

第五十七章-ExeCryptor v2.2.50.c/d/e/f/g 脱壳

UnPackMe C:

本章我们继续加强 ExeCryptor UnPackMe 的难度。UnPackMe C 与 UnPackMe B 的难度比较接近,只不过 UnPackMe C 在运行的时候会检测是否存在注册表以及文件监视工具,如果检测到了会将它们关闭。



由于这里我并没有开启注册表以及文件的监视工具,所以 UnPackMe C 与 UnPackMe B 的脱壳方法是一样的,我的机器上 MOV EAX,DWORD PTR SS:[EBP - C]这条指令的地址为 486DF7。

00486DF7 8B45 F4 MOV EAX,DWORD PTR SS:[EBP-C]

00486DFA E8 61670100 CALL 0049D560 ; UnPackMe.0049D560

00486DFF 5B POP EBX

00486E00 8B0424 MOV EAX,DWORD PTR SS:[ESP]

00486E03 52 PUSH EDX

我们将脚本中硬件执行断点的地址修改为 486DFA。

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 486DFA,"x"  
    mov [486DFA],0  
    mov [486DFB],0  
    cob ToRepair
```

run

ToRepair:

```
cmp eip,486DFA
jne ToSkip
log eax
mov [table],eax
run
```

ToSkip:

```
add table,4
jmp start
```

final:

```
ret
```

到达 OEP 处以后别忘了删除掉 break-on-execute 断点(PS:使用 OllyBone 插件设置了 break-on-execute 断点的话,记得要删除,没有用到 OllyBone 插件的话,就不用管了),接着将监控线程挂起,然后在 ZwTerminateProcess 这个 API 函数的入口处设置一个硬件执行断点,接着执行该脚本修复 IAT,IAT 修复完毕以后就可以进行 dump 了,然后打开 IMP REC 修复 dump 文件,这样 UnPackMe C 就搞定了。

UnPackMe D:

接下来我们来看看 UnPackMe D,双击运行,看看等级 D 的保护措施:



我们可以看到调试消息这个选项开启了,也就是说会检测调试消息。不知道对我们有没有影响,我们用 OD 加载 UnPackMe D,还是跟之前一样断在了系统断点处,我们删除掉断点列表窗口中一次性断点,接着给代码段设置 break-on-execute 断点,运行起来就可以到达 OEP 处了。

004271B0	55	PUSH EBP	
004271B1	8BEC	MOV EBP,ESP	
004271B3	6A FF	PUSH -1	
004271B5	68 600E4500	PUSH 450E60	
004271BA	68 C8924200	PUSH 4292C8	
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
004271C5	50	PUSH EAX	
004271C6	64:8925 000000	MOV DWORD PTR FS:[0],ESP	
004271CD	83C4 A8	ADD ESP,-58	
004271D0	53	PUSH EBX	
004271D1	56	PUSH ESI	
004271D2	57	PUSH EDI	
004271D3	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
004271D6	FF15 DC0A4600	CALL NEAR DWORD PTR DS:[460ADC]	UnPackMe.0048AF52
004271DC	33D2	XOR EDX,EDX	
004271DE	8AD4	MOV DL,AH	
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX	

到目前为止,我们还没有看到 Debug Messages 这个选项开启了对我们有什么实质上的影响。下面我们来查看一下线程的情况。

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
000001C4	7C810659	7FFAA000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
000002C4	7C810659	7FFDC000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000428	7C810659	7FFAC000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
000005D4	7C810659	7FFD5000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000608	7C810659	7FFD4000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
0000079C	004F7085	7FFDE000	ERROR_SUCCESS (00)	Active	32 + 0	0.0468 s	0.0937 s
000007A8	7C810659	7FFD9000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000814	7C810659	7FFAB000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000850	7C810659	7FFD6000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000858	7C810659	7FFAD000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000920	7C810659	7FFDB000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
0000097C	7C810659	7FFD8000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000DA4	7C810659	7FFAF000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000E78	7C810659	7FFDA000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000E7C	00270000	7FFD0000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s
00000EA8	7C810659	7FFAE000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s
00000F4C	7C810659	7FFD7000	ERROR_SUCCESS (00)	Active	32 - 15	0.0000 s	0.0000 s

跟之前的一样,我们还是将除了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。

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
000001C4	7C810659	7FFAA000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
000002C4	7C810659	7FFDC000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000428	7C810659	7FFAC000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
000005D4	7C810659	7FFD5000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000608	7C810659	7FFD4000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
0000079C	004F7085	7FFDE000	ERROR_SUCCESS (00)	Active	32 + 0	0.0468 s	0.0937 s
000007A8	7C810659	7FFD9000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000814	7C810659	7FFAB000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000850	7C810659	7FFD6000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000858	7C810659	7FFAD000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000920	7C810659	7FFDB000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
0000097C	7C810659	7FFD8000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000DA4	7C810659	7FFAF000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000E78	7C810659	7FFDA000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000E7C	00270000	7FFD0000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s
00000EA8	7C810659	7FFAE000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s
00000F4C	7C810659	7FFD7000	ERROR_SUCCESS (00)	Suspended	32 - 15	0.0000 s	0.0000 s

好,下面我们定位到 OEP 下方调用的第一个 API 函数指令处,对其 IAT 项设置内存写入断点,接着利用 OD 的 Trace into 进行自动跟踪。

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

