

## 第四十章

本章在修复 PElLock 的 IAT,AntiDump 之前,我们先来讨论知识点,第一个知识点我们上一章已经碰到了,只不过没有展开讲而已,第二个知识点就是如何绕过 PElLock 或者类似软件对硬件断点的检测。

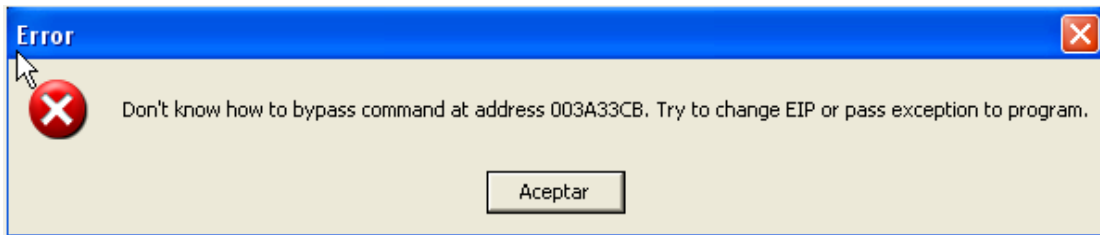
知识点 1 在上一章节我们遇到过,大家可能没有在意,大家是否还记得上一章节中,OD 加载目标程序以后,直接运行起来会弹出一个错误框,提示不知道该如何处理 xxx 处的非法指令。

为了解决这个问题,我们需要用到 Patched4 这个程序(又一个打过补丁的 OD),我们将其置于原 OD 所在的文件夹中,因为原版 OD 在处理非法指令的时候有个 BUG,就算我们勾选上调试选项中的忽略非法指令异常,当 OD 遇到非法指令的时候还是会弹出一个错误框,Patched4 这个 OD 修复了这个 BUG。

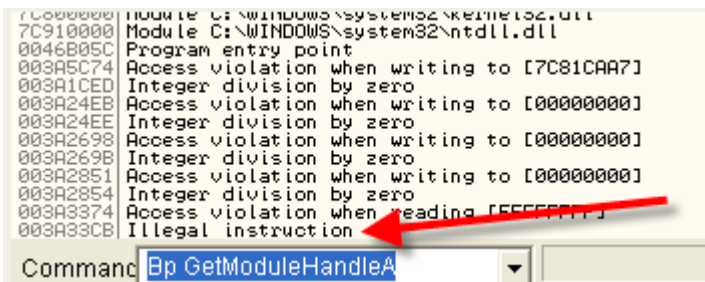
大家将 Patch4 这个 OD 跟原版 OD 放到同一个文件夹下面,下面我们分析 PElLock 的时候需要用到。

好,再来给大家解释一遍(PS:老外比较细心,也比较啰嗦,嘿嘿)。

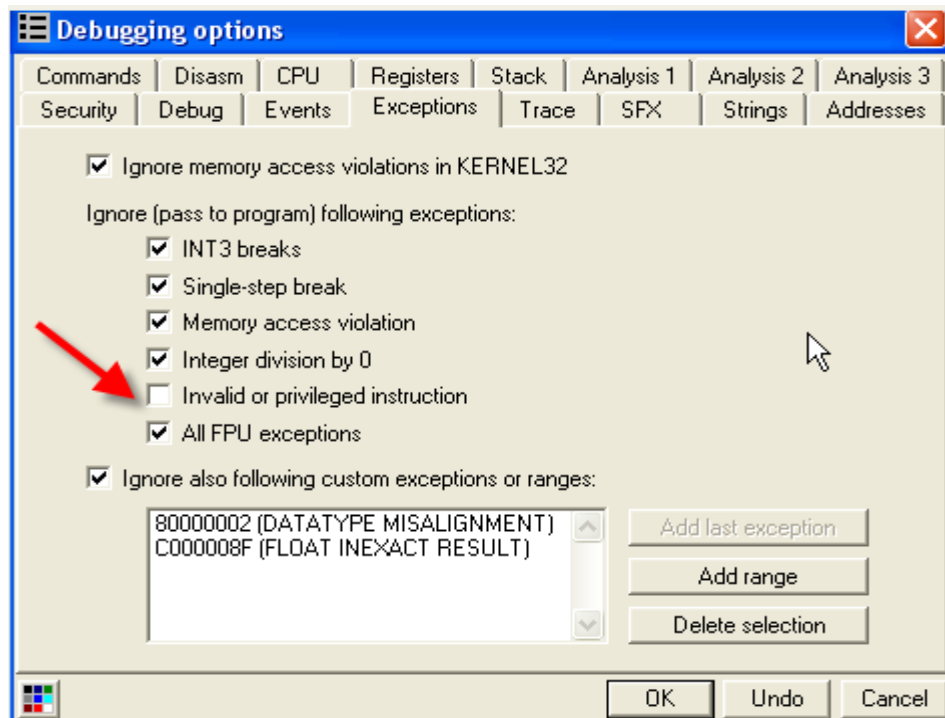
原版 OD 处理非法指令的时候有个 BUG,勾选上调试选项中的忽略非法指令异常,加载 UnPackMe\_PELock1.06,接着运行起来。



