o , w . . . k 8 C !
00460978	04 A7 80 7C 51 0E 81 7C EE 1E 80 7C 1D 2F 81 7C	e 0 C ! Q 8 u i ! - A C ! # / u !
00460988	00 00 00 00 09 2A 81 7C DA CD 81 7C 16 1E 80 7C # u i ! - u i ! - A C !
00460998	15 99 80 7C 45 00 88 00 45 00 88 00 45 00 88 00	3 0 C ! E . e . E . e . E . e .
004609A8	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
004609B8	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
004609C8	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
004609D8	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
004609E8	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
004609F8	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
00460A08	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
00460A18	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
00460A28	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
00460A38	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
00460A48	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
00460A58	45 00 88 00 45 1B 82 7C 00 00 00 00 00 00 00 00	E . e . 8 + e !
00460A68	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

我们可以看到脚本确实是在修复 IAT,但是我们会发现部分修复后的值是错误的。

00460978	04 A7 80 7C 51 0E 81 7C EE 1E 80 7C 1D 2F 81 7C	e 0 C ! Q 8 u i ! - A C ! # / u !
00460988	00 00 00 00 09 2A 81 7C DA CD 81 7C 16 1E 80 7C # u i ! - u i ! - A C !
00460998	15 99 80 7C 45 00 88 00 45 00 88 00 45 00 88 00	3 0 C ! E . e . E . e . E . e .
004609A8	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
004609B8	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
004609C8	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
004609D8	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
004609E8	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .

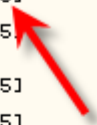
我们查看一下日志信息会发现从 46099C 这一项开始,修复后的值都是错误的。

Address	Hex dump	ASCII
0046097C	51 0E 81 7C EE 1E 80 7C 1D 2F 81 7C 00 00 00 00	Q 8 u i ! - A C ! # / u ! . . .
0046098C	09 2A 81 7C DA CD 81 7C 16 1E 80 7C 15 99 80 7C	. # u i ! - u i ! - A C ! 3 0 C !
0046099C	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
004609AC	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
004609BC	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
004609CC	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
004609DC	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
004609EC	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
004609FC	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
00460A0C	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
00460A1C	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
00460A2C	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
00460A3C	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
00460A4C	45 00 88 00 45 00 88 00 45 00 88 00 45 00 88 00	E . e . E . e . E . e . E . e .
00460A5C	A5 1B 82 7C 00 00 00 00 00 00 00 00 00 00 00	8 + e !
00460A6C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00460A7C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00460A8C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00460A9C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00460AAC	00 00 00 00 00 00 00 00 94 F8 46 00 47 BD 48 00 8 ° F . G c H .
00460ABC	F9 64 47 00 CB FE 47 00 58 04 48 00 A1 40 48 00	- d G . 7 F = G . X + H . i 0 H .
00460ACC	9D 9B 48 00 7A 4C 48 00 26 87 49 00 7D 58 49 00	0 6 H . z L H . & 9 I . 3 X I .
00460ADC	66 44 48 00 F7 8A 48 00 F8 CA 46 00 FF 52 47 00	f 0 H . 6 6 H . ° 4 F . 8 A R .

```

00483423 Access violation when writing to [7C809915]
          contenido: 00472EBD | Entry address
          tabla: 0046099C
0047F5D8 Access violation when reading [00880045]
          eax: 00880045
0047F5D8 Access violation when reading [00880045]
          contenido: 00472719 | Entry address
          tabla: 004609A0
004824D3 Access violation when reading [00880045]
          eax: 00880045
004824D3 Access violation when reading [00880045]
          contenido: 0047C259 | Entry address
          tabla: 004609A4
0047B961 Access violation when reading [00880045]
          eax: 00880045
0047B961 Access violation when reading [00880045]
          contenido: 0048A03E | Entry address
          tabla: 004609A8
004770E5 Access violation when reading [00880045]
          eax: 00880045
004770E5 Access violation when reading [00880045]
          contenido: 0048EF06 | Entry address
          tabla: 004609AC
00476492 Access violation when reading [00880045]
          eax: 00880045
00476492 Access violation when reading [00880045]
          contenido: 0047DDE8 | Entry address
          tabla: 004609B0
0048A42D Access violation when reading [00880045]
          eax: 00880045
0048A42D Access violation when reading [00880045]
          contenido: 0049E46F
          tabla: 004609B4
004872D2 Access violation when reading [00880045]
          eax: 00880045
004872D2 Access violation when reading [00880045]
          contenido: 00480313 | Entry address
          tabla: 004609B8
00482C53 Access violation when reading [00880045]
          eax: 00880045
00482C53 Access violation when reading [00880045]