```

004271B0 55          PUSH EBP
004271B1 8BEC       MOV EBP,ESP
004271B3 6A FF     PUSH -1
004271B5 68 600E4500 PUSH 450E60
004271B8 68 C8924200 PUSH 4292C8
004271BF 64:A1 00000000 MOV EAX,DWORD PTR FS:[0]
004271C5 50        PUSH EAX
004271C6 64:8925 00000000 MOV DWORD PTR FS:[0],ESP
004271CD 83C4 A8    ADD ESP,-58
004271D0 53        PUSH EBX
004271D1 56        PUSH ESI
004271D2 57        PUSH EDI
004271D3 8965 E8    MOV DWORD PTR SS:[EBP-18],ESP
004271D6 FF15 DC0A4600 CALL NEAR 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,EBX

```

A red arrow points to the instruction `CALL NEAR DWORD PTR DS:[460ADC]`. Below the assembly window, the breakpoint menu is open, showing options like "Memory, on access" and "Memory, on write".

自动跟踪需要一段时间。

004755DC	8905 DC0A4600	MOV DWORD PTR DS:[460ADC],EAX	kernel32.GetVersion
004755E2	8D05 62AF4800	LEA EAX,DWORD PTR DS:[48AF62]	
004755E8	C600 C3	MOV BYTE PTR DS:[EAX],0C3	
004755EB	E9 72590100	JMP 00480AF62	UnPackMe.00480AF62
004755F0	E8 0B000000	CALL 00475600	UnPackMe.00475600
004755F5	FF25 E00A4600	JMP NEAR DWORD PTR DS:[460AE0]	UnPackMe.004755F0

好了,自动跟踪结束了,断在了这里,我们在跟踪日志中定位到 MOV EAX,DWORD PTR SS:[EBP-C]这条语句,我们可以看到它在这里。

17.	Main	UnPackMe	0047557C	JMP 0046E71C	
16.	Main	UnPackMe	0046E912	RETN	ESP=0012FE00 FL=0
15.	Main	UnPackMe	004811F7	TEST AL,AL	
14.	Main	UnPackMe	004811F9	JNZ 0049AB1E	ECX=00000006, ESP=0012FE04
13.	Main	UnPackMe	0049AB1E	POP ECX	ECX=7C81120A, ESP=0012FE08
12.	Main	UnPackMe	0049AB1F	POP ECX	ESP=0012FE0C, EBP=0012FF38
11.	Main	UnPackMe	0049AB20	POP EBP	ESP=0012FE10
10.	Main	UnPackMe	0049AB21	RETN	EAX=7C8111DA ESP=0012FF38
9.	Main	UnPackMe	0046D8DC	MOV EAX,DWORD PTR SS:[EBP-C]	
8.	Main	UnPackMe	0046D8DF	MOV ESP,EBP	
7.	Main	UnPackMe	0046D8E1	JMP 00480B68	
6.	Main	UnPackMe	00480B68	JMP 004723F5	ESP=0012FF3C, EBP=0012FFC0
5.	Main	UnPackMe	004723F5	POP EBP	ESP=0012FF40
4.	Main	UnPackMe	004723F6	RETN	ESP=0012FF44
3.	Main	UnPackMe	00488247	RETN	
2.	Main	UnPackMe	00480B98	JMP 00493469	
1.	Main	UnPackMe	00493469	JMP 004755DC	
0.	Main	UnPackMe	004755DC	MOV DWORD PTR DS:[460ADC],EAX	

也就是说跟 UnPackMe C 并没有什么区别。

0046D8DC 8B45 F4 MOV EAX,DWORD PTR SS:[EBP-C] ; kernel32.GetVersion

0046D8DF 8BE5 MOV ESP,EBP

我们只需要将硬件执行断点的地址修改为 46D8DF 即可,修改后的脚本如下:

```

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 46D8DF,"x"

