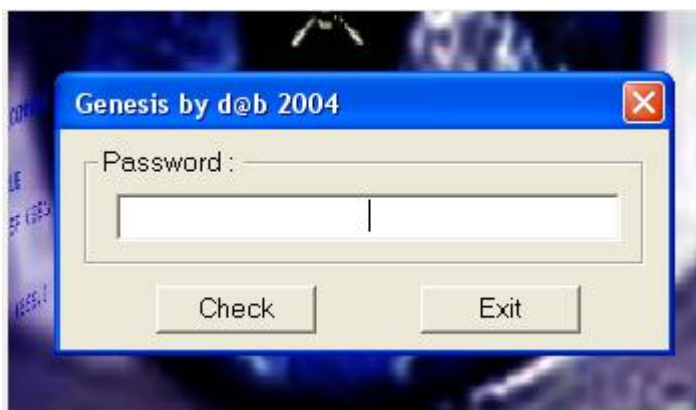


第四十五章-ReCrypt v0.80 脱壳

本章我们一起来分析一个用 OllyDbg 比较难脱的壳,这款壳就算我们配置好隐藏插件程序也无法正常运行,会弹出一个错误框。

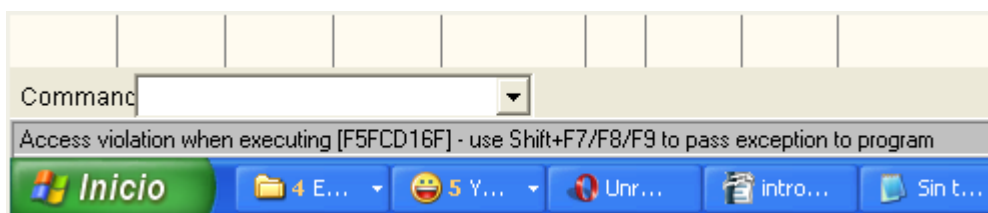
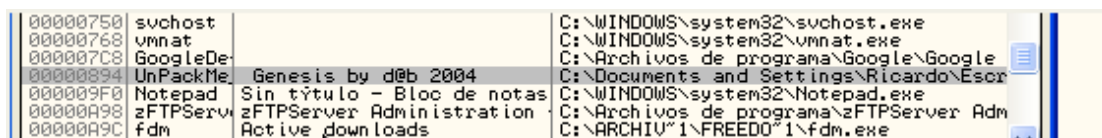
(PS:其实用海风月影大哥的 StrongOD 插件就可以完美解决,但是,04 年的时候 StrongOD 还没有出来,啧啧!!!)。其实用别的调试器也可以搞定,但是我们整个教程的核心是 OllyDbg,所以这里我们还是用 OllyDbg 来进行调试,嘿嘿。

我们的实验程序名称叫做 UnPackMe_ReCrypt0.80。这个程序的反调试比较强力(PS:当时算比较强力,现在不算什么了)。我们首先不加载调试器直接运行该程序。



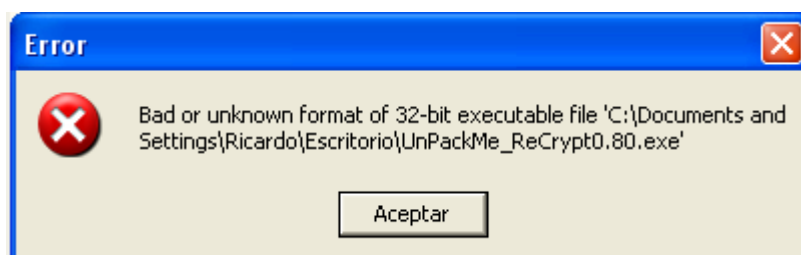
好,程序运行起来了,但是大家细心的话会发现一个比较蛋疼的问题,这个程序的 CPU 占用率居然高达 99%。我们一起来看看它做了哪些手脚。

我们打开 OllyDbg,选择菜单项 File -> Attach,打开可以附加的进程列表。

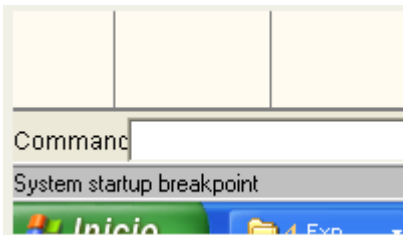


我们可以看到无法正常加载。

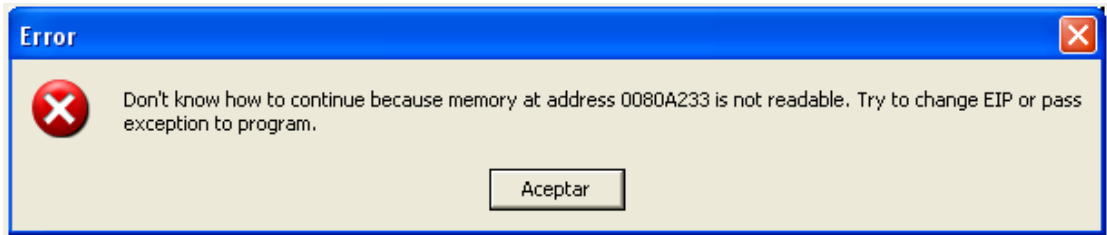
好,那么我们就用前面章节介绍过的专门用于定位 OEP 的 OllyDbg 或者 Patched 5 这个 OllyDbg 来打开它。



我们可以看到弹出了一个错误框提示无效的 PE 文件,这里我们单击 Aceptar(西班牙语译为:接受,这里译为确定)按钮。



断在了系统断点处,我们直接运行起来。

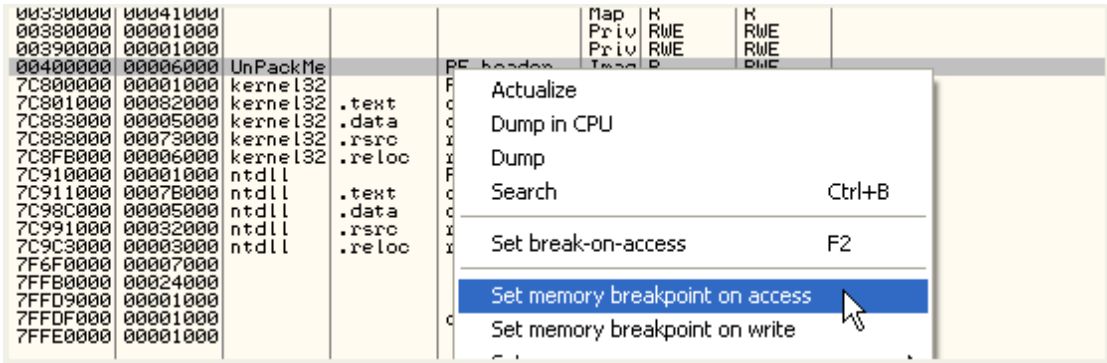


又弹出了一个错误框。

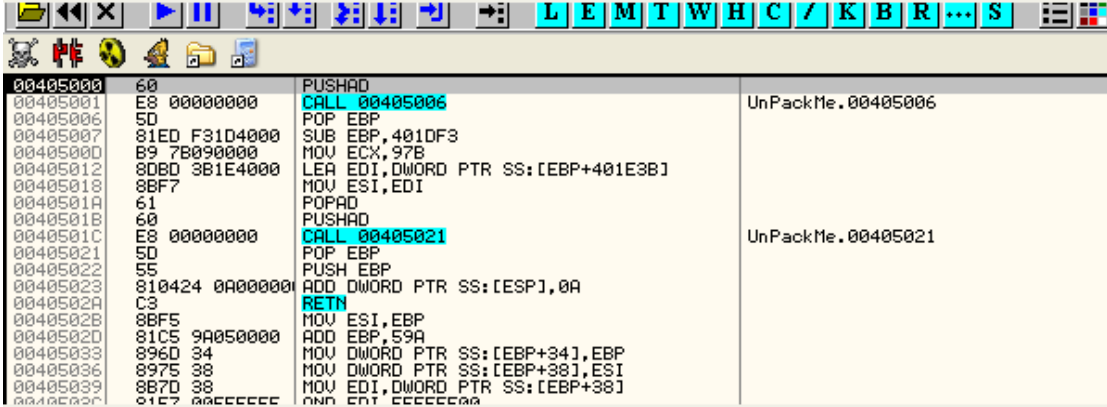
好,那么我们重启 OD,再次断在了 Sytem startup breakpoint(系统断点)处,现在我们单击工具栏上的 M 按钮打开区段列表窗口。

00380000	00001000								
00390000	00001000								
00400000	00006000	UnPackMe		PE header					
7C800000	00001000	kernel32		PE header					
7C801000	00082000	kernel32	.text	code,import					
7C883000	00005000	kernel32	.data	data					
7C888000	00073000	kernel32	.rsrc	resources					
7C8FB000	00006000	kernel32	.reloc	relocations					