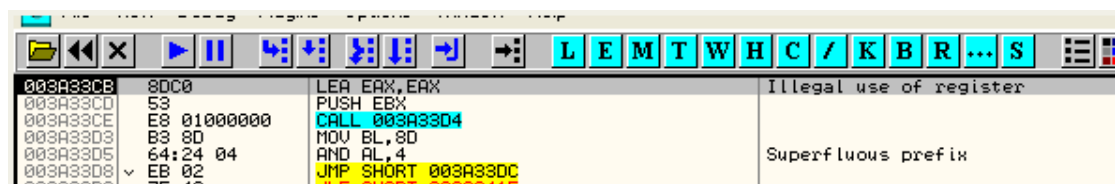
我们可以看到弹出了一个错误框,但是我们用 Patch4 这个 OD 来加载 UnPackMe\_PELock1.06 的话,就可以完美运行。我们来看看日志信息。



我们可以看到最后一次异常是一个非法指令异常,现在我们不勾选的忽略非法指令异常选项。



这里我们去掉 Invalid or privileged instruction 这个选项的对勾,加载 UnPackMe\_PELock1.06,运行起来。

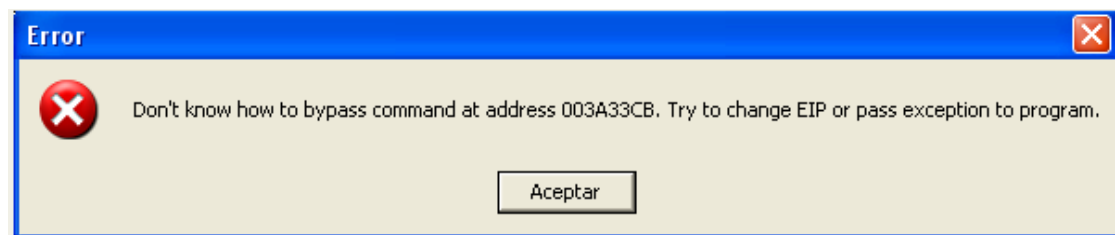


断在了异常处,如果我们按 Shift + F9 也可以继续运行下去。

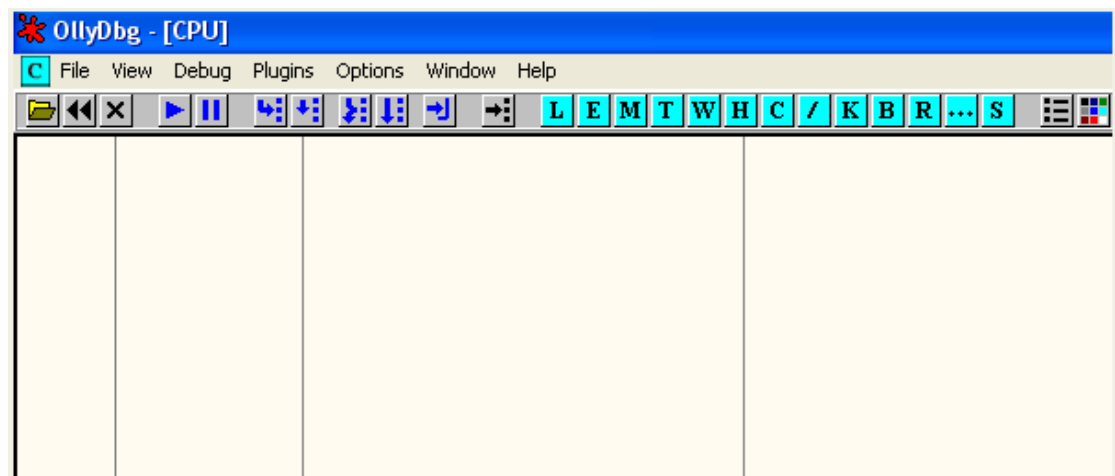
也就是原版 OD 是就算你勾选了忽略非法指令异常这一项,OD 并没有自动忽略。

如果是仅仅有一个这类异常,我们按一下 Shift + F9 也没什么。但是如果目标程序会产生 200 或者 300 个这样的异常,难道我们也手工去按 Shift + F9 不成(PS:连续按 Shift + F9 200 多次,那还不得疯掉哇,哈哈)。

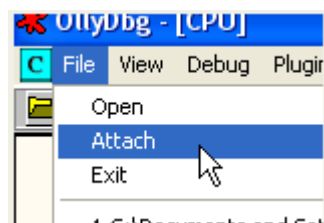
我们再次勾选上 Invalid or privileged instruction 这个选项,运行起来,再次弹出了错误框。