    mov [46D8DF],0

    mov [46D8DF],0

    cob ToRepair

    run

```

ToRepair:

run

ToSkip:

add table,4

```
jmp start
```

final:

ret

好了,现在我们重启 OD,再次断到 OEP 处,接着删除掉 break-on-execute 断点,然后在 ZwTerminateProcess 这个 API 函数的入口处设置一个硬件执行断点,接着执行该脚本。

Address	Hex	dump	ASCII
00460818	F0 68 DA 77 1B 76 DA 77 F4 EA DA 77 E7 EB DA 77	-k rw+vrw0 rw0	
00460828	83 78 DA 77 00 00 00 00 CF 65 C3 58 D8 03 C4 58	ax rw... deXi	
00460838	00 00 00 00 04 6A EF 77 66 95 EF 77 89 6A EF 77	...ej w0 w0	
00460848	F3 AD EF 77 ED 09 EF 77 99 8B EF 77 08 B5 EF 77	% wY w0 i wA	
00460858	2A 70 EF 77 B2 7C EF 77 73 F2 77 1E C9 F1 77	*j w0 wSwaf	
00460868	0C BC EF 77 52 04 EF 77 FA 80 EF 77 F1 00 EF 77	. wRE w i w i	
00460878	51 B2 EF 77 26 05 EF 77 2A E3 EF 77 5F 39 F2 77	Q w w0 w 9	
00460888	71 B4 EF 77 2E AD EF 77 E1 61 EF 77 B8 85 EF 77	q1 w i wpa w0a	
00460898	CC D2 EF 77 43 70 EF 77 FB EA F0 77 12 83 EF 77	lFE wCp w i w0a	
00460908	01 72 F0 77 A9 34 F0 77 05 93 EF 77 68 EF EF 77	0x w04 w0 w0 wh	
00460918	8A D2 EF 77 B2 6F EF 77 3F 38 F2 77 6D E8 EF 77	-E w0 w0 w8=wf	
00460928	68 E0 EF 77 00 60 EF 77 90 5B EF 77 6D AC EF 77	h0 w . wE l w m	
00460938	94 6C F0 77 22 80 EF 77 3D C8 F1 77 3D 6D F0 77	0 l w w l w w w m	
00460948	3F 0C EF 77 85 78 EF 77 26 09 EF 77 FB 5E EF 77	0 l w a c w w w l	
00460958	66 8A EF 77 FC 8A EF 77 0F 62 EF 77 49 5E EF 77	6 e w w w w b w l	
00460968	97 5D EF 77 1A 9A EF 77 6B FA EF 77 7B C9 F0 77	u j w w w w k w c	
00460978	DA 98 F2 77 1A 40 F2 77 55 EA EF 77 C5 61 EF 77	r y w w w w U w a	
00460988	70 E6 EF 77 F0 81 EF 77 2D 6C EF 77 98 6E EF 77	p p w w w w l w y n	
00460998	4F 83 EF 77 09 ED EF 77 EB AA EF 77 26 69 F0 77	0 a w w w w w l	
00460A08	B1 95 EF 77 6F 80 EF 77 8A 5A EF 77 E9 49 F2 77	w w w w w w w l	
00460A18	26 F1 F0 77 C9 D0 F0 77 51 E0 F0 77 33 8C EF 77	% w w w w w w w l	
00460A28	6C EF EF 77 29 94 EF 77 00 00 00 68 17 80 7C	l y w l w w w k	
00460A38	D4 A7 80 7C 51 0E 81 7C EE 1E 80 7C 1D 2F 81 7C	e 0 Q l Q u w i A C j	
00460A48	40 7A 94 7C 09 2A 81 7C DA CD 81 7C 16 1E 80 7C	0 z 0 l w i r u i A	
00460A58	15 99 80 7C F8 0E 81 7C B6 2B 81 7C E4 9A 80 7C	S 0 C l 0 w i A u i	
00460A68	51 9A 80 7C E3 14 82 7C BF 50 83 7C E8 8D 83 7C	Q U C l 0 b e i P a i b	