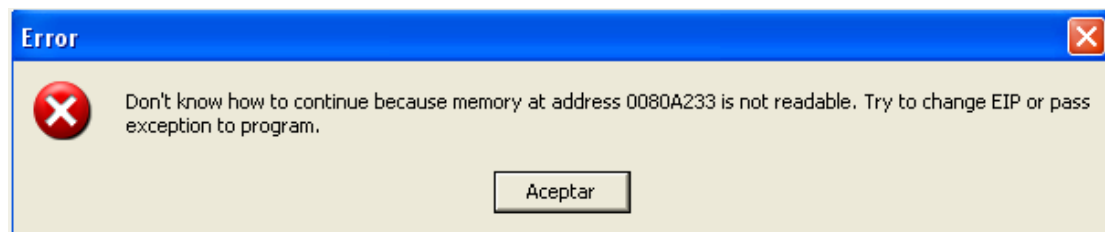
这里我们可以看到主模块仅仅只有一个区段,所以刚刚加载的时候弹出的那个无效 PE 格式的错误框是由于修改了原程序 PE 头中 NumberOfRvaAndSizes(数据目录结构数组的项数)字段导致的。接下来我们对该区段设置内存访问断点,大家应该还记得 Patched 5 这款 OD 吧,前面章节我们介绍过,这款 OD 是打过补丁的,其内存访问断点仅仅在执行代码的时候才会断下来,读取或者写入并不会断下来,这样对于我们定位 OEP 非常有帮助。



运行起来。

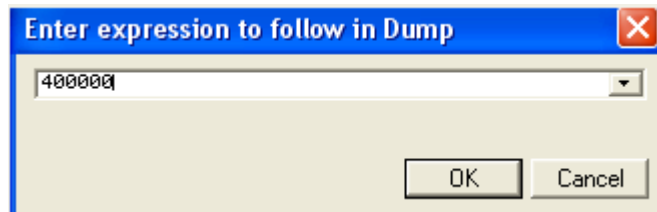


断在了 405000 地址处,好,现在我们删除内存访问断点,接着直接运行起来。

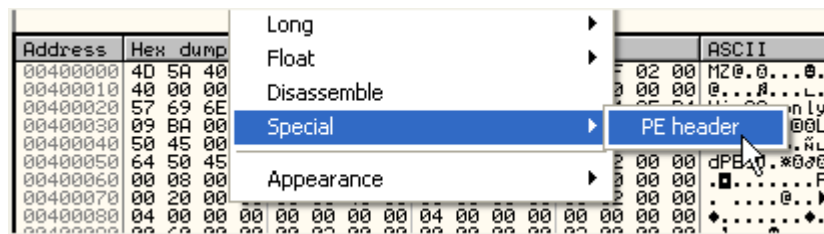


我们可以看到还是弹出了错误框,好,那么我们来查看 PE 头中 NumberOfRvaAndSizes(数据目录结构数组的项数)字段的值是多少。

我们在数据窗口中单击鼠标右键选择 Go to Expression,输入 400000(主模块基地址)。



接着还是单击鼠标右键选择 Special-PE header 切换到 PE 结构解析模式。



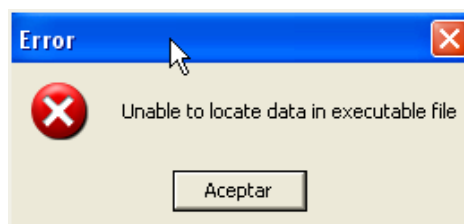
往下拉,定位到 NumberOfRvaAndSizes 字段。

Address	Hex dump	Data	Comment
0040008A	0000	DW 0000	MinorSubsystemVersion = 0
0040008C	00000000	DD 00000000	Reserved
00400090	00600000	DD 00006000	SizeOfImage = 6000 (24576.)
00400094	00020000	DD 00002000	SizeOfHeaders = 200 (512.)
00400098	00000000	DD 00000000	Checksum = 0
0040009C	0200	DW 0002	Subsystem = IMAGE_SUBSYSTEM_WINDOWS_GUI
0040009E	0000	DW 0000	DLLCharacteristics = 0
004000A0	00001000	DD 00001000	SizeOfStackReserve = 10000 (40960.)
004000A4	00100000	DD 00001000	SizeOfStackCommit = 1000 (4096.)
004000A8	00001000	DD 00001000	SizeOfHeapReserve = 10000 (40960.)
004000AC	00100000	DD 00001000	SizeOfHeapCommit = 1000 (4096.)
004000B0	EE92BF59	DD 59BF92EE	LoaderFlags = 59BF92EE
004000B4	095F0B7A	DD 7A0B5FD9	NumberOfRvaAndSizes = 7A0B5FD9 (2047565785.)
004000B8	00000000	DD 00000000	Export Table address = 0
004000BC	00000000	DD 00000000	Export Table size = 0
004000C0	C9560000	DD 000056C9	Import Table address = 56C9

我们可以看到该字段的值是 7A0B5FD9(PS:数据目录结构数组项数怎么可能会这么大,明显被做了手脚),通常情况下该字段的值为 0x10,好,这里我们把字段的值修改为 0x10,看看会发生什么。

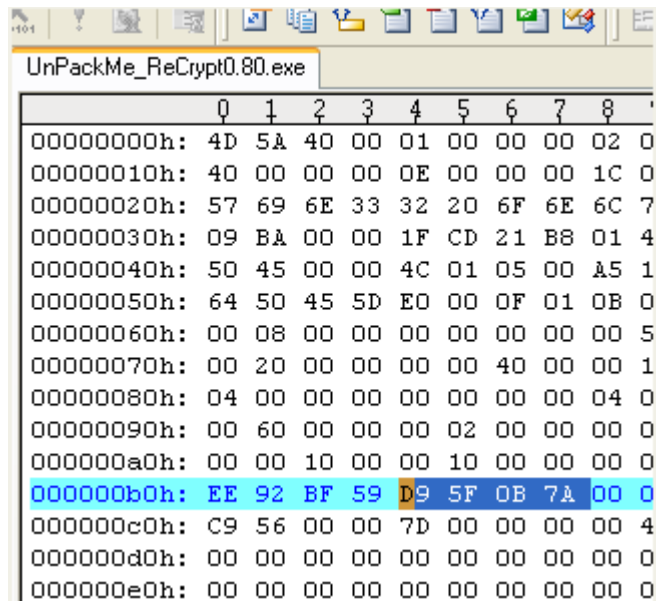
004000AC	00001000	DD 00001000	SizeOfStackReserve = 10000 (40960.)
004000B0	EE92BF59	DD 59BF92EE	LoaderFlags = 59BF92EE
004000B4	10000000	DD 00000010	NumberOfRvaAndSizes = 10 (16.)
004000B8	00000000	DD 00000000	Export Table address = 0
004000BC	00000000	DD 00000000	Export Table size = 0
004000C0	C9560000	DD 000056C9	Import Table address = 56C9

接下来我们在选中该字段,单击鼠标右键选择 Copy to executable file 尝试将所做的修改保存到文件。



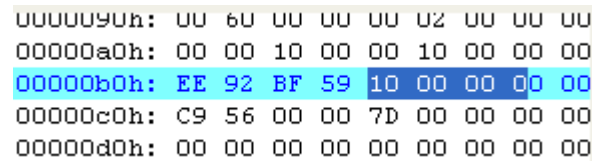
我们可以看到报错了,无法在可执行文件中定位到该数据。

好那我们尝试用十六进制编辑器来修改吧。

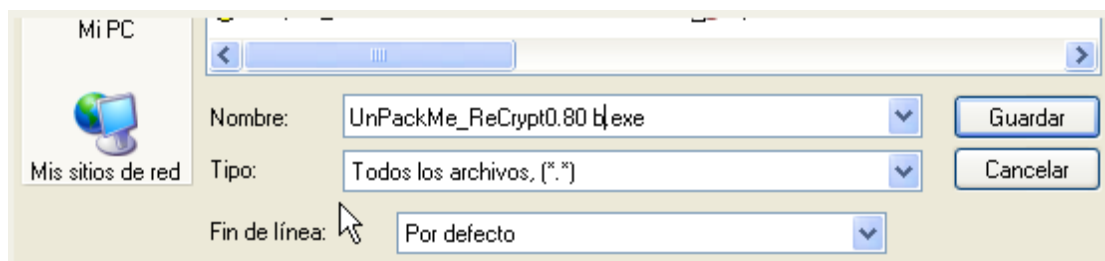


(PS:这里使用的十六进制编辑器是 UltraEdit,当然你也可以使用其他十六进制编辑器,比如 WinHex)。

这里我们定位到 0xB4 偏移处,也就是 NumberOfRvaAndSizes 字段处,就该字段的值修改为 0x10。



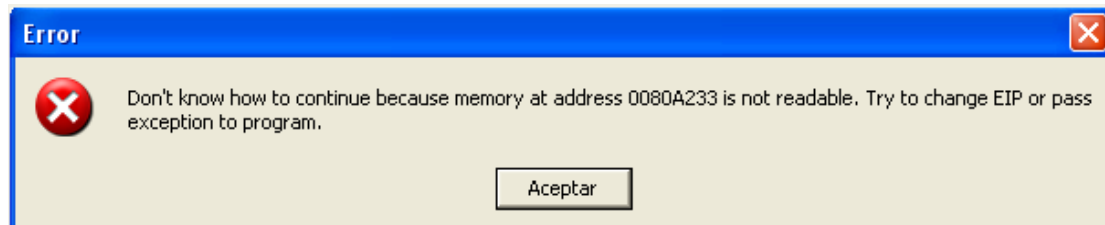
接着将所做的修改保存到文件。



这里我将修改过的文件重命名为了 UnPackMe_ReCrypt0.80 b.exe。

Address	Hex dump	Data	Comment
0040009E	0000	00 0000	DLLCharacteristics = 0
004000A0	00001000	00 00100000	SizeOfStackReserve = 100000 (1048576.)
004000A4	00100000	00 00001000	SizeOfStackCommit = 1000 (4096.)
004000A8	00001000	00 00100000	SizeOfHeapReserve = 100000 (1048576.)
004000AC	00100000	00 00001000	SizeOfHeapCommit = 1000 (4096.)
004000B0	EE92BF59	00 59BF92EE	LoaderFlags = 59BF92EE
004000B4	095F0B7A	00 7A0B5FD9	NumberOfRvaAndSizes = 7A0B5FD9 (20475657)
004000B8	00000000	00 00000000	Export Table address = 0
004000BC	00000000	00 00000000	Export Table size = 0