```



从日志信息中我们可以看到从这个地址开始,读取 880045 这个内存单元中的值的时候发生了异常。

如果我们将脚本修改为如下形式:

ToRepair:

```
cmp eip,483423
```

```
jne to ToSkip
```

```
log eax
```

```
mov [table],eax
```

```
run
```

这样的话,当发生异常时,就会跳转到 ToSkip 标签处继续遍历下一个 IAT 项,这样做的话就不会保存这些错误的值了。

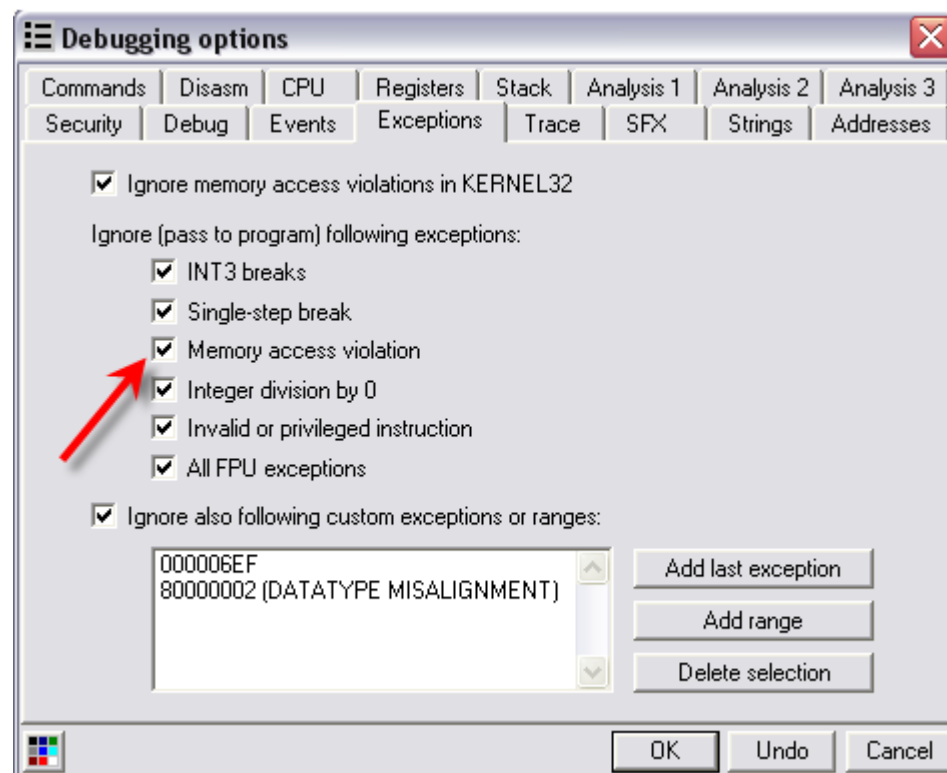
这里我们执行该脚本,会发现还是出错。

到底是哪里的问题呢?我们单步调试该脚本会发现:

上一章中我们利用了发生异常时,程序会调用 ZwTerminateProcess 来结束进程这一特征。但是这里我们会发现发生异常时该程序并没有调用 ZwTerminateProcess,难道是我们需要忽略内存访问异常吗?好,我们忽略掉内存访问异常试试。