00460A78	AA CC 80 7C 62 2E 86 7C 7F DF 81 7C E7 4A 81 7C	l f P C l 0 a i w i P e	
00460A88	5B CF 81 7C 08 2F 81 7C 90 47 84 7C 0C 8A 83 7C	l b u i l 0 i 0 G a i	
00460A98	90 A4 80 7C CF BC 80 7C 62 D2 80 7C 62 15 81 7C	E K C l 0 i b e C l b s	
00460AA8	77 D0 80 7C 5E A3 80 7C 78 34 83 7C ED 09 92 7C	w S C l u C l x 4 a i	
00460AB8	FD 79 92 7C D4 05 92 7C 3D 04 92 7C A7 27 81 7C	z y e i e C l i 0 e i	
00460AC8	76 2E 81 7C 8B E2 85 7C A9 60 83 7C 45 1C 83 7C	v u i l 0 a i 0 a i E l	
00460AD8	77 0A 81 7C 3C 15 81 7C FE 4F 83 7C 54 5D 83 7C	w u i l 0 s u i 0 a i T i	
00460AE8	23 20 81 7C 72 67 83 7C 40 97 80 7C 27 09 83 7C	# u i r g a i 0 e C l	
00460AF8	C5 98 80 7C 05 10 91 7C B9 23 81 7C ED 10 91 7C	l 0 a i l 0 e i l 0 e i	
00460B08	B9 4C 83 7C 8A 18 92 7C 9F 2D 81 7C F1 9E 80 7C	l l a i l 0 e i l 0 e i	
00460B18	FC 38 81 7C A5 18 82 7C 44 20 83 7C BC 22 83 7C	* 8 u i l 0 e i l 0 e i	
00460B28	61 23 83 7C 47 9B 80 7C 41 26 81 7C 8E 0E 81 7C	a a i l G a C l A u i l	
00460B38	87 0D 81 7C 0E 18 80 7C 24 1A 80 7C F5 0D 80 7C	C u i l 0 a C l s C i s	
00460B48	FE DD 80 7C 01 BE 80 7C 0F AC 80 7C A0 F7 82 7C	m i C l 0 e C l s C i s	
00460B58	BB 08 83 7C A1 BA 80 7C EB 98 80 7C 15 A4 80 7C	l 0 a i l l C l 0 u C l	
00460B68	66 E8 80 7C EC E7 80 7C 01 9E 80 7C 79 9E 80 7C	f b C l y C l 0 e C l	
00460B78	74 0D 83 7C F8 98 80 7C D4 A0 80 7C B6 BD 80 7C	t a i l 0 e C l a C l	
00460B88	41 4D 83 7C 28 97 80 7C 19 FF 80 7C 82 FE 80 7C	A M a i l 0 C l i C l	
00460B98	CF B4 80 7C DA 11 81 7C C6 97 80 7C 19 B2 85 7C	C H C l i l 0 u C l	
00460BA8	E7 12 82 7C AB 1E 83 7C EE 86 82 7C 69 3F 87 7C	% e i l l a i l a e i l	
00460BB8	93 DC 81 7C 11 13 82 7C 8B B5 81 7C 39 0D 82 7C	# u i l l 0 i l i l i	

我们可以看到 IAT 项都被修复了,也就是说的确跟 UnPackMe C 没有区别。下面我们继续来看 UnPackMe E,看看有没有什么不同的地方。

UnPackMe E:

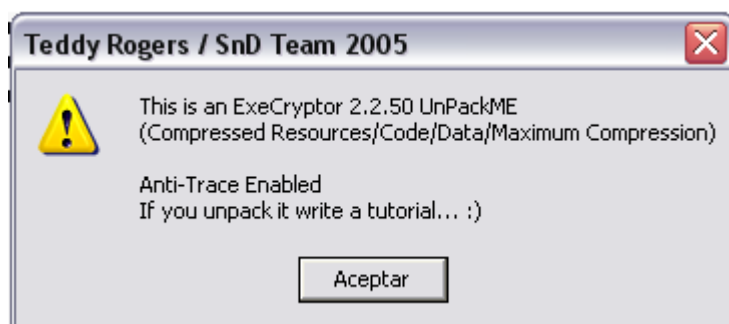


我们运行 UnPackMe E,看看跟 UnPackMe D 相比有什么区别。我们可以看到 UnPackMe E 的 Active Watch(这个单词我们可以理解为动态监视)这个选项开启了。

我们会发现脱 UnPackMe E,UnPackMe F 与脱 UnPackMe D 的步骤基本上是一样的,这里我不再赘述了。

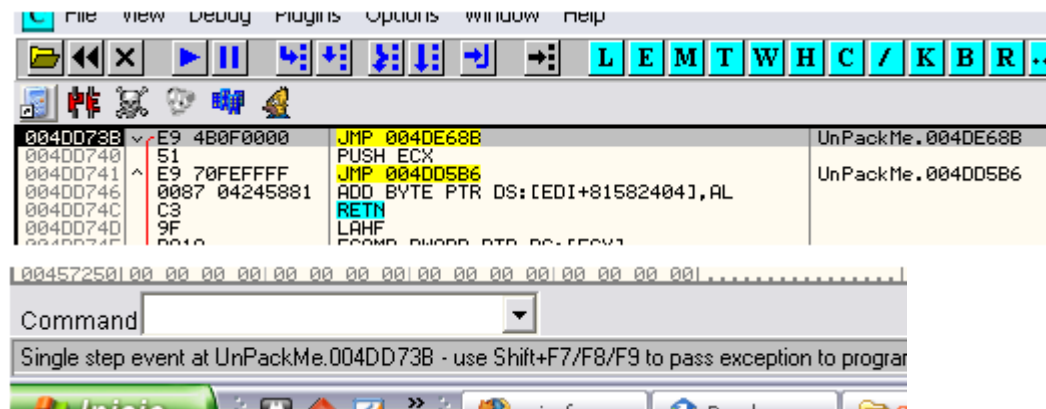
我们直接来看 UnPackMe G,UnPackMe G 的话就会玩一些新花样了。

UnPackMe G:



我们双击运行 UnPackMe G,可以看到这个等级反跟踪选项被开启了。我们需要利用 OD 的 trace into(自动跟踪)功能来定位 MOV EAX,DWORD PTR SS:[EBP-C]这条指令,不知道反跟踪这个选项会不会对此造成影响。