好,这里 NumberOfRvaAndSizes 的值我们已经还原了,运行起来。



很明显有什么地方我们没有绕过。提示正在访问一个不存在的区段地址,OD 无法继续运行。

这里常规思路是行不通的,我们需要一点奇招。大家需要做的就是:

琢磨

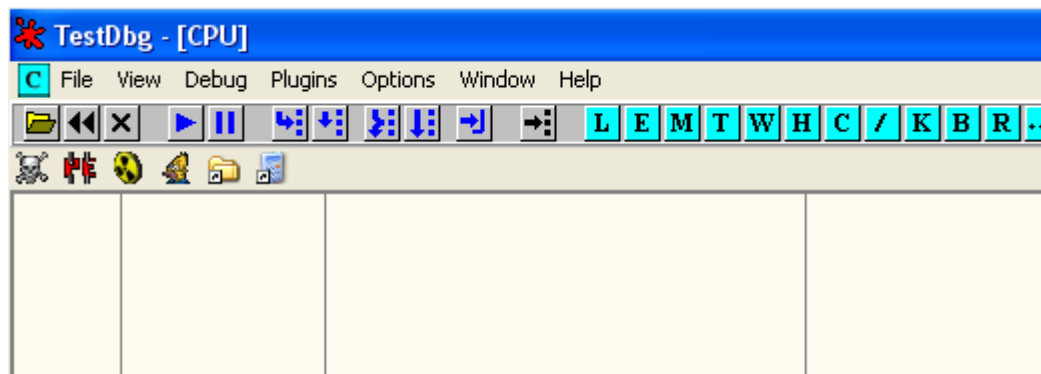
琢磨

琢磨

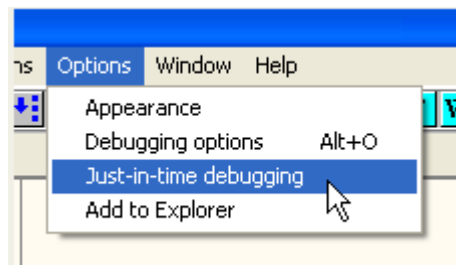
嘿嘿

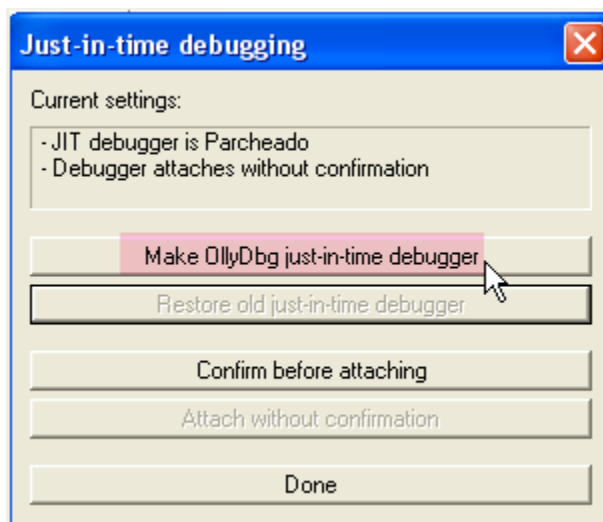
多思考思考

好了,下面我就来给大家介绍一些思路,这里并不是每一种思路都能奏效,但是你需要熟悉这些思路,因为以后经常会碰到。

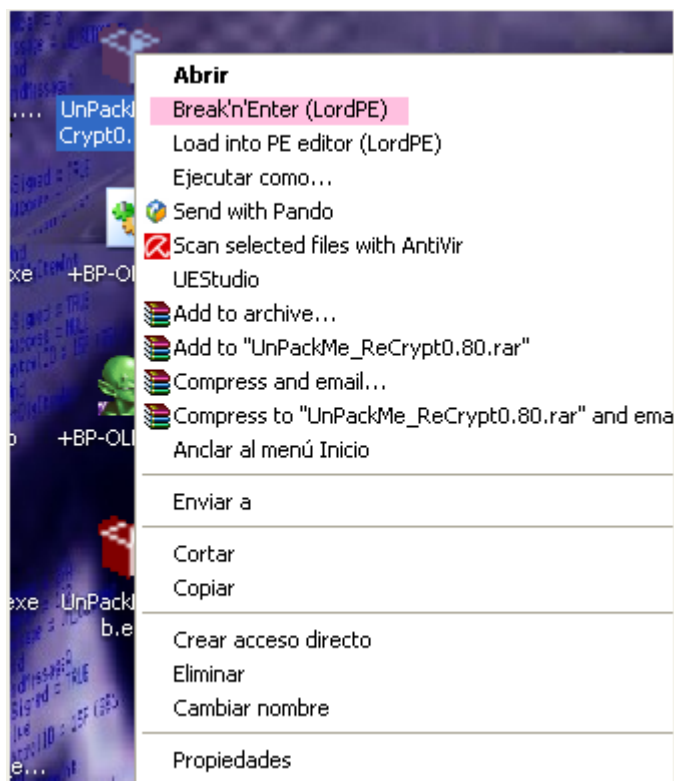


下面打开 Patched 5 这款 OD,选择主菜单项中的 Option-Just-in-time debugging,将其设置为即时调试。(Just-in-time debugging)即时调试:(PS:这个功能大家应该不会陌生吧)当有程序崩溃的时候,OD 就会自动附加上该程序。





我们单击 Make OllyDbg just-in-time debugger(将 OD 设置为即时调试器)按钮。如果以后你不想 OllyDbg 作为即时调试器的话,可以单击 Restore old just-in-time debugger(恢复原来的即时调试器)按钮。因为每当应用程序崩溃,就被 OD 附加上是很烦的。现在我们关闭 OD,修改了 NumberOfRvaAndSizes 字段的 UnPackMe_ReCrypt0.80 b.exe 暂时不用了,因为修改 NumberOfRvaAndSizes 并不起作用,我们选中原程序 UnPackMe_ReCrypt0.80.exe,单击鼠标右键。

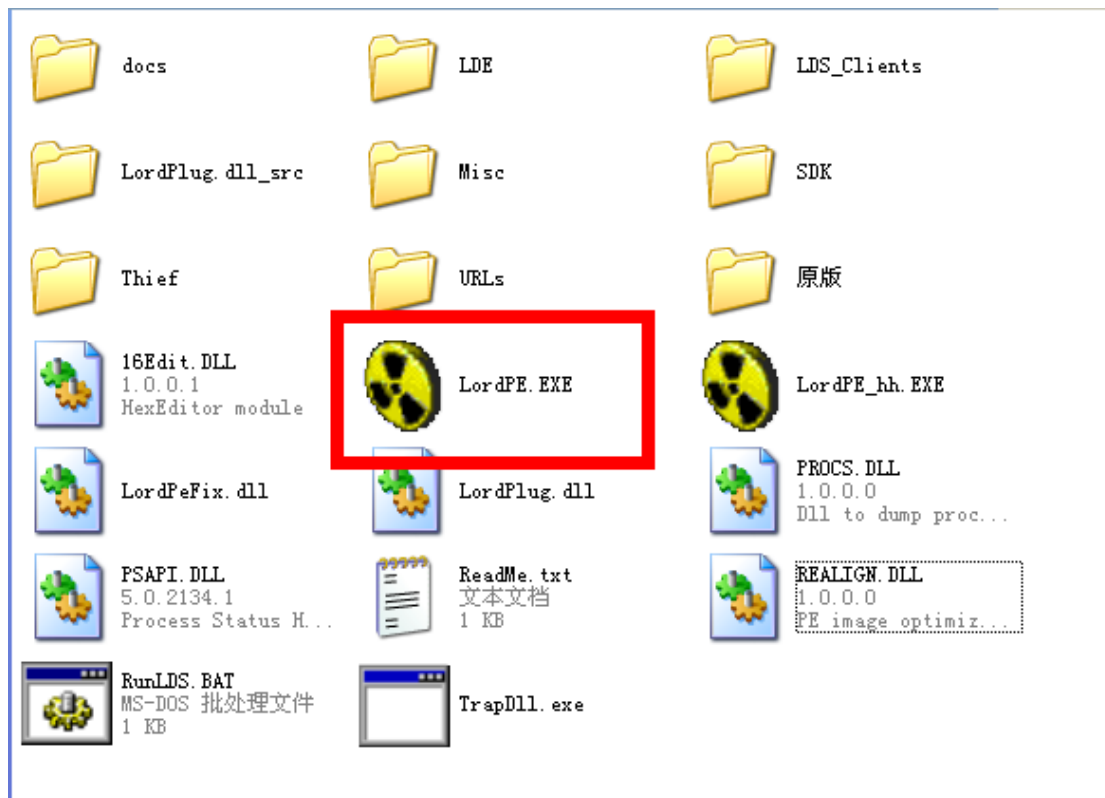


我们可以看到系统右键菜单上有个选项 Break'n'Enter(LordPE)。这个选项的作用是给指定的应用程序入口点处设置一个 INT 3 断点,直接运行该程序就会崩溃,然后即时调试器就会附加之,方便我们的调试。