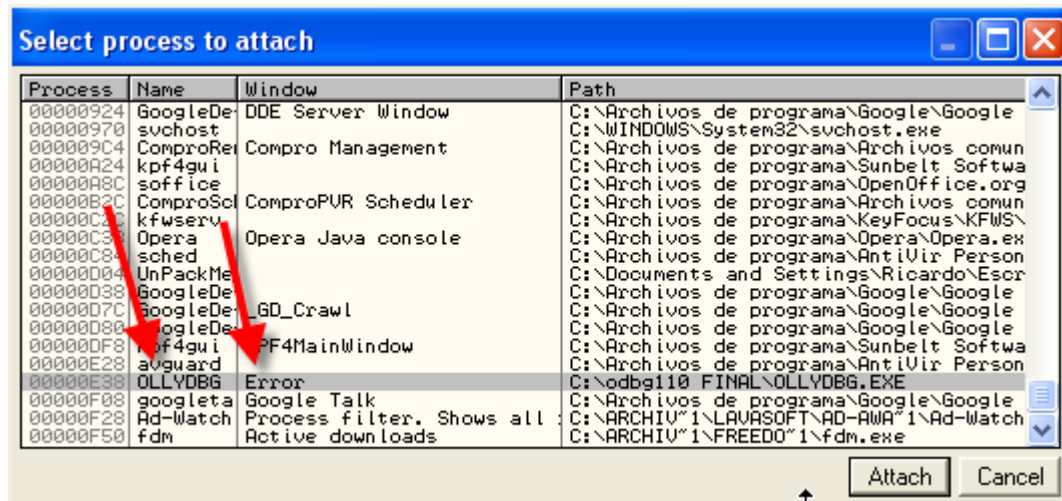


下面我们来尝试修复 OD 的这个 BUG,再新开一个 OD。

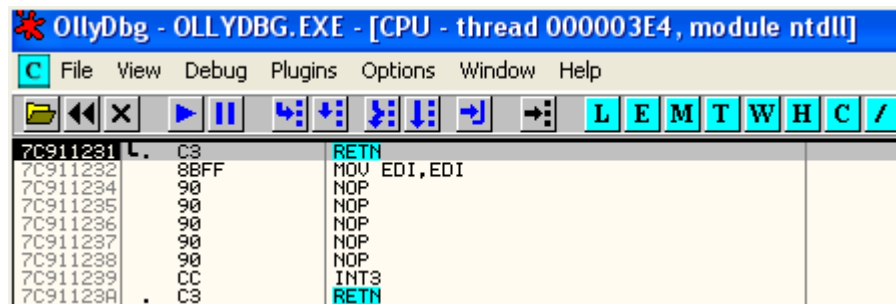


选择新的 OD 主菜单中的 File-Attach。定位到弹出错误框的 OD 所在的进程。





定位到弹出错误框 OD 所在的进程,单击 Attach(附加)。

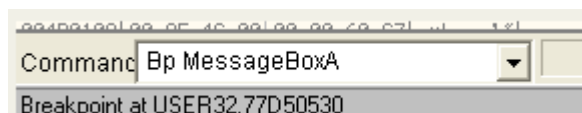


断在了这里,我们运行起来。

接着我们来看一下区段列表窗口。

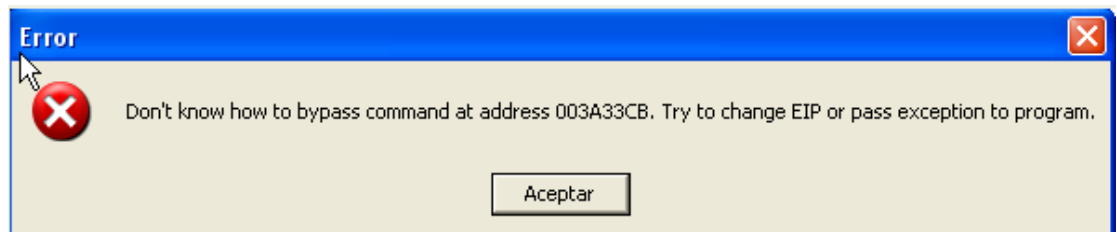


现在我们可以看到 OllyDbg 的各个区段情况,接下来我们给 MessageBoxA 这个 API 函数设置一个断点。



Address	Disassembly	Comment
77D5045A	8BFF	MOV EDI,EDI
77D5045C	55	PUSH EBP
77D5045D	8BEC	MOV EBP,ESP
77D5045E	833D BC04D777	CMP DWORD PTR DS:[77D704BC],0
77D5045F	74 24	JE SHORT 77D50510
77D50460	64:01 18000000	MOV EAX,DWORD PTR FS:[18]
77D50461	6A 00	PUSH 0
77D50462	FF70 24	PUSH DWORD PTR DS:[EAX+24]
77D50463	68 2400D777	PUSH 77D70B24
77D50464	FF15 C812D177	CALL DWORD PTR DS:[77D112C8]
77D50465	85C0	TEST EAX,EAX
77D50466	75 0A	JNZ SHORT 77D50510
77D50467	C705 200BD777	MOV DWORD PTR DS:[77D70B20],1
77D50468	6A 00	PUSH 0
77D50469	FF75 14	PUSH DWORD PTR SS:[EBP+14]
77D5046A	FF75 10	PUSH DWORD PTR SS:[EBP+10]
77D5046B	FF75 0C	PUSH DWORD PTR SS:[EBP+0C]
77D5046C	FF75 08	PUSH DWORD PTR SS:[EBP+08]
77D5046D	E8 2D000000	CALL 77D5055C
77D5046E	5D	POP EBP
77D5046F	C2 1000	RET 10
77D50470	90	NOP
77D50471	90	NOP