我们用 OD 加载 UnPackMe G,断在了系统断点处,接着我们删除掉断点窗口中的一次性断点,然后对代码段设置 break-on-execute 断点,接着运行起来,我们会发现在到达 OEP 之前,会断下来 5 到 6 次(由于单步异常导致的),这是反跟踪这个选项带来第一处影响。



我们可以看到 OD 状态栏中提示:发生了单步异常,需要我们手动按 Shift + F7/F8/F9 忽略掉这个异常继续往下执行。这里我们不能够勾选忽略单步异常这个选项,如果我们勾选了这个选项的话,那么 OllyBone 插件就不起作用了,所以这里我们必须手动按 Shift+F9 忽略这些单步异常,大约按 5 到 6 次 Shift +F9 就可能断到 OEP 处了。

4271B0	55	PUSH EBP	
4271B1	8BEC	MOV EBP,ESP	
4271B3	6A FF	PUSH -1	
4271B5	68 600E4500	PUSH 450E60	
4271B8	68 C8924200	PUSH 4292C8	
4271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
4271C5	50	PUSH EAX	
4271C6	64:8925 000000	MOV DWORD PTR FS:[0],ESP	
4271CD	83C4 A8	ADD ESP,-58	
4271D0	53	PUSH EBX	
4271D1	56	PUSH ESI	
4271D2	57	PUSH EDI	
4271D3	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
4271D6	FF15 DC0A4600	CALL NEAR DWORD PTR DS:[460ADC]	UnPackMe.0048D33F
4271DC	33D2	XOR EDX,EDX	
4271DE	8AD4	MOV DL,AH	
4271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX	
4271E6	8BC8	MOV ECX,EAX	
4271E8	81E1 FF000000	AND ECX,0FF	
4271EE	89D0 30E64500	MOV DWORD PTR DS:[45E630],ECX	
4271F4	91E1 80	CUI ECV 0	

另外一个细节就是,我们会发现除了该程序运行必需的两个线程,并不存在其他的监控线程了,这也算是反跟踪选项带了的又一个不同之处吧。

[Threads]							
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
00000F7C	004E02EA	7FFDD000	ERROR_SUCCESS (00)	Active	32 + 0	0.0312 s	0.0781 s
00000F80	00270000	7FFDC000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s

我们可以看到这里只有该程序运行必需的两个线程,如果还存在其他线程(这里我们姑且称这些线程为监控线程)的话,我们需要将这些监控线程挂起。这里由于反跟踪模式开启了,为了以防万一,我们将 OllyAdvanced 插件里面的 Anti-RDTSC 以及 GetTickCount 这两个选项勾选上。

Olly Advanced 1.26 Beta 12

Bugfixes Additional Options

Additional Options 2

Anti-Debug Anti-Debug 2

Debug Bits

☒ IsDebuggerPresent

☒ NIGlobalFlag (not recommended)

☒ HeapFlags

☒ ForceFlags

Anti-RDTSC (Driver Based)

☒ Enable

☒ Method 1 ☐ Method 2

Protected Application Environment

☒ SuspendThread

☒ BlockInput

Other

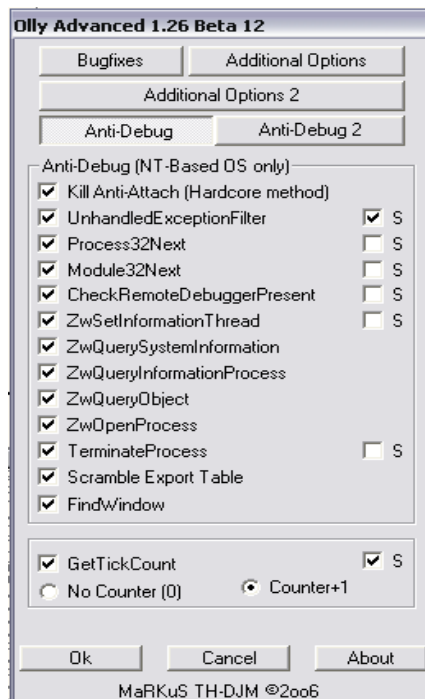
☒ Break on TLS Callback

☒ Anti-Anti Hardware BP

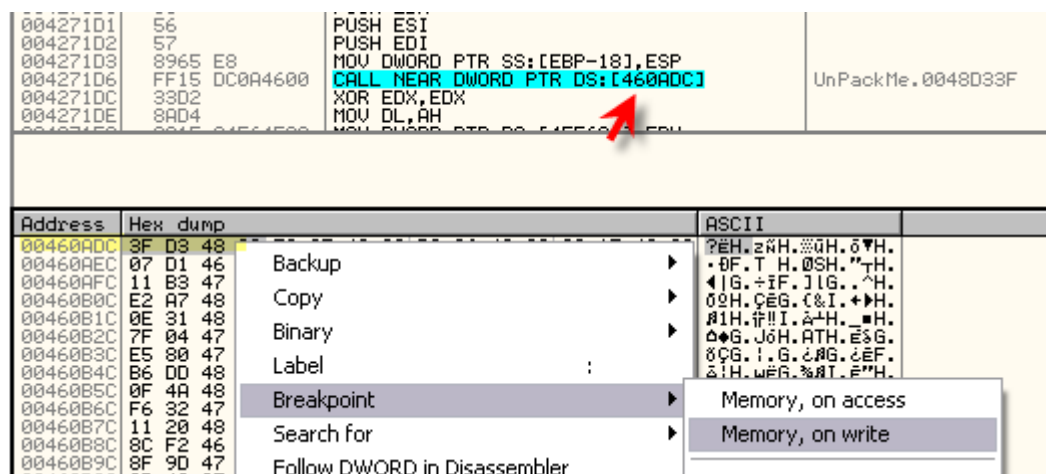
Ok Cancel About

MaRkuS TH-DJM ©2006

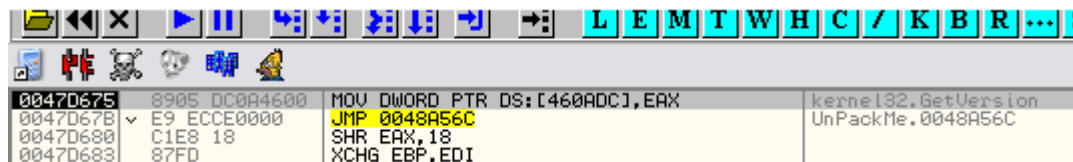
和



大家在使用 Anti-RDTSC 这个反反调试选项的时候,要稍微留意一下 OD 右下角显示的状态,如果显示为中断,过一会儿弹出一个消息框提示驱动无法正常启动的话,说明 Anti-RDTSC 这个选项附带的驱动程序无法正常工作,遇到情况的话,大家可以重启一下电脑试试看。