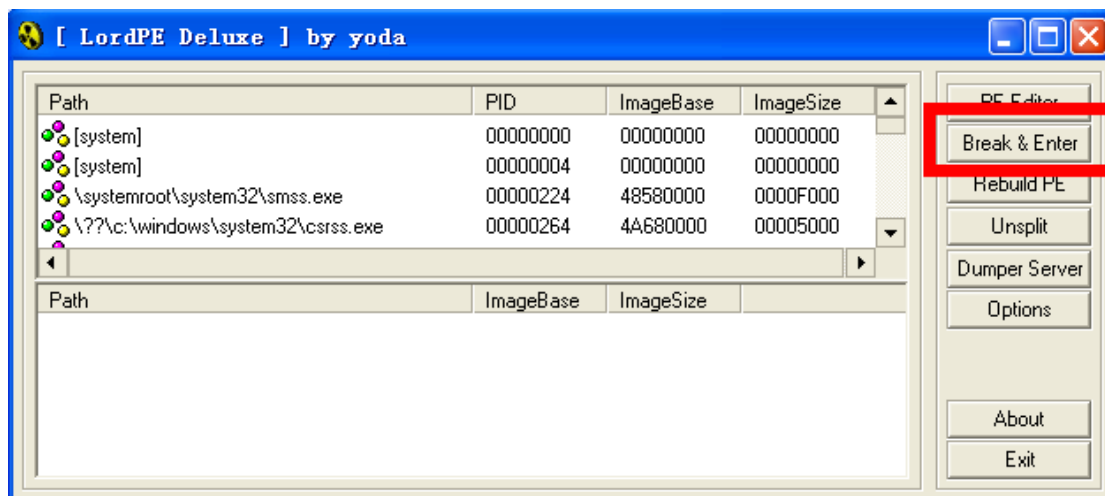
(PS:如果大家没有配置 LordPE 的话,系统菜单是不会出现 Break'n'Enter(LordPE)以及 Load into PE editor(LordPE)这两个菜单项的)。