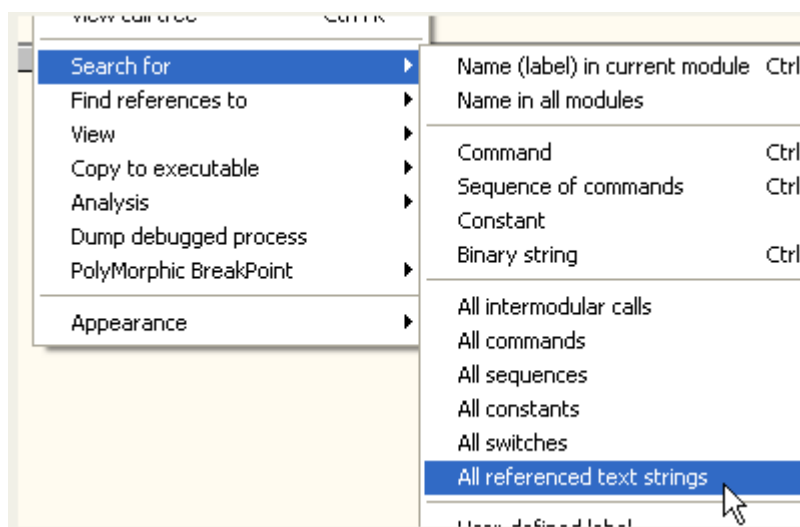
由于我们单击错误消息框上面的 OK 按钮,就到达 MessageBoxA 函数的 RET 指令处,我们给该 RET 指令处也设置一个断点。



弹出错误消息框后,我们单击 Aceptar(OK)按钮。就会断在 MessageBoxA 这个 API 函数的返回 RET 指令处,我们单击 F7 键单步。

Address	Disassembly	Comment
0045401H	90	NOP
0045401B	90	NOP
0045401C	55	PUSH EBP
0045401D	8BEC	MOV EBP,ESP
0045401F	81C4 FCF9FF	ADD ESP,-604
00454025	8D45 0C	LEA EAX,DWORD PTR SS:[EBP+C]
00454028	50	PUSH EAX
00454029	8B55 08	MOV EDX,DWORD PTR SS:[EBP+8]
0045402C	52	PUSH EDX
0045402D	8D0D FCF9FF	LEA ECX,DWORD PTR SS:[EBP-604]
00454033	51	PUSH ECX
00454034	E8 1B2C0500	CALL 004A6C54
00454039	83C4 0C	ADD ESP,0C
0045403C	8D85 FCF9FF	LEA EAX,DWORD PTR SS:[EBP-604]
00454042	8B15 7C3B4D00	MOV EDX,DWORD PTR DS:[4D3B7C]
00454048	68 10200000	PUSH 2010
0045404D	68 10B44B00	PUSH 4BB410
00454052	50	PUSH EAX
00454053	52	PUSH EDX
00454054	E8 BDB40500	CALL 004AF516
00454059	6A 00	PUSH 0
0045405B	E8 70AFAFFF	CALL 00431A08
00454060	59	POP ECX
00454061	8BE5	MOV ESP,EBP
00454063	5D	POP EBP

下面我们查看一下字符串列表,看看有没有消息框提示的错误信息字符串。



Address	Disassembly	Text string
00401174	MOV DWORD PTR DS:[4CD290],4B0134	ASCII "NULL if not ..."
00401250	MOV DWORD PTR DS:[4CD290],4B0144	ASCII "Too lo
004013A2	MOV EDX,4B0158	ASCII "RA"
004013E4	MOV EDX,4B0158	ASCII "RB"
00401484	MOV EDX,4B015E	ASCII "ST"
00401519	MOV DWORD PTR DS:[4CD290],4B0161	ASCII "FPU re
0040154C	MOV DWORD PTR DS:[4CD290],4B0183	ASCII "Closin
00401757	MOV EDX,4B01A0	ASCII "XMMWOR
004017AE	MOV EDX,4B01A8	ASCII "EIP"

我们看看有没有字符串里面含有 bypass 这个单词。

00435078	PUSH 4B6B95	ASCII "Don't know how to step command at address %08X. Try to run, change EIP or pass exception to program."
00435088	PUSH 4B6C54	ASCII "Don't know how to step command at address %08X. Try to change EIP or pass exception to program."
004351F7	MOV EDX,4B6D57	ASCII "reset actual breakpoint."
00435208	PUSH 4B6CB4	ASCII "Debugged program set single step flag (bit 7 in EFL). I don't know how to step command at address %08X. Try to change EIP or pass exception to program."
00435239	PUSH 4B6D71	ASCII "Don't know how to continue because memory at address %08X is not readable. Try to change EIP or pass exception to program."
00435263	PUSH 4B6DEC	ASCII "Don't know how to bypass breakpoint at address %08X. Try to delete breakpoint, change EIP or pass exception to program."
00435273	PUSH 4B6E64	ASCII "Don't know how to bypass command at address %08X. Try to change EIP or pass exception to program."
00435302	PUSH 4B6E03	ASCII "OpenProcessToken"
00435394	PUSH 4B6ED7	ASCII "LookupPrivilegeValueA"
004353A6	PUSH 4B6EDD	ASCII "AdjustTokenPrivileges"
004353E3	CALL 4B6E00	ASCII "GetProcAddress"