现在我们跟之前一样对 460ADC 这个 IAT 项设置内存写入断点,删除掉 break-on-execute 断点,接下来利用 OD 的 trace into 功能进行自动跟踪,看看会发生什么。



我们断到了这里。

下面我们在跟踪日志中定位 MOV EAX,DWORD PTR SS:[EBP-C]这条指令。

11.	Main	UnPackMe	00478F25	FUP EAX	EAX=7C81120H, ESP=0012FE08
10.	Main	UnPackMe	00478F26	POP EBP	ESP=0012FE0C, EBP=0012FF38
9.	Main	UnPackMe	00478F27	RETN	ESP=0012FE10
8.	Main	UnPackMe	00491E65	MOV EAX,DWORD PTR SS:[EBP-C]	EAX=7C81110A
7.	Main	UnPackMe	00491E68	MOV ESP,EBP	ESP=0012FF38
6.	Main	UnPackMe	00491E6A	PUSH 4842D4	ESP=0012FF34
5.	Main	UnPackMe	00491E6F	JMP 00476845	
4.	Main	UnPackMe	00476845	RETN	ESP=0012FF38
3.	Main	UnPackMe	004842D4	POP EBP	ESP=0012FF3C, EBP=0012FFC0
2.	Main	UnPackMe	004842D5	RETN	ESP=0012FF40
1.	Main	UnPackMe	00484BCD	RETN	ESP=0012FF44
0.	Main	UnPackMe	0047D675	MOV DWORD PTR DS:[460ADC],EAX	

这里我们可以看到 MOV EAX,DWORD PTR SS:[EBP-C]这条指令的地址为 491E65。

00491E65 8B45 F4 MOV EAX,DWORD PTR SS:[EBP-C] ; kernel32.GetVersion

00491E68 8BE5 MOV ESP,EBP

也就说到目前为止,trace into 并没有出现问题,有可能是我们勾选上了 OllyAdvanced 插件中的 Anti-RDTSC 以及 GetTickCount 这两个选项的缘故吧,我们还可以看到此时线程函数入口地址为 270000 的这个线程中止了。

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
00000F7C	004E02EA	7FFDD000	ERROR_SUCCESS (00000000)	Active	32 + 0	0.0937 s	8.6718 s
00000F80	00270000	7FFDC000	ERROR_SUCCESS (00000000)	Paused	32 + 0	0.0000 s	1.3593 s

好,我们需要的信息已经搜集完毕了,接下来我们先进行 dump,接着来修改脚本,然后执行该脚本修复 IAT。

OllyDump - UnPackMe_ExeCryptor2.2.50.g.exe

Start Address: 400000

Size: E1000

Dump

Entry Point: E02EA

-> Modify: 271B0

Get EIP as OEP

Cancel

Base of Code: 93000

Base of Data: 4B000

☒ Fix Raw Size & Offset of Dump Image

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	0004A000	00001000	0004A000	00001000	E0000020
.rdata	0000C000	0004B000	0000C000	0004B000	C0000040
.data	00009000	00057000	00009000	00057000	C0000040
u8ajdry	00003000	00060000	00003000	00060000	E0000060
.rsrc	00008000	00063000	00008000	00063000	40000040
txnbnds	00001000	0006B000	00001000	0006B000	C0000040
7vapk...	00027000	0006C000	00027000	0006C000	E0000020
i8kmfeug	0004E000	00093000	0004E000	00093000	E0000060

☐ Rebuild Import

☒ Method1 : Search JMP[API] | CALL[API] in memory image
☐ Method2 : Search DLL & API name string in dumped file

这里我们将脚本中硬件执行断点的地址替换掉。