Address	Message
00483423	tabla: 00460A30 Hardware breakpoint 3 at UnPackMe.00483423 eax: 7C809740 ! kernel32.TlsGetValue
00483423	Access violation when writing to [7C809740]
7C91E88E	Hardware breakpoint 2 at ntdll.ZwTerminateProcess contenido: 0049EE49 tabla: 00460A34
00483423	Hardware breakpoint 3 at UnPackMe.00483423 eax: 7C809927 ! kernel32.LocalReAlloc
00483423	Access violation when writing to [7C809927]
7C91E88E	Hardware breakpoint 2 at ntdll.ZwTerminateProcess contenido: 00474D6A tabla: 00460A38 ! ASCII "jMG"
00483423	Hardware breakpoint 3 at UnPackMe.00483423 eax: 7C809BC5 ! kernel32.TlsSetValue
00483423	Access violation when writing to [7C809BC5]
7C91E88E	Hardware breakpoint 2 at ntdll.ZwTerminateProcess contenido: 004733CA tabla: 00460A3C
00483423	Hardware breakpoint 3 at UnPackMe.00483423 eax: 7C911005 ! ntdll.RtlEnterCriticalSection
00483423	Access violation when writing to [7C911005]
7C91E88E	Hardware breakpoint 2 at ntdll.ZwTerminateProcess contenido: 00498571 tabla: 00460A40
00483423	Hardware breakpoint 3 at UnPackMe.00483423 eax: 7C8123B9 ! kernel32.GlobalReAlloc
00483423	Access violation when writing to [7C8123B9]