我们可以看到我们要找到字符串在这里,我们在这个字符串上面双击。

00435240	E9 27010000	JMP 00435374	OLLYDBG.00435374
00435240	837D 14 00	CMP DWORD PTR SS:[EBP+14],0	OLLYDBG.00435240
00435251	75 3A	JNZ SHORT 00435280	OLLYDBG.00435280
00435253	833D 30814D00	CMP DWORD PTR DS:[4D8130],0	OLLYDBG.00435253
0043525A	75 31	JNZ SHORT 00435280	OLLYDBG.00435280
0043525C	837D FC 00	CMP DWORD PTR SS:[EBP-4],0	OLLYDBG.0043525C
00435260	74 10	JE SHORT 00435272	OLLYDBG.00435272
00435262	53	PUSH EBX	Arg2
00435263	68 EC6D4B00	PUSH 4B6DEC	Arg1 = 004B6DEC ASCII "Don't
00435268	E8 AFED0100	CALL 0045401C	_Error
00435268	83C4 08	ADD ESP,8	OLLYDBG.00435268
00435270	EB 0E	JMP SHORT 00435280	OLLYDBG.00435280
00435272	53	PUSH EBX	Arg2
00435273	68 646E4B00	PUSH 4B6E64	Arg1 = 004B6E64 ASCII "Don't
00435278	E8 9FED0100	CALL 0045401C	_Error
0043527D	83C4 08	ADD ESP,8	OLLYDBG.0041B5A4
00435280	E8 1F63FEFF	CALL 0041B5A4	OLLYDBG.00435374
00435285	83C8 FF	OR EAX,FFFFFFFF	
00435288	E9 E7000000	JMP 00435374	
0043528D	8B55 DC	MOV EDX,DWORD PTR SS:[EBP-24]	
00435290	F682 1D030000	TEST BYTE PTR DS:[EDX+31D],1	
00435290	75 48	JNZ SHORT 004352E1	

我们可以看到这里有几处 JNZ 可以跳过弹出错误消息框的代码,这里我们将 435260 处的 JE 修改为 JMP 43528D,保存到文件中。

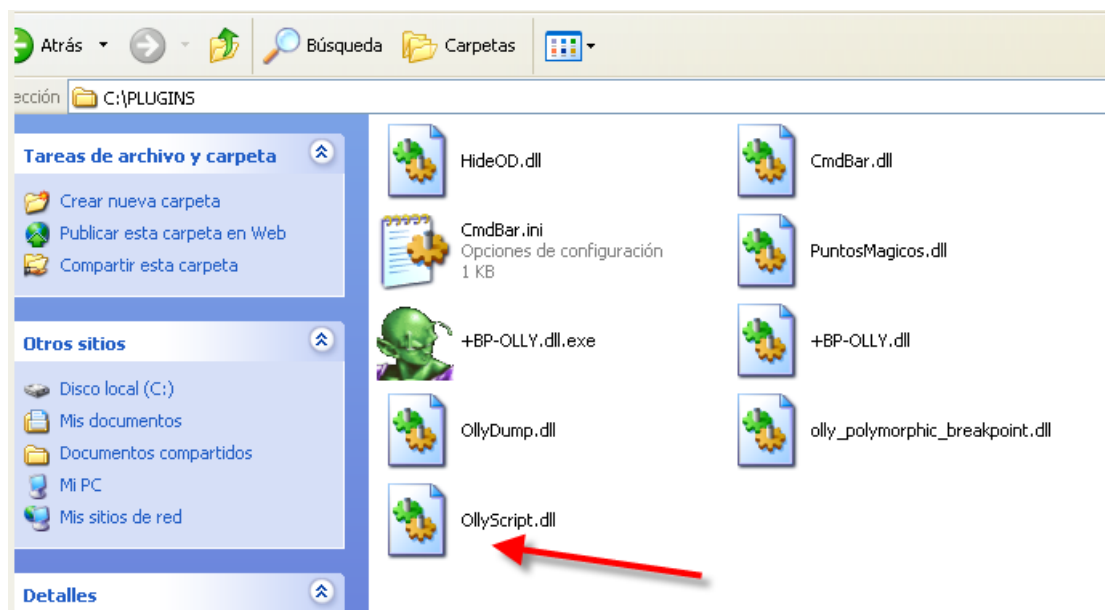
接下来我们看看 Patched4 这个 OD 该处的代码是怎么样的。

00435240	E8 5F63FEFF	CALL 0041B5A4	Parchead.0041B5A4
00435245	83C8 FF	OR EAX,FFFFFFFF	Parchead.00435374
00435248	E9 27010000	JMP 00435374	Parchead.00435248
0043524D	837D 14 00	CMP DWORD PTR SS:[EBP+14],0	Parchead.0043524D
00435251	75 3A	JNZ SHORT 00435280	Parchead.00435251
00435253	833D 30814D00	CMP DWORD PTR DS:[4D8130],0	Parchead.00435253
0043525A	75 31	JNZ SHORT 00435280	Parchead.0043525A
0043525C	837D FC 00	CMP DWORD PTR SS:[EBP-4],0	Parchead.0043525C
00435260	EB 2B	JMP SHORT 00435280	Parchead.00435260
00435262	53	PUSH EBX	ASCII "Don't know how to bypass b
00435263	68 EC6D4B00	PUSH 4B6DEC	Parchead._Error
00435268	E8 AFED0100	CALL 0045401C	ASCII "Don't know how to bypass c
0043526D	83C4 08	ADD ESP,8	Parchead._Error
00435270	EB 0E	JMP SHORT 00435280	Parchead.00435280
00435272	53	PUSH EBX	ASCII "Don't know how to bypass c
00435273	68 646E4B00	PUSH 4B6E64	Parchead._Error
00435278	E8 9FED0100	CALL 0045401C	Parchead.0041B5A4
0043527D	83C4 08	ADD ESP,8	Parchead.00435374
00435280	E8 1F63FEFF	CALL 0041B5A4	Parchead.004352E1
00435285	83C8 FF	OR EAX,FFFFFFFF	
00435288	E9 E7000000	JMP 00435374	
0043528D	8B55 DC	MOV EDX,DWORD PTR SS:[EBP-24]	
00435290	F682 1D030000	TEST BYTE PTR DS:[EDX+31D],1	
00435297	75 48	JNZ SHORT 004352E1	