大家做如下配置就可以出现以上两个菜单项了。

打开 LordPE 主程序。

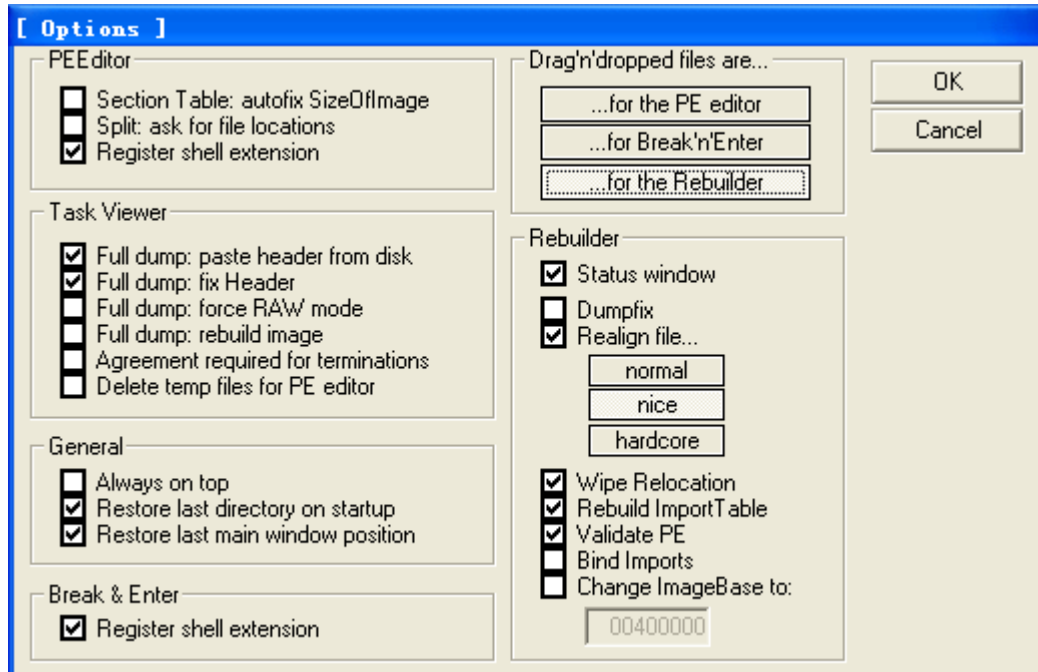


我们可以看到 LordPE 主界面中有 Break & Enter 这么一个按钮。

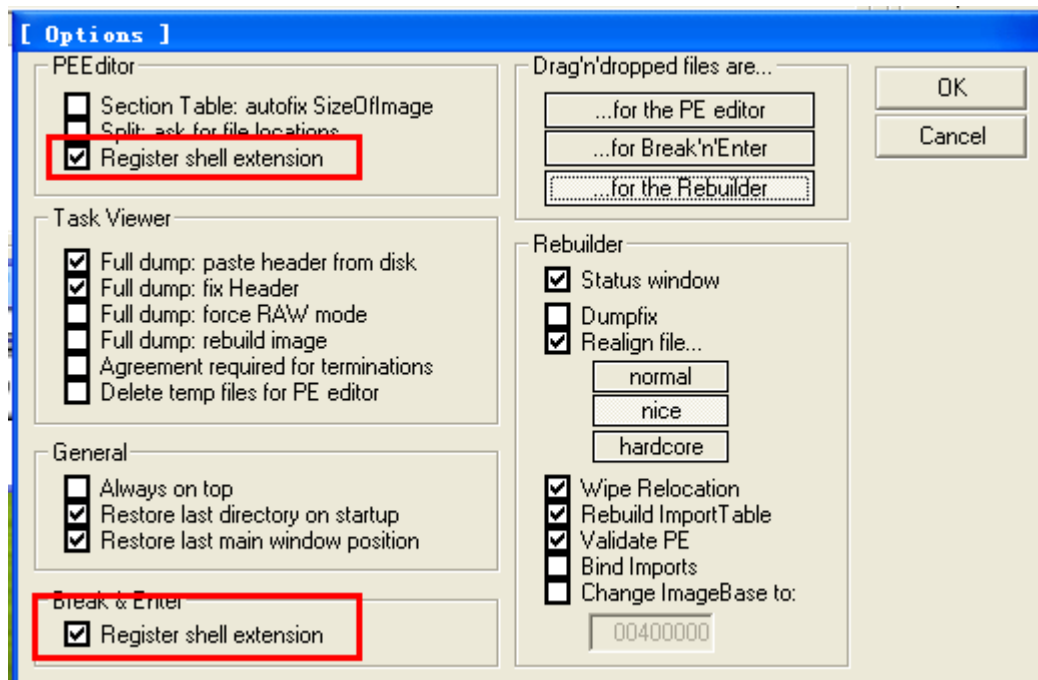


这个跟系统菜单项 **Break'n'Enter**(LordPE)的作用是一样的。

下面我们单击 Options 按钮对 LordPE 进行配置。

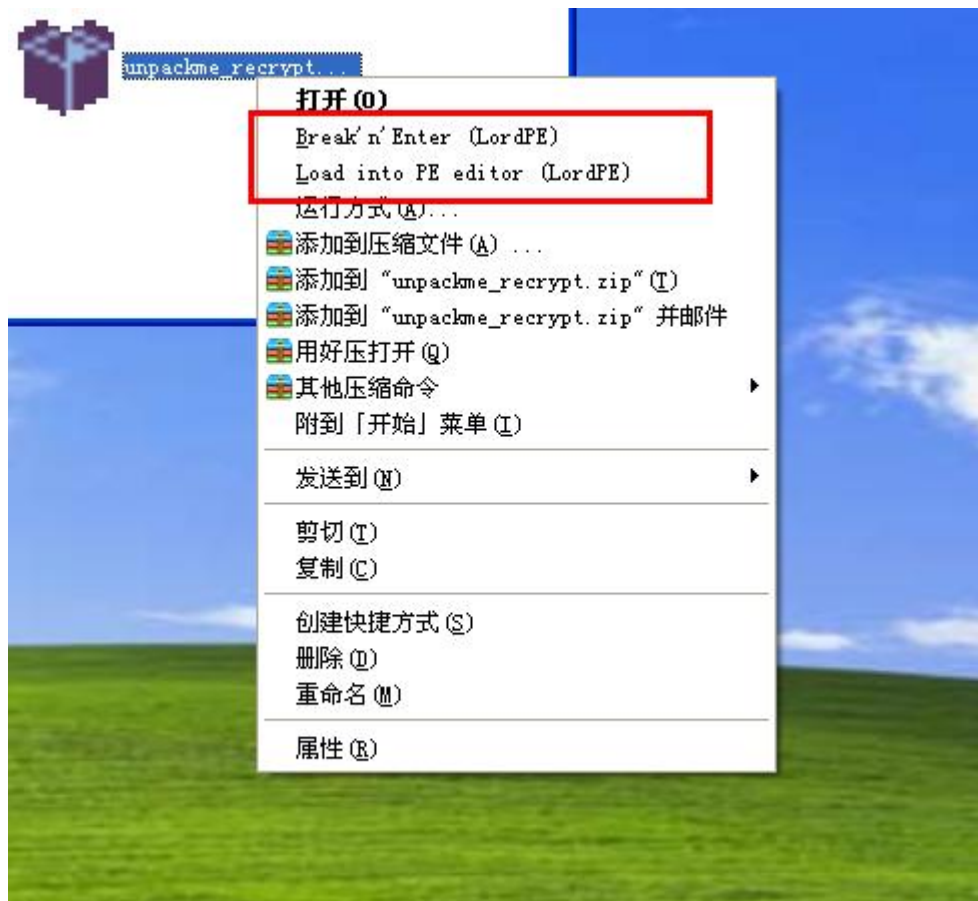


这里我们勾选上 PEEditor 以及 Break & Enter 分组中的 Register shell extension 选项就可。

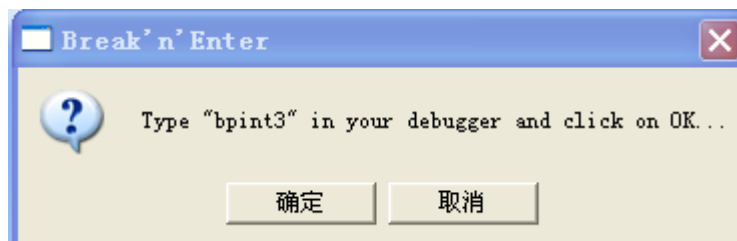


单击 OK 按钮。然后单击 LordPE 主界面中的 Exit 按钮退出。

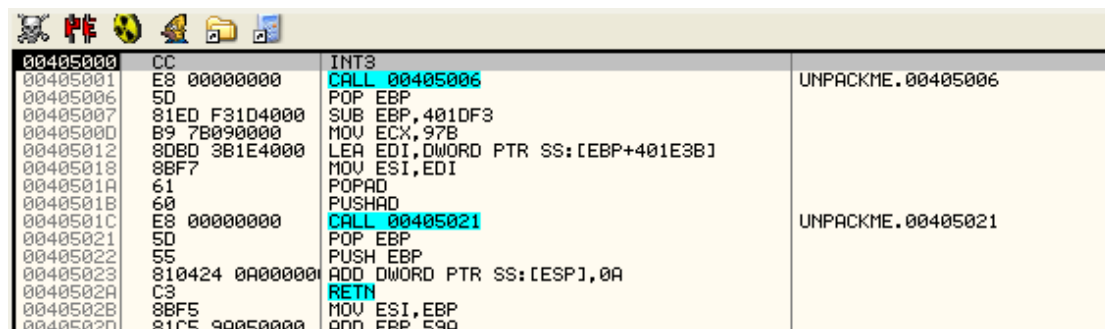
LordPE 的目录下就会生成一个名为 LordPE.ini 的配置文件。大家在选中 UnpackMe_Recrypt 0.80.exe, 单击鼠标右键, 看看是不是多了两个菜单项, 嘿嘿。



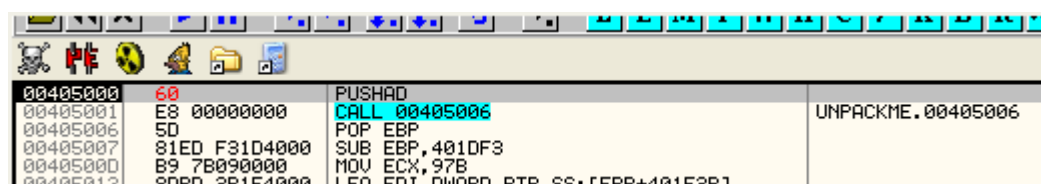
我们单击 **Break'n'Enter(LordPE)** 菜单项。



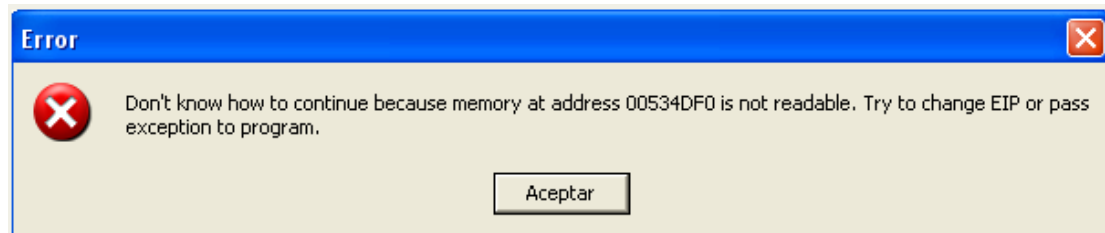
我们单击确定按钮。



OD 立马附加上了该程序,我们可以看到入口点处的第一个字节被 LordPE 修改为了 INT3 指令。我们将 INT3 恢复为 PUSHAD 指令。

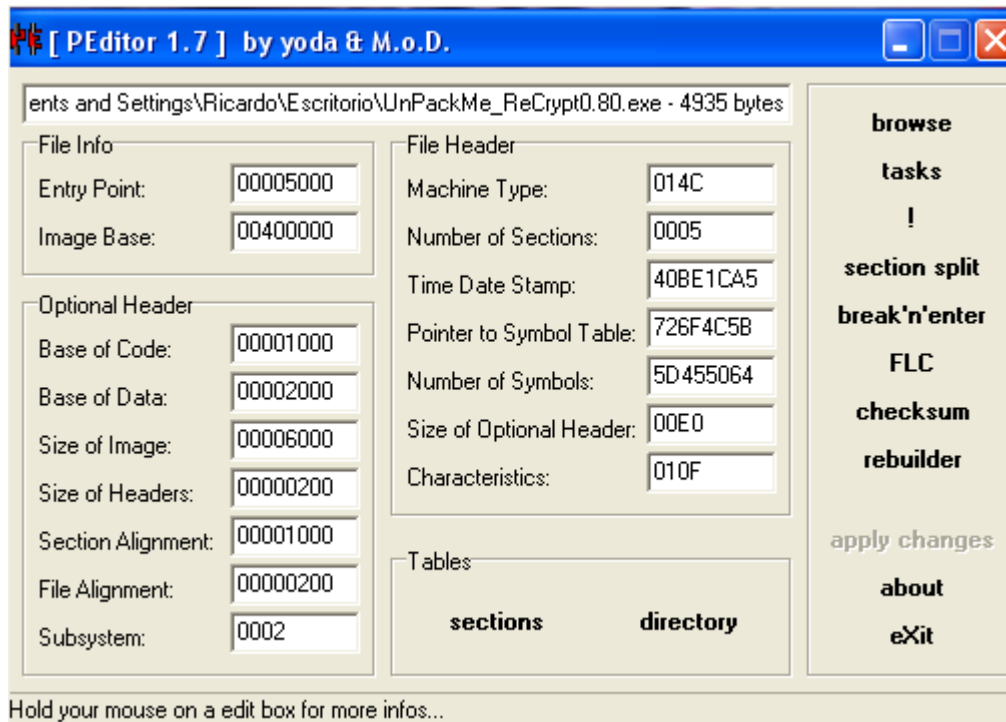


运行起来。

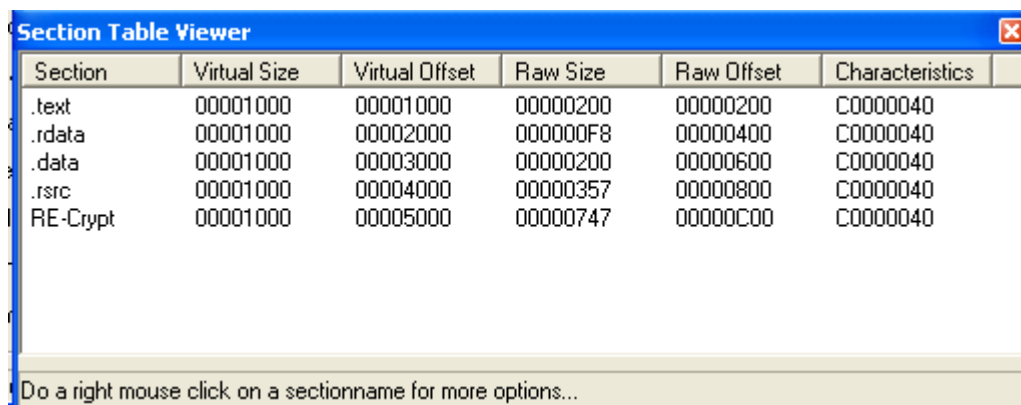


还是报这个错误。那我们只能一步步来跟踪咯,看看它都做了什么。

我们用 PEEditor 打开 UnpackMe_ReCrypt 0.80.exe 文件。



我们单击 sections 按钮查看一下区段。



我们可以看到通过 PEEditor 可以完美的查看各个区段。现在我产生了一个想法:希望这个想法可以奏效,嘿嘿。

首先我们对这个程序具体情况一无所知。那我们就考虑一下常规情况的吧,一般来说,针对于该程序大部分壳会从 401000 地址处开始解密区段。我们可以尝试修改某些区段的访问权限,比如说我们去掉某个区段的可写权限。当壳尝试写入该区段的话,就会报错,此时我们之前设置 Patched 5 这个即时调试器就能派上用场了。我们一起来试一试吧。我们可以从第一个区段开始尝试,但是考虑到壳解密区段过程中会将 IAT 的中一些 API 函数地址写入到.rdata 这个区段中(也就是紧随其后的一个区段),所以这里我们首先尝试.rdata 这个区段,如果.rdata 不起作用的话,我们再来尝试第一个区段。

Section Table Viewer						
Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics	
.text	00001000	00001000	00000200	00000200	C0000040	
.rdata	000001000	00002000	000000F8	00000400	C0000040	
.data	00			00000600	C0000040	
.rsrc	00			00000800	C0000040	
RE-Crypt	00			00000C00	C0000040	

C:\Program Files\Trents and Settings\Ricardo\Trents\UnPackMe_Hel\unpack.exe - 49 kb bytes

Edit Section: .rdata

Current Values	New Values
Name: .rdata	Name: .rdata
Virtual Size: 00001000	Virtual Size: 00001000
Virtual Offset: 00002000	Virtual Offset: 00002000
Raw Size: 000000F8	Raw Size: 000000F8
Raw Offset: 00000400	Raw offset: 00000400
Characteristics: C0000040	Characteristics: C0000040

apply changes
 do nothing
 char. wizard

To modify a value just change it in the right box and click on apply changes...