诶,我们会发现现在发生了异常以后,会调用 ZwTerminateProcess 了,也就是说这里我们应该忽略掉内存访问异常。



好,现在我们勾选上忽略内存访问异常这个选项,然后再次执行脚本,看看效果。

Address	Hex dump	ASCII
00460818	F8 58 DA 77 18 76 DA 77 F4 EA DA 77 E7 E8 DA 77	-k rw+vrwll0 ruf0 rw
00460828	83 78 DA 77 00 00 00 CF E5 C3 58 08 03 C4 58	an rw+vrwll0 ruf0 rw
00460838	00 00 00 00 04 6A EF 77 66 95 EF 77 89 6A EF 77	...EJ+vrwll0 ruf0 rw
00460848	F3 40 EF 77 ED 09 EF 77 99 88 EF 77 C0 B5 EF 77	%i+vrwll0 ruf0 rw
00460858	2A 7D EF 77 B2 7C EF 77 77 53 F2 77 1E C9 F1 77	*j+vrwll0 ruf0 rw
00460868	0C 8C EF 77 52 04 EF 77 FA 80 EF 77 F1 00 EF 77	.+vrwll0 ruf0 rw
00460878	51 B2 EF 77 26 05 EF 77 2A E3 EF 77 5F 39 F2 77	00+vrwll0 ruf0 rw
00460888	71 84 EF 77 2E AD EF 77 E1 61 EF 77 B8 85 EF 77	01+vrwll0 ruf0 rw
00460898	CC D2 EF 77 43 70 EF 77 F6 EA EF 77 12 83 EF 77	02+vrwll0 ruf0 rw
004608A8	01 72 00 77 A9 34 F0 77 05 93 EF 77 68 EF EF 77	03+vrwll0 ruf0 rw
004608B8	AA D2 EF 77 B2 6F EF 77 3F 38 F2 77 D6 E8 EF 77	04+vrwll0 ruf0 rw
004608C8	68 E0 EF 77 00 60 EF 77 90 58 EF 77 60 AC EF 77	05+vrwll0 ruf0 rw
004608D8	94 6C F0 77 22 80 EF 77 30 C8 F1 77 30 60 F0 77	06+vrwll0 ruf0 rw
004608E8	6F C0 EF 77 85 7B EF 77 26 09 EF 77 F8 5E EF 77	07+vrwll0 ruf0 rw
004608F8	36 8A EF 77 FC 8A EF 77 0F 62 EF 77 49 5E EF 77	08+vrwll0 ruf0 rw
00460908	97 73 EF 77 1A 9F EF 77 60 EA EF 77 78 09 F0 77	09+vrwll0 ruf0 rw
00460918	DA 9E F2 77 1A 40 F2 77 55 EA EF 77 C6 61 EF 77	0A+vrwll0 ruf0 rw
00460928	70 E6 EF 77 F0 81 EF 77 20 6C EF 77 98 6E EF 77	0B+vrwll0 ruf0 rw
00460938	4F 83 EF 77 09 ED EF 77 EB AA EF 77 26 69 F0 77	0C+vrwll0 ruf0 rw
00460948	B1 95 EF 77 6F B0 EF 77 8A 5A EF 77 E9 49 F2 77	0D+vrwll0 ruf0 rw
00460958	26 F1 F0 77 C9 D0 F0 77 51 E0 F0 77 33 8C EF 77	0E+vrwll0 ruf0 rw
00460968	6C EC EF 77 29 94 EF 77 00 00 00 68 17 80 7C	0F+vrwll0 ruf0 rw
00460978	77 00 80 7C 51 0E 81 7C 0E 80 7C 10 20 80 7C	10+vrwll0 ruf0 rw
00460988	40 70 84 7C 09 20 81 7C DA CD 81 7C 1E 80 7C	11+vrwll0 ruf0 rw
00460998	15 99 80 7C F8 0E 81 7C B6 2B 81 7C E4 9A 80 7C	12+vrwll0 ruf0 rw
004609A8	51 9A 80 7C E3 14 82 7C BF 50 83 7C E8 80 83 7C	13+vrwll0 ruf0 rw
004609B8	A8 CC 80 7C 62 2E 86 7C 77 DF 81 7C E7 4A 81 7C	14+vrwll0 ruf0 rw
004609C8	5B CF 81 7C 08 2F 81 7C 90 47 84 7C 0C 8A 83 7C	15+vrwll0 ruf0 rw
004609D8	90 A4 80 7C CF BC 80 7C 62 D2 80 7C 62 15 81 7C	16+vrwll0 ruf0 rw
004609E8	77 00 80 7C 51 0E 81 7C 0E 80 7C 10 20 80 7C	17+vrwll0 ruf0 rw
004609F8	FD 73 82 7C D4 09 82 7C 30 04 82 7C A7 27 81 7C	18+vrwll0 ruf0 rw