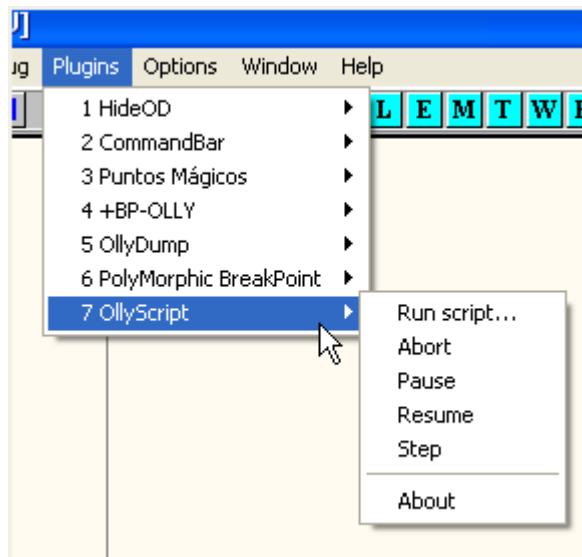
这里我们可以看到 435260 处的条件跳转被修改为无条件跳转,即修改为了 JMP 43528D,这样就能跳过填出错误消息框的代码。

这就是 Patched4 修复非法指令这个 BUG 的原理。大家可以自行给 OD 打补丁也可以直接用 Patched4 这个 OD。

知识点 2 就是要给大家演示如何编写脚本,我们需要用到 OllyScript0.92.dll 这个插件,将其放到 OD 的插件目录下。



打开 Patched4 这个 OD。

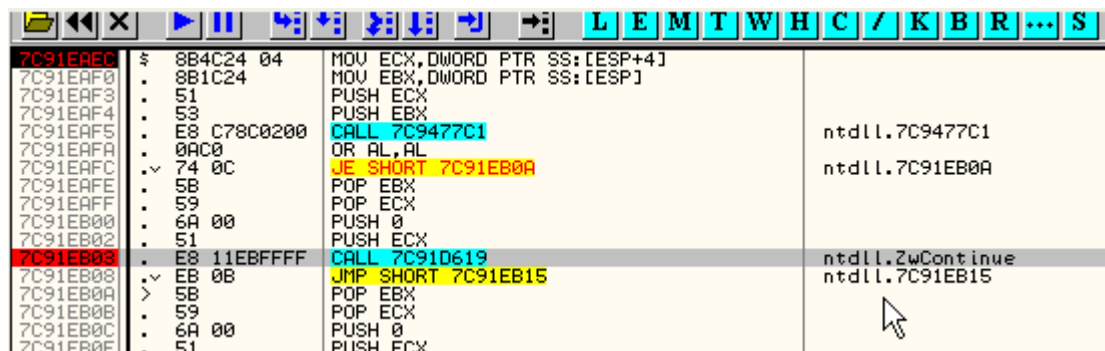


我们可以看到主菜单中出现 OllyScript 的插件子菜单项。

现在我们加载 UnPackMe\_PELock1.06,下面我们编写一个简单的脚本让这个壳无法检测到我们设置的硬件断点。

关于这个插件的使用说明大家可以参考插件文件夹下面的 readme.txt。

这个脚本的思路很简单:每次异常触发后都会断在 KiUserExceptionDispatcher,接下来就会由操作系统去调用异常处理函数,这个时候我们设置硬件断点可以被检测到,所以这个脚本要做的就是当前断在 KiUserExceptionDispatcher时,清除掉硬件断点,当断在下面的 CALL ZwContinue 时再将硬件断点恢复。



由于这是我们要编写的第一个脚本,所以不会让大家写太难的。首先给 KiUserExceptionDispatcher 和 ZwContinue 分别设置断点。现在我们开始编写脚本。一般来说脚本都有一个开始标签:

beginning:

设置这个标签可以让我们方便的 JMP beginning,接下来我们来设置硬件断点,举个例子:

bphws 12ffc4,“r”

bphws <BreakPoint HardWare Set> - 该命令可以对指定地址设置硬件断点,这里是对 12ffc4 设置硬件断点,“r”表示读取,”w”表示写入,”x”表示执行。

beginning:

bphws 12ffc4,“r”

work:

设置完硬件断点以后我们再来添加一个 work 标签。

beginning:

bphws 12ffc4,“r”

work:

eob to\_process

run

这里的 eob,<Execution On BreakPoint>-表示在下次中断发生时,跳转到指定标签处,这里该标签是 to\_process,即当程序中断下来时将跳转到 to\_process 标签处,接下来 run 命令表示运行起来。