```
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 491E68,"x"
mov [491E68],0
mov [491E68],0
cob ToRepair
run
```

ToRepair:

```
cmp eip,491E68
jne ToSkip
log eax
mov [table],eax
run
```

ToSkip:

```
add table,4
jmp start
```

final:

```
ret
```

好,现在我们在 ZwTerminateProcess 这个 API 函数的入口地址处设置一个硬件执行断点,接着执行该脚本,我们可以看到 IAT 项都被修复了。

start:

```
cmp table,460978  
ja final  
cmp [table],50000000  
ja ToSkip
```

我们将脚本修改成这个样子(只让其修复第一个 DLL 中导出函数对应的 IAT 项),我们执行该脚本,接下来看看日志信息。

```
74CC0000 module C:\WINDOWS\system32\ntedit9.dll  
74CC1000 Code size in header is 00010400, extending to size of section  
00401000 Sent virtual address to ollybone module for NX remove  
contentido: 00480E79  
tabla: 00460818  
7C91EAF0 Single step event at ntdll.7C91EAF0  
contentido: 00487713  
tabla: 0046081C  
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68  
eax: 77D504EA ! user32.MessageBoxA  
00491E68 Access violation when writing to [77D504EA]  
004DEFCD Single step event at UnPackMe.004DEFCD  
contentido: 0046C811  
tabla: 00460820  
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68  
eax: 77D504EA ! user32.MessageBoxA  
00491E68 Access violation when writing to [77D504EA]  
00491E68 Access violation when writing to [77D504EA]  
Debugged program was unable to process exception  
contentido: 0047983D  
tabla: 00460824  
0048DCAA Single step event at UnPackMe.0048DCAA  
contentido: 0048C308  
tabla: 00460828  
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68  
eax: 77D504EA ! user32.MessageBoxA  
00491E68 Access violation when writing to [77D504EA]  
00491E68 Access violation when writing to [77D504EA]  
Debugged program was unable to process exception  
contentido: 004833AB  
tabla: 00460974  
004833B1 Single step event at UnPackMe.004833B1  
contentido: 0046F805  
tabla: 00460978  
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68  
eax: 7C80A7D4 ! kernel32.GetLocalTime  
00491E68 Access violation when writing to [7C80A7D4]  
00491E68 Access violation when writing to [7C80A7D4]  
Debugged program was unable to process exception  
Command He ZwTerminateProcess
```

这里我们可以看到中间有几处单步异常(之前是没有的),好,我们重启 OD,断到 OEP 处,现在我不需要使用 OllyBone 插件了,所以我们可以将忽略单步异常的选项勾选上。

The screenshot shows the Immunity Debugger interface. The assembly window displays code for the `ZwTerminateProcess` function. The `Debugging options` dialog is open, with the `Exceptions` tab selected. The `Ignore memory access violations in KERNEL32` checkbox is checked. Under the `Ignore (pass to program) following exceptions:` section, the `Single-step break` checkbox is checked and highlighted by a red arrow. Other checked exceptions include `INT3 breaks`, `Memory access violation`, `Integer division by 0`, `Invalid or privileged instruction`, and `All FPU exceptions`. The `Ignore also following custom exceptions or range:` checkbox is also checked, and a list of custom exceptions is shown, including `000006EF` and `80000002 (DATATYPE MISALIGNMENT)`.

我们再次在 `ZwTerminateProcess` 的入口处设置一个硬件执行断点,执行该脚本,接下来我们来看看第一个 DLL 中的 IAT 项有没有被修复。

Address	Hex dump	ASCII
00460818	F0 68 DA 77 1B 76 DA 77 F4 EA DA 77 E7 EB DA 77	-k rw+vrwq0 rwb0 rw
00460828	83 78 DA 77 00 00 00 00 CF 65 C3 58 08 03 C4 58	ax rw...deix-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 09 EF 77 99 8B EF 77 C0 B5 EF 77	%i'wy'w0'i'w!a'w
00460858	2A 7D EF 77 B2 7C EF 77 77 53 F2 77 1E C9 F1 77	*j'w!!!'wWS=w!F!w
00460868	0C 0C EF 77 C2 04 EF 77 F0 8D FF 77 F1 0D FF 77	d'wRe'w' i'w+! 'w

我们再来查看一下日志信息。

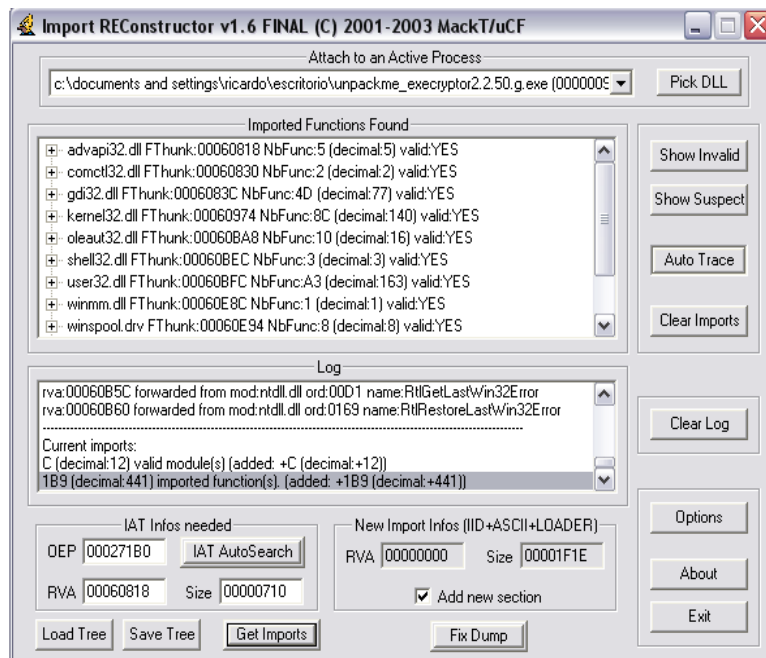
```