这里单击右下角的 char.wizard 按钮打开向导页面修改该区段的访问权限。

Section Characteristic Wizard

Flags of the Characteristics of the Section:

- ☐ sharable in memory
- ☐ executable as code
- ☒ readable
- ☒ writeable
- ☐ contains extended relocations
- ☐ discardable as needed
- ☐ can't be cached
- ☐ not pageable
- ☐ contains COMDAT data
- ☐ contains comments or other infos
- ☐ will not become part of the image
- ☐ contains executable code
- ☒ contains initialized data
- ☐ contains uninitialized data
- ☐ shouldn't be padded to next boundary

take it
 do nothing

Current characteristics: C0000040

Click on the checkboxes to build your own characteristics :)

这里我们去掉 writeable(可写)复选框上的对勾。

Section Table Viewer					
Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	00001000	00001000	00000200	00000200	C0000040
.data	00001000	00002000	000000F8	00000400	40000040
.data	00001000	00003000	00000200	00000600	C0000040
.rsrc	00001000	00004000	00000357	00000800	C0000040
RE-Crypt	00001000	00005000	00000747	00000C00	C0000040

Do a right mouse click on a section name for more options...

单击右上方的 take it(确定)按钮。这里我们可以看到,.rdata 这个区段的访问权限已经修改成功了。有可能壳的作者会在运行时调用 VirtualProtect 之类的 API 函数将各个区段的访问权限修改回去。没有关系,我们还是来看看先。

这里我们直接双击运行已经被修改区段访问权限的 UnpackMe_Recrypt 0.80.exe。即时调试器 Patched 5 立马启动了。

The screenshot shows a debugger window with the following content:

Assembly View:

Address	Disassembly	Comment
004052C6	MOV DWORD PTR DS:[EDI],EAX	user32.MessageBoxA
004052C8	MOV ECX,DWORD PTR SS:[EBP+68]	
004052CB	MOV DWORD PTR DS:[ECX],EAX	
004052CD	ADD EDI,4	
004052D0	ADD DWORD PTR SS:[EBP+68],4	
004052D4	JMP SHORT 004052B7	
004052D6	ADD ESI,10	
004052D9	JMP SHORT 004052D0	
004052DB	JNZ SHORT 00405323	
004052DD	INC ESI	
004052DE	JMP SHORT 004052E2	
004052E0	JMP NEAR DWORD PTR DS:[EB02C683]	
004052E6	LODS DWORD PTR DS:[ESI]	
004052E7	CALL 004052E0	
004052EC	JE SHORT 00405346	
004052EE	POP EBX	
004052EF	CMP EAX,EBX	
004052F1	JLE SHORT 004052FA	
004052F3	XOR DWORD PTR DS:[EBX],EAX	
004052F5	ADD EBX,4	
004052F8	JMP SHORT 004052EF	
004052FA	CALL 00405316	
004052FF	???	Unknown command
00405301	???	Unknown command
00405303	INC EBX	
00405304	JE SHORT 0040536B	
00405306	POPAD	
00405309	PUSH EDX	
0040530A	INS DWORD PTR ES:[EDI],DX	I/O command
0040530C	OUTS DX,DWORD PTR ES:[EDI]	I/O command
0040530D	JE SHORT 00405374	
0040530F	PUSH ESP	
00405310	PUSH 64616572	

Hex Dump View:

Address	Hex dump	ASCII
00402068	F2 E1 B3 15 0C AE 31 83 4F C9 13 EE B5 57 C9 EA	=p!\$.«1ã0f!!~ÄWfÜ
00402078	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00402088	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

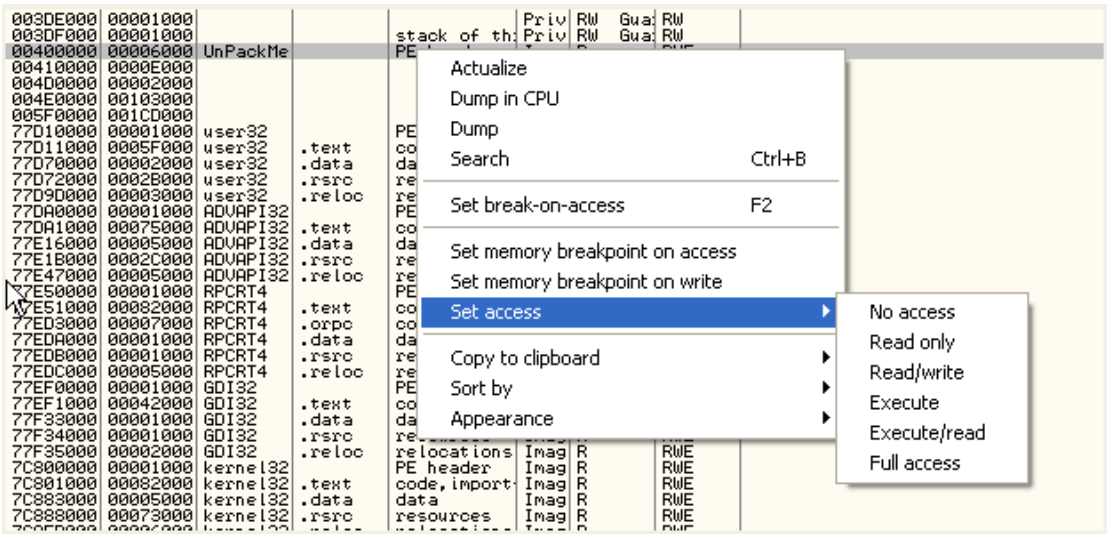
断在了这里,该指令尝试将 MessageBoxA 这个 API 函数的地址保存到.rdata 这个区段对应的地址空间中(嘿嘿,说明我的猜想是正确的)。由于我们刚刚以前去掉了.rdata 段的 writable 属性,说明当执行到该指令时,程序就会发生异常崩溃,即时调试器 OD 就会自动附加之。

好,我们继续。下面我们需要手工将该区段的访问属性修改回去,让其能够对该区段进行正常的写入。

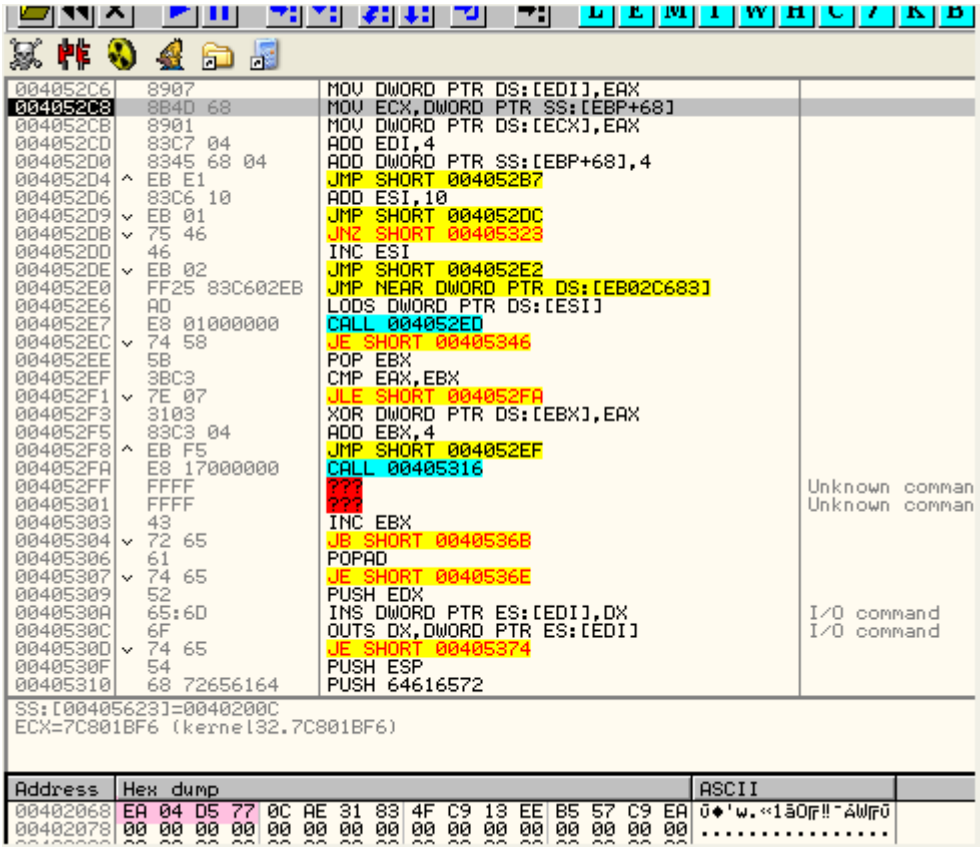
Address	Virtual Size	Section Name	Section Type	PE Header	Priv	RW	Gua	RW
003DE000	00001000				Priv	RW	Gua	RW
003DF000	00001000				Priv	RW	Gua	RW
00400000	00006000	UnPackMe	PE header	Image	R	E		RWE
00410000	0000E000			Map	R	E		R E
004D0000	00002000			Map	R	E		R E
004E0000	00103000			Map	R	E		R
005F0000	001CD000			Map	R	E		R E
77D10000	00001000	user32	PE header	Image	R			RWE
77D11000	0005F000	user32	.text	code,import	Image	R		RWE
77D70000	00002000	user32	.data	data	Image	R		RWE