这里就是关键的部分了,当前 OD 中断后,我们就需要检查是否断在了 KiUserExceptionDispatcher 入口处,所以我们在 to\_process 标签下面来进行检查。

to\_process:

cmp eip,7C91EAEC

je to\_clear

cmp eip,7C91EB03

je to\_recover

jmp to\_final

我这里 7C91EAEC 是 KiUserExceptionDispatcher 的入口地址,当断在 KiUserExceptionDispatcher 入口处时,eip 为 7C91EAEC,所以跳转到 to\_clear 标签处清除硬件断点,接下来当断在 CALL ZwContinue 指令处时,就跳转到 to\_recover 标签处恢复硬件断点。

清除内存断点如下:

to\_clear:

bphwc 12ffc4

jmp work

当前跳转到 to\_clear 标签以后,通过 bphwc <BreakPoint HardWare Clear>命令删除 12ffc4 处的硬件断点,接着跳转到 work 标签处继续运行程序。

另外一个 to\_recover 标签处是当断在 CALL ZwContinue 处时重新设置硬件断点的。

to\_recover:

jmp beginning

接下来就是直接跳转到开头 beginning 标签处了,也就是将重新设置硬件断点。

接下来就是程序断在硬件断点处时,也就是 jmp final。

to\_process:

cmp eip,7C91EAEC

je to\_clear

cmp eip,7C91EB03

je to\_recover

```
jmp final
```

最后是:

final:

```
MSGYN “是否继续?”
```

```
cmp $RESULT,1
```

```
je beginning
```

```
ret
```

这里的 MSGYN <Message Yes or No>命令会将指定消息,显示到一个对话框中,这个对话框上面有”是”,”否”按钮。如果点击”是”,保留变量\$RESULT 等于 1,否则保留变量\$RESULT 等于 0。如果我们想继续的话,就单击”是”,这样就会跳转 beginning 标签处继续执行脚本,如果我们不想继续,就单击”否”,这样就会直接 ret,脚本就执行完毕了,OD 就会停在硬件断点处。

整个脚本就是这样的:

beginning:

```
bphws 12ffc4,“r”
```

work:

```
eob to_process
```

```
run
```

to\_process:

```
cmp eip,7C91EAEC
```

```
je to_clear
```

```
cmp eip,7C91EB03
```

```
je to_recover
```

```
jmp to_final
```

to\_clear:

```
bphwc 12ffc4
```

```
jmp work
```

to\_recover:

```
jmp beginning
```

final:

```
MSGYN “是否继续?”
```

```
cmp $RESULT,1
```

```
je beginning
```

```
ret
```