74CC1000 module C:\windows\system32\ole32.dll
00401000 Code size in header is 00010400, extending to size of 1
Sent virtual address to ollybone module for NX remove
contentido: 00480E79
tabla: 00460818
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68
eax: 77DA6BF0 ! ADVAPI32.RegCloseKey
00491E68 Access violation when writing to [77DA6BF0]
7C91E88E Hardware breakpoint 1 at ntdll.ZwTerminateProcess
contentido: 00487713
tabla: 0046081C
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68
eax: 77DA761B ! ADVAPI32.RegOpenKeyExA
00491E68 Access violation when writing to [77DA761B]
7C91E88E Hardware breakpoint 1 at ntdll.ZwTerminateProcess
contentido: 0046C811
tabla: 00460820
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68
eax: 77DAEAF4 ! ADVAPI32.RegCreateKeyExA
00491E68 Access violation when writing to [77DAEAF4]
7C91E88E Hardware breakpoint 1 at ntdll.ZwTerminateProcess
contentido: 0047983D
tabla: 00460824
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68
eax: 77DAEBE7 ! ADVAPI32.RegSetValueExA
00491E68 Access violation when writing to [77DAEBE7]
7C91E88E Hardware breakpoint 1 at ntdll.ZwTerminateProcess
contentido: 0048C308
tabla: 00460828
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68
eax: 77DA7883 ! ADVAPI32.RegQueryValueExA
00491E68 Access violation when writing to [77DA7883]
7C91E88E Hardware breakpoint 1 at ntdll.ZwTerminateProcess
contentido: 004833AB
tabla: 00460974
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68
eax: 7C80176B ! kernel32.GetSystemTime
00491E68 Access violation when writing to [7C80176B]
7C91E88E Hardware breakpoint 1 at ntdll.ZwTerminateProcess
contentido: 0046F805
tabla: 00460978
00491E68 Hardware breakpoint 2 at UnPackMe.00491E68
eax: 7C80A7D4 ! kernel32.GetLocalTime
00491E68 Access violation when writing to [7C80A7D4]
7C91E88E Hardware breakpoint 1 at ntdll.ZwTerminateProcess

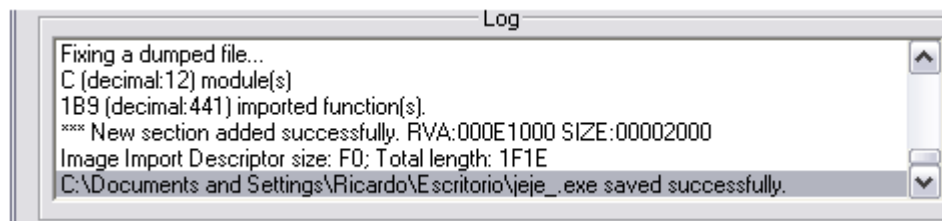
```

我们可以看到第一个 DLL 中的 IAT 项这次都被修复了,而且跟之前一样中途会跳转到 ZwTerminateProcess 处,好,现在我们再次将脚本修改回去,让其修复整个 IAT。

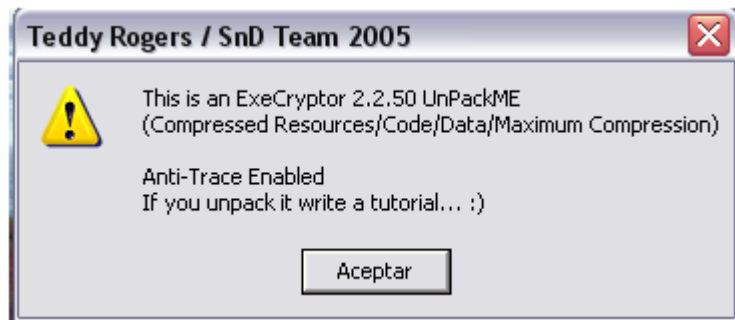
执行了该脚本以后,我们可以看到所有的项都被修复了。



我们修复 dump 文件。



运行修复后的 dump 文件。



好了,程序完美运行,本章到此结束。