00460A08	76 2E 81 7C 8B 82 85 7C A9 60 83 7C 45 1C 83 7C	19+vrwll0 ruf0 rw
00460A18	77 0A 81 7C 3C 15 81 7C FE 4F 83 7C 54 50 83 7C	20+vrwll0 ruf0 rw
00460A28	23 2C 81 7C 72 67 83 7C 40 97 80 7C 27 09 83 7C	21+vrwll0 ruf0 rw
00460A38	C5 9B 80 7C 05 10 91 7C B9 23 81 7C ED 10 91 7C	22+vrwll0 ruf0 rw
00460A48	B9 4C 83 7C 8A 18 92 7C 9F 20 81 7C F1 9E 80 7C	23+vrwll0 ruf0 rw
00460A58	F0 38 81 7C A5 1B 82 7C 44 28 83 7C BC 22 83 7C	24+vrwll0 ruf0 rw
00460A68	61 23 83 7C 47 9B 80 7C 41 26 81 7C 0B 0B 81 7C	25+vrwll0 ruf0 rw
00460A78	87 00 81 7C 0E 18 80 7C 24 1A 80 7C F5 0D 80 7C	26+vrwll0 ruf0 rw
00460A88	FE DD 80 7C 01 BE 80 7C 0F AC 80 7C A0 F7 82 7C	27+vrwll0 ruf0 rw
00460A98	BB 0B 83 7C A1 BA 80 7C EB 98 80 7C 15 A4 80 7C	28+vrwll0 ruf0 rw
00460AA8	66 E8 80 7C EC E7 80 7C 01 9E 80 7C 79 9E 80 7C	29+vrwll0 ruf0 rw
00460AB8	74 0D 83 7C F8 9B 80 7C D4 A0 80 7C B6 BD 80 7C	30+vrwll0 ruf0 rw
00460AC8	41 4D 83 7C 28 97 80 7C 19 FF 80 7C 82 FE 80 7C	31+vrwll0 ruf0 rw
00460AD8	CF B4 80 7C DA 11 81 7C 06 97 80 7C 19 B2 85 7C	32+vrwll0 ruf0 rw
00460AE8	AC 17 82 7C AB 1E 83 7C EE 86 82 7C 69 3F 87 7C	33+vrwll0 ruf0 rw
00460AF8	03 DC 81 7C 11 13 87 7C 8B B5 81 7C 39 0A 87 7C	34+vrwll0 ruf0 rw
00460B08	20 99 80 7C 9C 92 80 7C 61 0A 87 7C 42 24 80 7C	35+vrwll0 ruf0 rw
00460B18	2B 8C 81 7C F9 2F 87 7C 44 30 87 7C 0A 3C 87 7C	36+vrwll0 ruf0 rw
00460B28	EE 61 83 7C 69 BC 80 7C 80 34 87 7C D3 34 87 7C	37+vrwll0 ruf0 rw
00460B38	1C 3B 87 7C 77 1D 80 7C A0 AD 80 7C DE A8 80 7C	38+vrwll0 ruf0 rw
00460B48	39 2F 81 7C 25 CF 81 7C 8D 10 87 7C B1 4E 83 7C	39+vrwll0 ruf0 rw
00460B58	D9 37 81 7C 31 03 92 7C 40 03 92 7C D7 ED 80 7C	40+vrwll0 ruf0 rw
00460B68	2D FD 80 7C 2F FC 80 7C DE 2A 81 7C 11 01 81 7C	41+vrwll0 ruf0 rw
00460B78	89 BE 80 7C B5 9F 80 7C 97 CC 80 7C B1 2E 83 7C	42+vrwll0 ruf0 rw
00460B88	8D 99 80 7C 10 2E 83 7C 2F 99 80 7C 7A 97 80 7C	43+vrwll0 ruf0 rw
00460B98	66 97 80 7C A1 B6 80 7C 97 CC 80 7C 00 00 00 00	44+vrwll0 ruf0 rw
00460BA8	C0 48 0F 77 38 4C 0F 77 94 A5 11 77 59 48 0F 77	45+vrwll0 ruf0 rw
00460BB8	82 4E 0F 77 98 D4 11 77 98 50 0F 77 4F 50 0F 77	46+vrwll0 ruf0 rw
00460BC8	10 50 0F 77 3F 50 0F 77 D9 66 0F 77 50 48 0F 77	47+vrwll0 ruf0 rw
00460BD8	55 4C 0F 77 C2 4B 0F 77 95 D2 11 77 80 5D 15 77	48+vrwll0 ruf0 rw
00460BE8	00 00 00 00 C1 70 A8 7C B0 70 A8 7C B0 0E A5 7C	49+vrwll0 ruf0 rw
00460BF8	00 00 00 00 EA D6 D1 77 93 B6 D5 77 7D B5 D5 77	50+vrwll0 ruf0 rw