该脚本的整个执行流程,大家可以参考下图中的解释,这里我就不再赘述了。

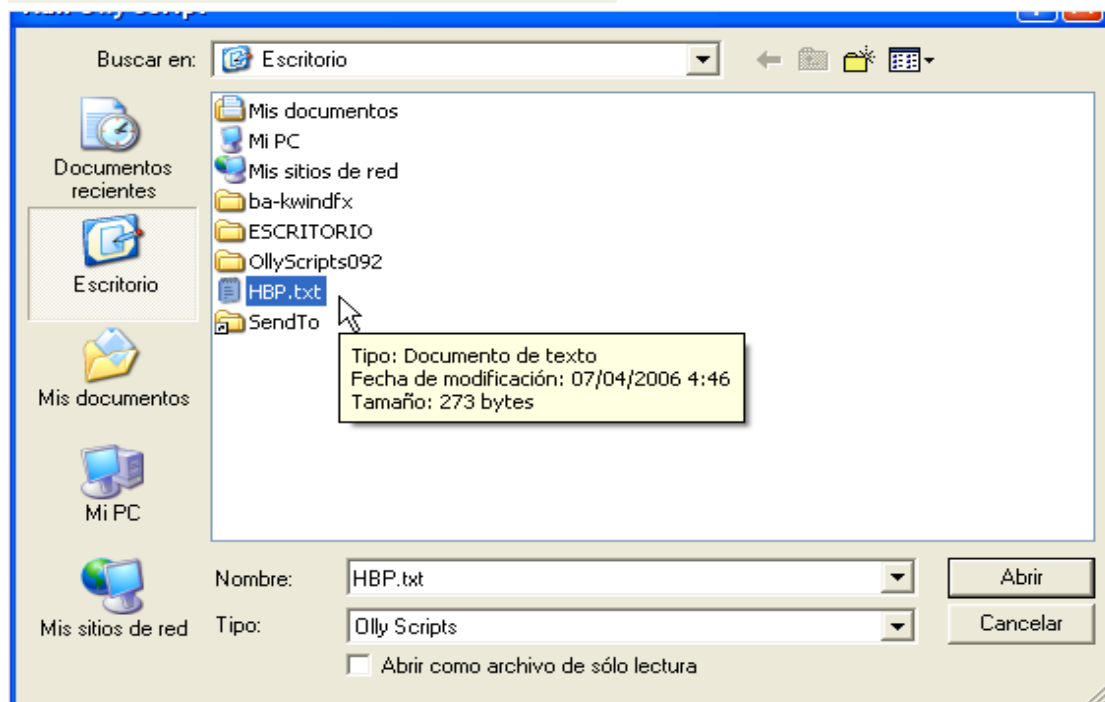
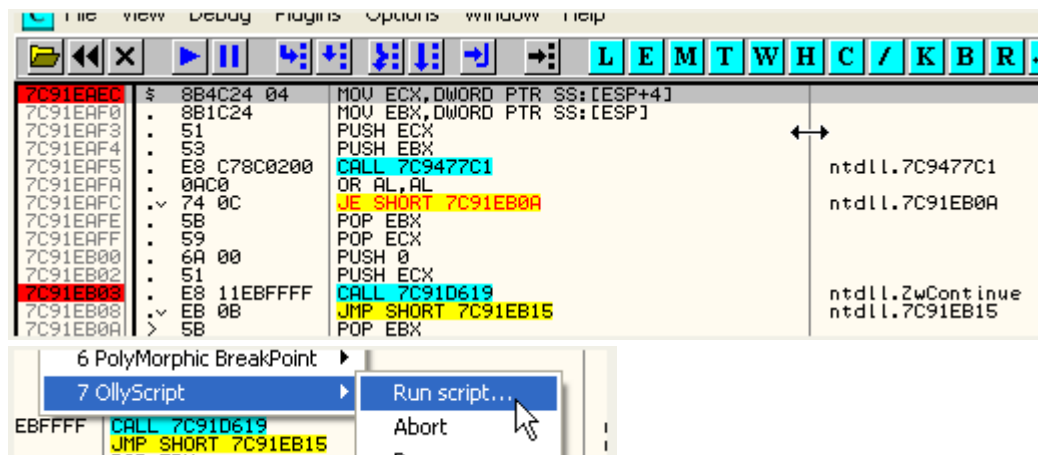


```

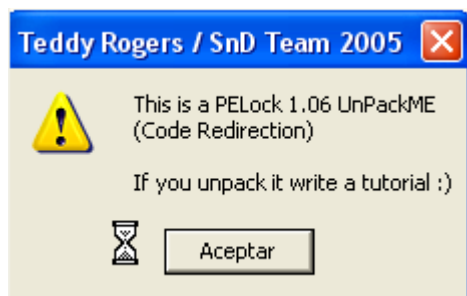
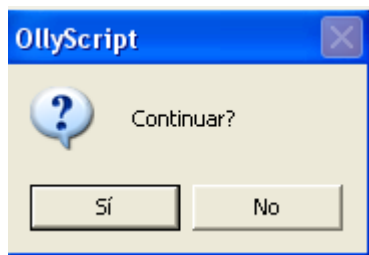
0000 beginning:
0001 bphws 12ffc4,"i"
0002 work:
0003 eob to_process //当发生中断的时候,就会跳转到to_process标签处
0004 run
0005
0006 to_process:
0007 cmp eip,7C91EAE0
0008 je to_clear //当断在KiUserExceptionDispatcher入口处时,跳转到to_clear标签处
0009 cmp eip,7C91EB03
0010 je to_recover //当断在call ZwContinue处时,就跳转到to_recover标签处
0011 jmp final //当断在硬件断点处时,就跳转到final标签处
0012
0013 to_clear:
0014 bphwc 12ffc4 //
0015 jmp work //首先清除硬件断点
0016 //接着跳转work标签处继续往下执行,等待下一次中断
0017
0018 to_recover:
0019 jmp beginning //
0020 //之前跳转到beginning标签处重新设置硬件断点
0021
0022 final:
0023 MSGVN "是否继续?"
0024 cmp $RESULT,1 //判断是否单击的是"Yes"按钮,如果是继续执行脚本,否则,直接返回,控制权交予OD,
0025 je beginning //此时OD断在硬件断点处
0026 ret

```

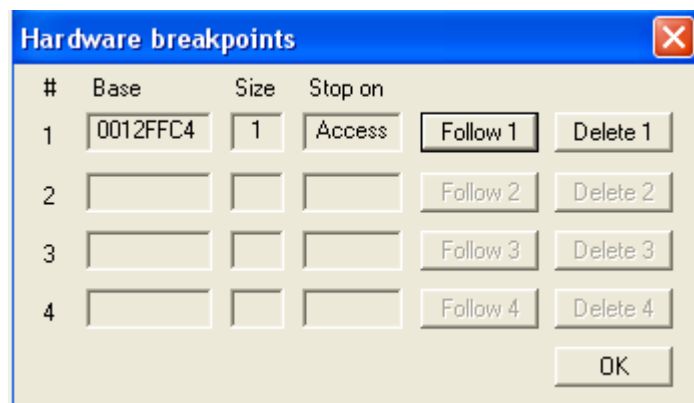
我们将上面的脚本保存到一个txt文本中,接着用OD加载UnPackMe\_PELock1.06,然后给KiUserExceptionDispatcher入口处和call ZwContinue指令处分别设置一个断点,接着在主菜单中找到OllyScript插件。



这里我们看到断在了硬件断点处,如果要继续执行脚本的话,单击 YES 按钮。



这里我们可以看到完美运行,我们来看看硬件断点窗口。



这里我们可以看到硬件断点设置成功了,也可以正常断下来,达到了我们预期的效果。

本章到此结束,下一章我们来介绍如何编写脚本修复 PELock 的 IAT。