这里我们可以看到 OD 仅仅显示了一个区段,我们刚刚在 PEEditor 中看到显示的是 5 个区段,没有关系。我们直接在该区段上单击

鼠标右键选择 Full access,给该区段赋予所有权限。



接下来我们单击 F7 单步试试,看看能不能执行该指令。(PS:存在两种情况,情形 1:该指令成功执行 情形 2:报错)



嘿嘿,我们可以看到执行功能执行了,并没有报错。由于 OD 中只显示了一个区段,看起来不是很方便,我们还是来看看 PEEditor 中的区段排列情况。

这里我们选中 0x401000 地址处的第一个字节,按住鼠标左键不放,将鼠标往下拖,直到选中 0x401000-0x402000 整个范围位置。

Address	Hex dump	ASCII
00401EB0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401EC0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401ED0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401EE0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401EF0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401F00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401F10	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401F20	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401F30	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401F40	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401F50	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401F60	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401F70	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401F80	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401F90	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401FA0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401FB0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401FC0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401FD0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401FE0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401FF0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00402000	D4 20 00 00 C6 20 00 00 00 00 00 00 AC 20 00 00	É ..â
00402010	9A 20 00 00 88 20 00 00 7C 20 00 00 00 00 00 00	û ..ë
00402020	68 20 00 00 20 20 00 00 00 00 00 00 BA 20 00 00	h
00402030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

接着单击鼠标右键选择 Breakpoint-Memory, on access。

0040104F 8B 00
00401051 837D 0C 10
00401055 75 65
00401057 EB 5B
00401059 68 FF000000
0040105E 68 44304000
00401063 68 E9030000
00401068 FF75 08
0040106B E8 5E000000
00401070 33D2
00401072 83F8 01
00401075 7D 02

JMP SHORT 0040106C
CMP DWORD PTR SS:[
JNZ SHORT 004010BC
JMP SHORT 004010B4
PUSH 0FF
PUSH 403044
PUSH 3E9
PUSH DWORD PTR SS:
CALL 004010CE
XOR EDX, EDX
CMP EAX, 1
JGE SHORT 00401079

Backup
Copy
Binary
Breakpoint
Search for
Go to
Hex
Text
Short
Long
Float
Disassemble
Special
Appearance

Memory, on access
Memory, on write
Hardware, on access
Hardware, on write
Hardware, on execution

Address	Hex dump
00401EB0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401EC0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401ED0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401EE0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401EF0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401F00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401F10	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401F20	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401F30	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401F40	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401F50	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

运行起来。

00402010 1E AC D6 77 1C B1 D3 77 AD A8 D1 77 00
00402020 68 20 00 00 00 00 00 00 00 00 00 00 00 BA
00402030 0C 20 00 00 5C 20 00 00 00 00 00 00 00 00
00402040 E8 20 00 00 00 20 00 00 00 00 00 00 00 00
00402050 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Command
Memory breakpoint when executing [00401000]

Inicio 3 Exp... 5 Yah...

这里我们可以看到断在了 401000 地址处,也就是 OEP 处。

L E M T W H C / K B R ... S

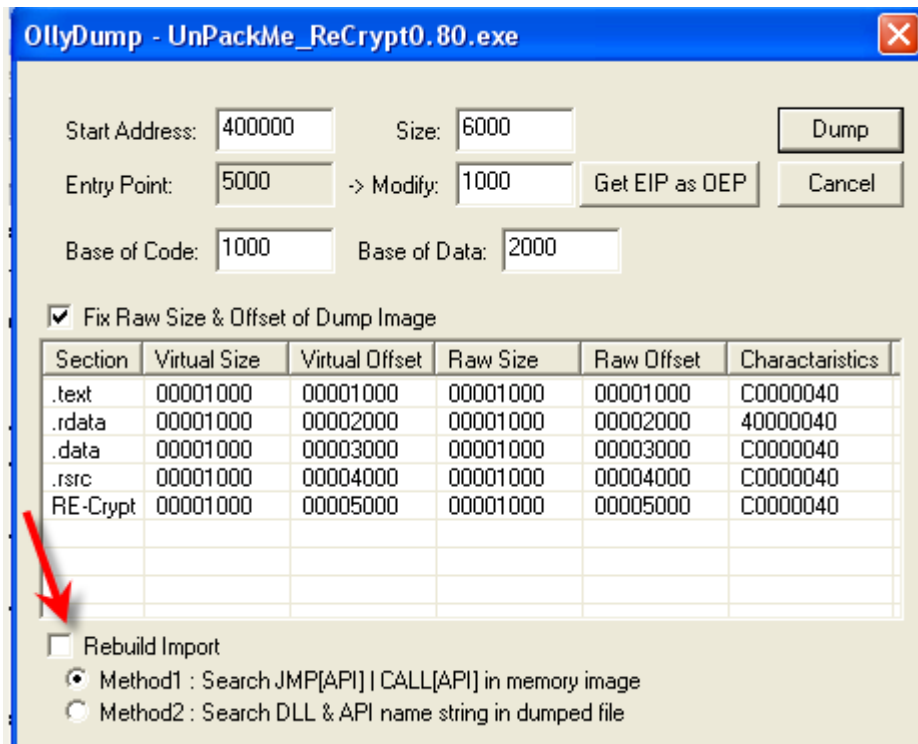
00401000 6A 00
00401002 E8 D9000000
00401007 A3 40304000
0040100C 6A 00
0040100E 68 2B104000
00401013 6A 00
00401015 68 00304000
0040101A FF35 40304000
00401020 E8 A3000000
00401025 50
00401026 E8 AF000000
0040102B 55
0040102C 8BEC
0040102E 817D 0C 110100
00401035 75 1A

PUSH 0
CALL 004010E0
MOV DWORD PTR DS:[403040], EAX
PUSH 0
PUSH 40102B
PUSH 0
PUSH 403000
PUSH DWORD PTR DS:[403040]
CALL 004010C3
PUSH EAX
CALL 004010DA
PUSH EBP
MOV EBP, ESP
CMP DWORD PTR SS:[EBP+0], 111
JNZ SHORT 00401051

JMP to kernel32.GetModuleHandleA

ASCII "Genesis"
JMP to user32.DialogBoxParamA
JMP to kernel32.ExitProcess

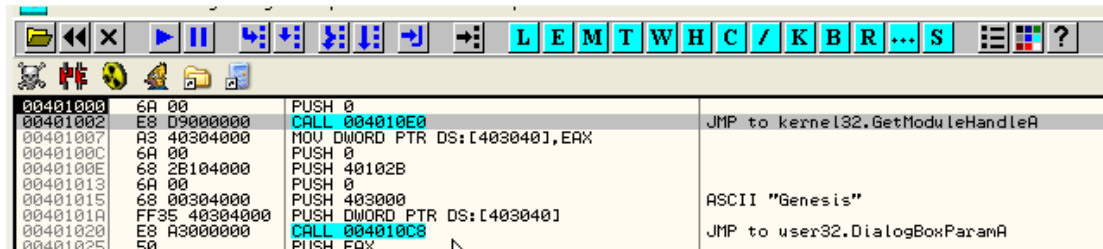
好了,最困难的一步我们已经迈过了,下面我们打开 OllyDump 对该程序进行 dump。



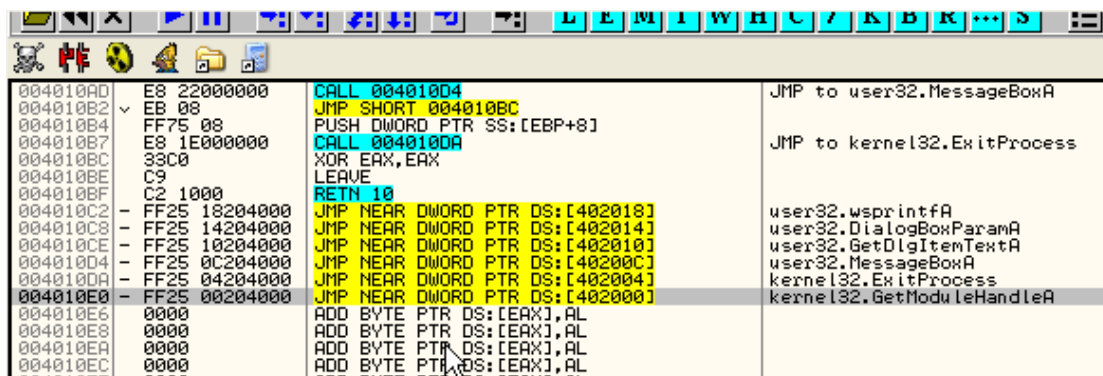
这里我们去掉 Rebuild Import 前面的对勾,不使用 OllyDump 来修复导入表。仅仅用它来 dump。

下面我们来定位 IMP REC 重建 IAT 所需要的数据。

这里我们就拿 OEP 下方的 GetModuleHandleA 这个 API 函数的调用处来说吧。



这里我们在该指令上单击回车键或者单击鼠标右键选择 Follow 跟进。



可以看到我们很容易的就定位到了 IAT。很明显这几项都没有被重定位过。例如:GetModuleHandleA 这个 API 函数对应的项值是 0x402000,很显然是属于 IAT 的。