我们可以看到这次就没有问题了。

完整的脚本如下：

```

var table

var content

    mov table,460818

start:

    cmp table,460F28

    ja final

    cmp [table],50000000

    ja ToSkip

    mov content,[table]

    cmp content,0

    je ToSkip

    log content

    log table

```

```

mov eip,content

bphws 483423,"x"

mov [483423],0

mov [483424],0

cob ToRepair

run

```

ToRepair:

```

cmp eip,483423

jne ToSkip

log eax

mov [table],eax

run

```

ToSkip:

```

add table,4

jmp start

```

final:

```
ret
```

这里大家要记住在执行该脚本之前,需要手动在 ZwTerminateProcess 这个 API 函数入口处设置一个硬件执行断点,还要勾选上忽略内存访问异常的选项,还没完,被忘了挂起线程。

好,我们重启 OD,断到 OEP 处,删除掉 break-on-execute 断点,接着执行该脚本。

The screenshot shows the OllyDbg interface. The assembly window displays the following code:

```

7C91E8B8 B8 03010000 MOV EAX,103
7C91E8BD BA 0003FE7F MOV EDX,7FFE0300
7C91E8C2 FF12 CALL NEAR DWORD PTR DS:[EDX]
7C91E8C4 C3 RETN
7C91E8C5 8D49 00 LEA ECX,DWORD PTR DS:[ECX]
7C91E8C8 90 NOP
7C91E8C9 90 NOP
7C91E8CA 90 NOP
7C91E8CB 90 NOP
7C91E8CC 90 NOP
7C91E8CD B8 04010000 MOV EAX,104
7C91E8D0 B8 00000000 MOV EAX,0
EAX=0004C70C

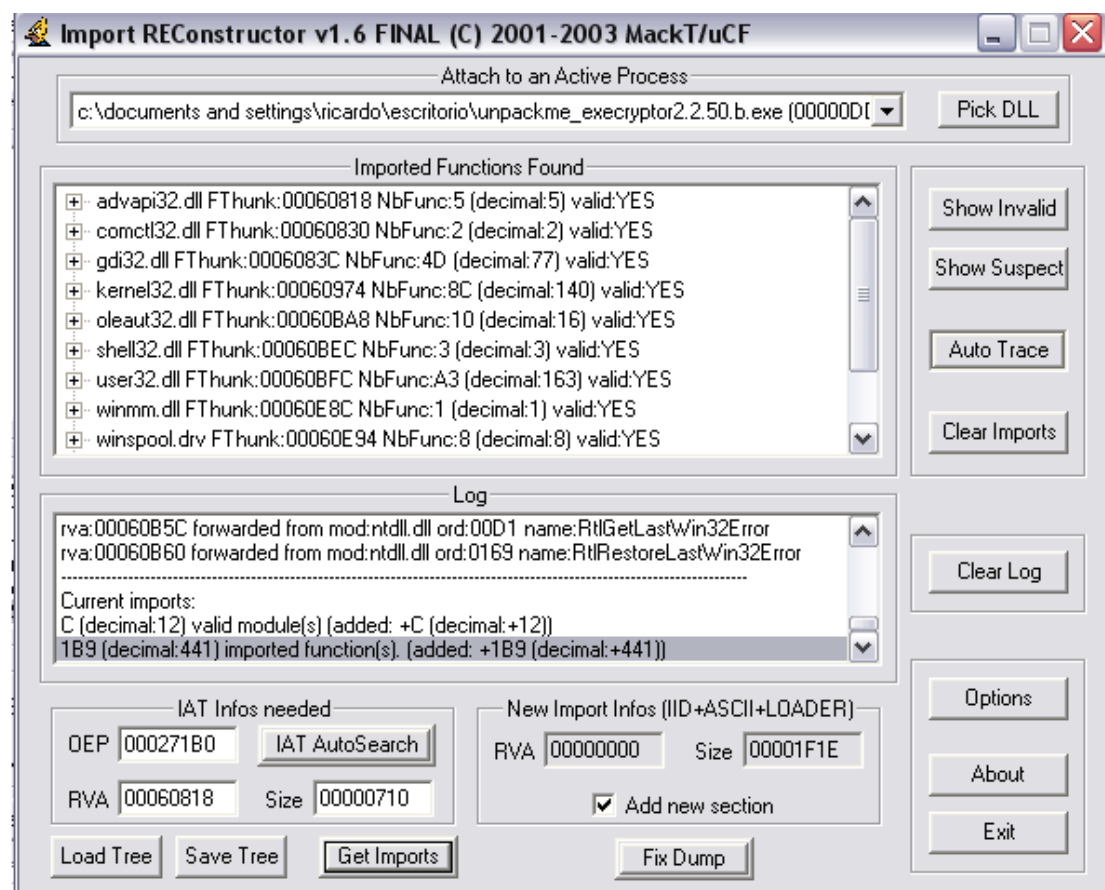
```

The memory dump window shows the following data:

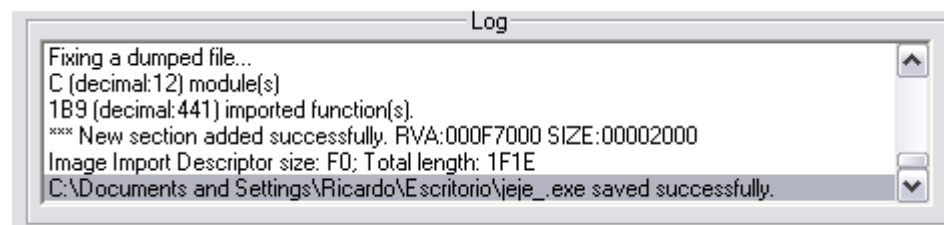
Address	Hex dump	ASCII
00460D88	FE EC 03 77 83 F7 04 77 DE F2 D2 77 DF 1A D3 77	üëwã-ëwí=ëw→ëw
00460D89	F6 F0 04 77 9C F3 04 77 33 F2 D2 77 6C C9 D1 77	+ëw&%ëw3=ëwlfëw
00460D8A	F6 88 D1 77 B8 96 D1 77 0C 94 D1 77 61 C6 D3 77	+iëw@üëw.öëwasëw
00460D8B	81 E5 D2 77 80 03 D3 77 55 E6 D1 77 AD A8 D1 77	üöëwÇëwUpëw+ëw
00460D8C	EA 04 D5 77 24 13 D2 77 58 BF D1 77 33 B9 D1 77	ü♦'w\$!ëwXëwëw
00460D8D	65 F6 04 77 2F B7 D1 77 B4 F6 04 77 6C BF D1 77	e÷ëw/äëwlfëwlëw
00460D8E	F5 B5 D1 77 24 15 D3 77 E2 C2 D1 77 29 69 D5 77	Säëw\$Sëw0tëw)l'w
00460D8F	DF BA D5 77 8C 14 D2 77 4C 1F D3 77 F9 D7 D1 77	'wlfëwLëw-iëw
00460D90	F7 D6 D1 77 65 C4 D1 77 04 B6 D1 77 C8 BD D1 77	-iëwe-öwëäëwëëëw
00460D91	AE B6 D1 77 CD 48 D2 77 3E 0B D2 77 C7 86 D1 77	<<äëw=Hëw>öëwäëw
00460D92	9D 86 D1 77 26 BF D1 77 3F B5 D1 77 69 D8 D1 77	@äëw&ëw?äëwlëw
00460D93	85 CB D1 77 71 BE D1 77 6E C6 D1 77 9D 8F D1 77	äfrëwq#ëwnäëw@äëw
00460D94	FD BE D1 77 31 86 D1 77 17 E9 D3 77 0E EE D4 77	*#ëwläëw#üëw.-ëw
00460D95	9A F3 D2 77 B5 37 D2 77 78 9E D1 77 88 EE D4 77	ü%ëwA7ëwXäëwl-ëw
00460D96	00 00 00 00 F7 A8 B1 76 00 00 00 C8 74 F8 72üëwU....ëwëw
00460D97	73 66 F9 72 87 72 F8 72 43 80 F8 72 67 37 F9 72	sf-rëwëwëwëwëwëw
00460D98	FB 41 F9 72 67 83 F8 72 90 53 F8 72 00 00 00 00	'A-rëwëwëwëwëw...
00460D99	CE 00 37 76 7C 86 37 76 B0 86 37 76 33 25 36 76	fr.7v!ä7vëwä7vëw
00460D9A	1E 31 36 76 D8 7C 37 76 89 C2 37 76 CD 46 38 76	▲16v!i7vëw7v=F8v
00460D9B	CE EE 36 76 00 00 00 00 48 D0 4C 77 9C C8 40 77	fr-6v...HsLwëwfrmw
00460D9C	CC 42 4F 77 2C D0 4C 77 DA F6 4C 77 73 33 50 77	frB0w.sLwfrLws3Pw
00460D9D	10 64 40 77 03 0E 52 77 33 0F 52 77 40 A6 54 77	frdhwëwRw3Rwëwëw
00460D9E	F1 A7 54 77 92 9C 4F 77 6F 57 52 77 99 33 4E 77	ëwTwëwëwëwëwëwëw
00460D9F	B2 5D 4E 77 90 C0 5A 77 00 00 00 00 F3 F0 CC 74	ëwTwëwëwëwëwëwëw
00460DA0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

我们可以看到 IAT 项都被修复了。

下面我们用 OllyDump 插件来进行 dump,接着打开 IMP REC。



我们可以看到获取的 IAT 项都是有效的,接下来我们修复 dump 文件。



接下来我们需要修复 TLS,我们用 OD 加载修复后的 dump 文件,定位到 PE 头。

0040014C	00000000	DD 00000000	Must be 0
00400150	10F10900	DD 0009F110	TLS Table address = 9F110
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
00400160	00000000	DD 00000000	Bound Import Table address = 0

将 TLS Table address 和 TLS Table size 这两个字段的值修改为零。

0040014C	00000000	DD 00000000	Must be 0
00400150	00000000	DD 00000000	TLS Table address = 0
00400154	00000000	DD 00000000	TLS Table size = 0
00400158	00000000	DD 00000000	Load Config Table address = 0
0040015C	00000000	DD 00000000	Load Config Table size = 0
00400160	00000000	DD 00000000	Bound Import Table address = 0

保存修改到文件。



直接双击运行,我们可以看到完美运行。

好了,本章到此结束。