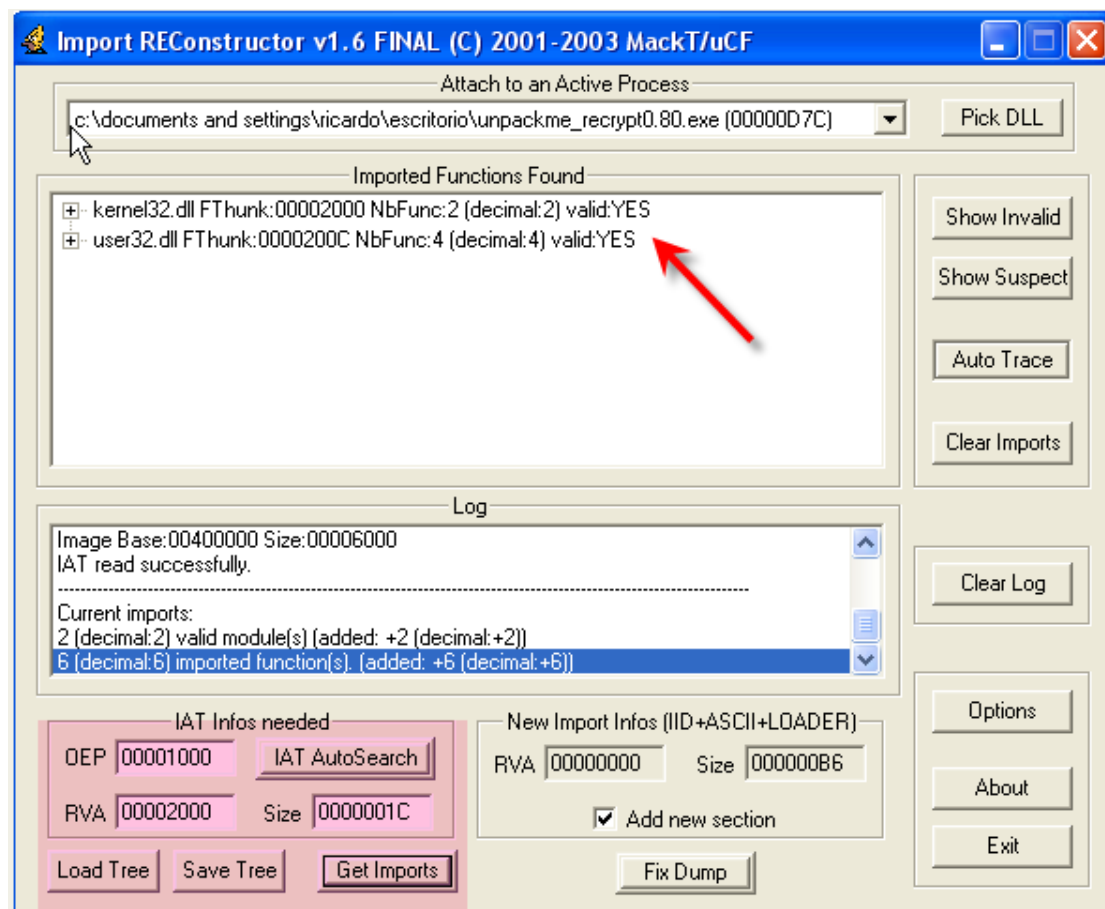
Address	Hex dump	ASCII
00401FF0	00 00 00 00 00 00 00 00
00401FF8	00 00 00 00 00 00 00 00
00402000	29 B5 80 7C A2 CA 81 7C)AÇ!ø=ü!
00402008	00 00 00 00 EA 04 05 77	...U♦'w
00402010	1E AC 06 77 1C B1 03 77	▲%iwlΣEw
00402018	AD A8 01 77 00 00 00 00	¿¿0w.....
00402020	68 20 00 00 00 00 00 00	h.....
00402028	00 00 00 00 E 20 00 00	...
00402030	0C 20 00 00 5C 20 00 00	... \
00402038	00 00 00 00 00 00 00 00
00402040	E8 20 00 00 00 20 00 00	è.....
00402048	00 00 00 00 00 00 00 00
00402050	00 00 00 00 00 00 00 00
00402058	00 00 00 00 29 B5 80 7C	...)AÇ!

我们在数据窗口中定位到 IAT。可以看到该程序的 IAT 很小。IAT 的起始地址为 0x402000,结束地址为 0x40201C,所以 IAT 的长度的为 0x1C。下面我们打开 IMP REC 将 IAT 的信息填充进去。

IAT 起始地址(RVA):0x2000

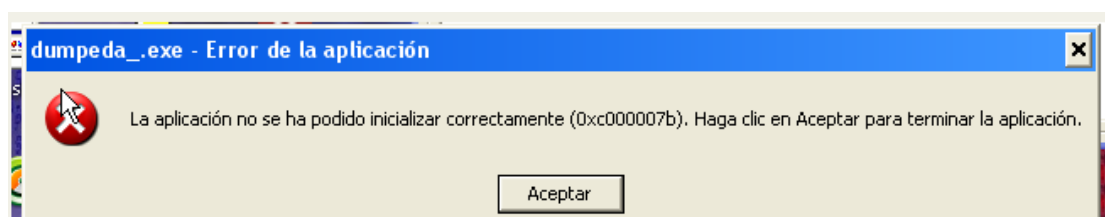
IAT 大小:0x1C

OEP(RVA):0x1000

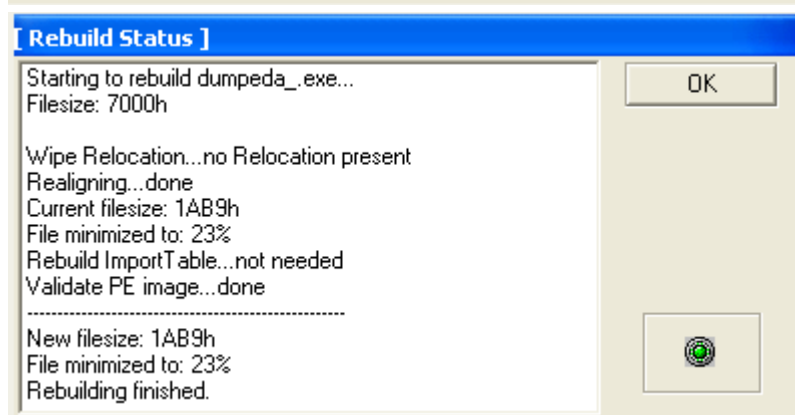
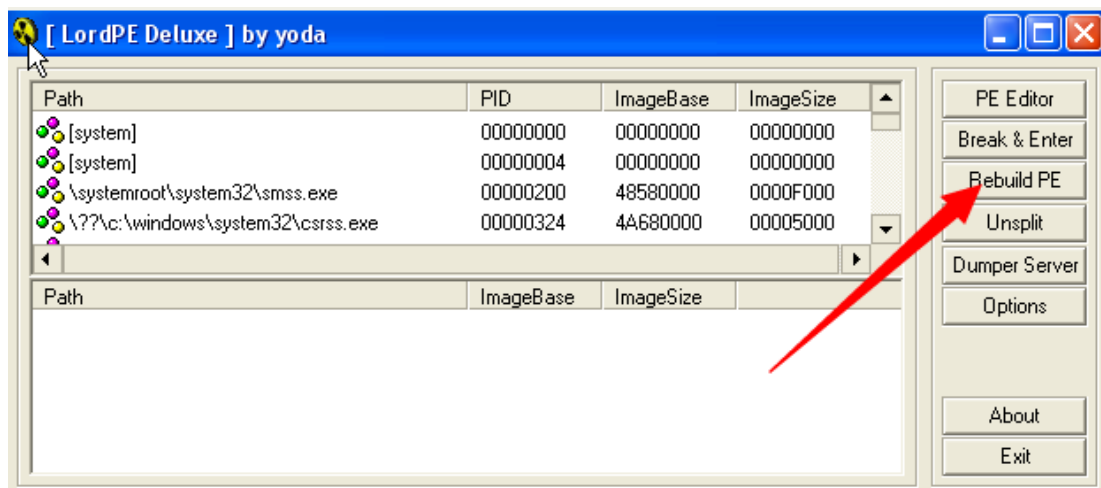


这里我们将 OEP,IAT 的起始地址以及大小填充到 IMP REC 中。接着单击 Get Imports 按钮。我们可以看到获取的 IAT 项都是有效的,啧啧!!!

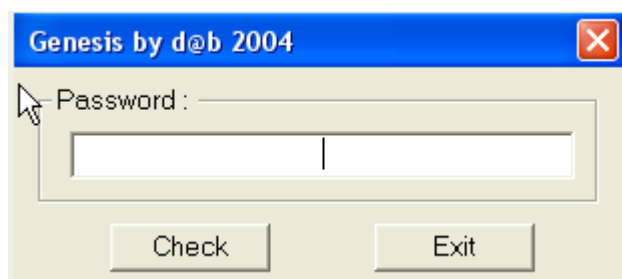
接下来单击 Fix Dump 修复刚刚我们 dump 出来的文件。修复完毕以后直接运行起来,可以看到报错了。



我们将其拖拽到 LordPE 中的 Rebuild PE 按钮上,这样就可以重建 PE 了。



重建 PE 完毕,直接运行起来,嘿嘿,完美运行。



看来我们的剑走偏锋还是有所成效的。总结:大部分壳的加密思路都是有共同之处的,但是有些壳会玩一些花招来抵御常规的脱壳思路,剑走偏锋往往能收到奇效,嘿嘿。