




# 使用LordPE和Import REC脱壳

原创

虾仁炖猪心  于 2020-04-18 17:35:46 发布  1689  收藏 8

分类专栏: [加壳脱壳](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: [https://blog.csdn.net/weixin\\_43575859/article/details/105598559](https://blog.csdn.net/weixin_43575859/article/details/105598559)

版权



[加壳脱壳](#) 专栏收录该内容

3 篇文章 1 订阅

订阅专栏

本博客用到的资源链接放在博客末尾。

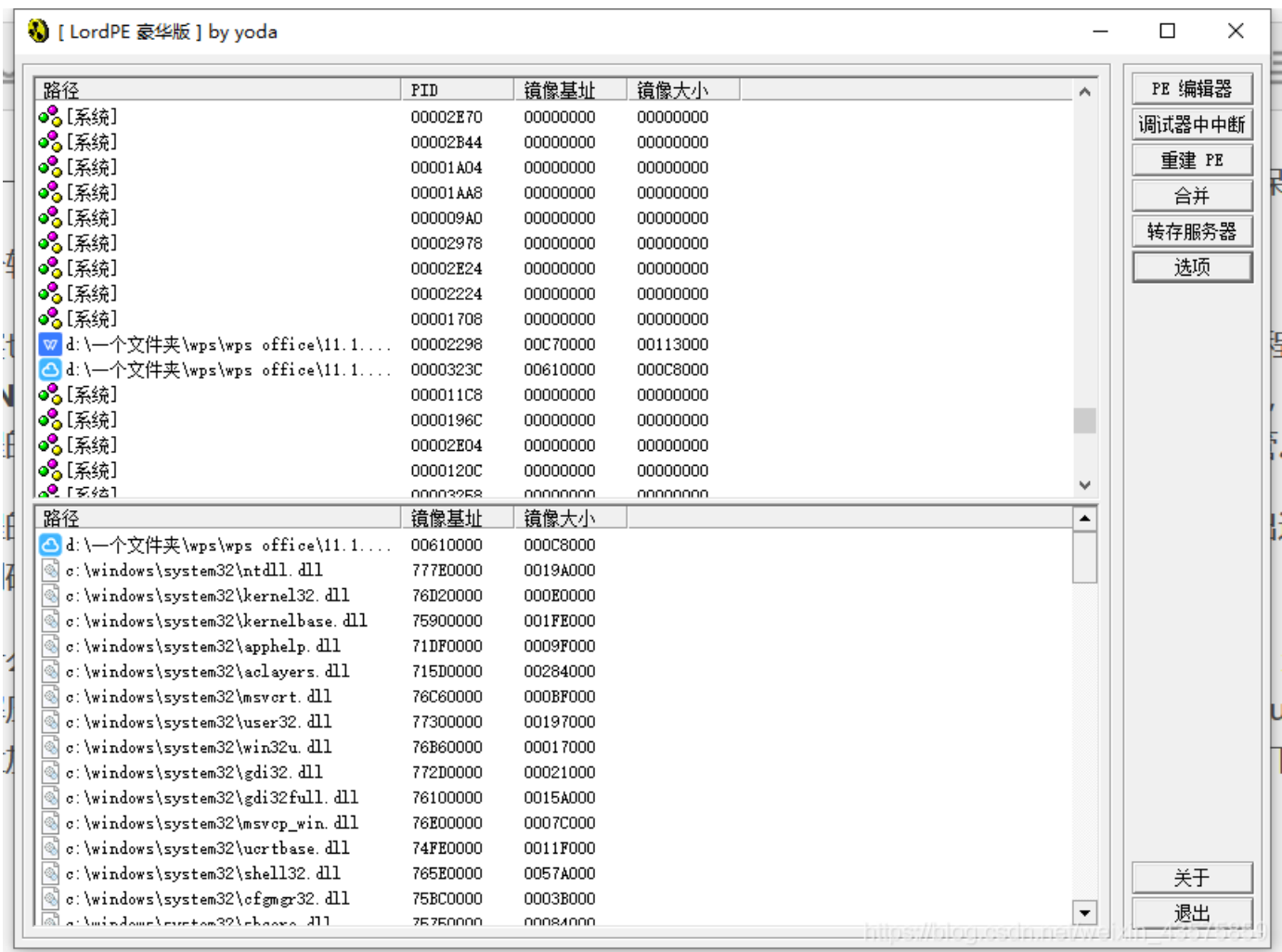
LordPE是一款能够抓取内存映像的用来脱壳的软件, 它能够把一个进程的内存数据给Dump下来, 然后保存为文件。

光了解一个软件的用处用起来如果出问题还是会比较头疼, 所以这里简单说一下LordPE软件的原理。

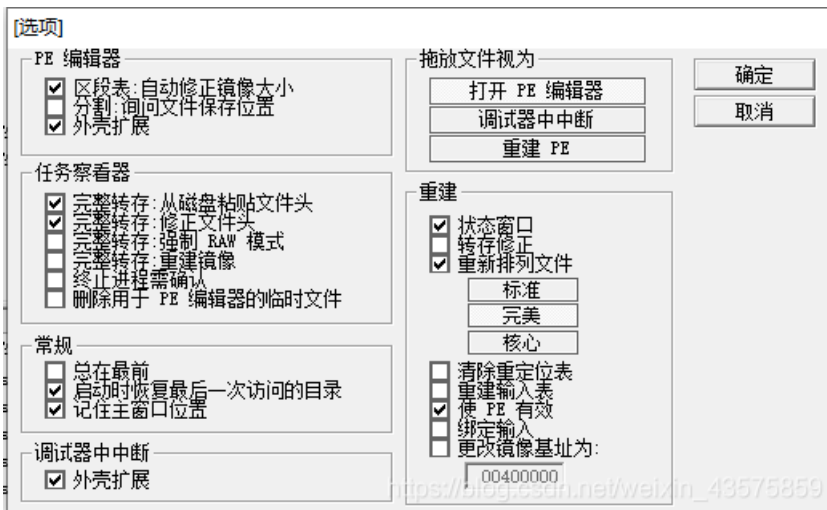
该软件其实也是调用了系统的API。它首先调用CreateToolhelp32Snapshot函数来获取一个系统中的进程的快照。然后再用Module32Next函数来循环获取每个进程的信息, 进程的信息会被填到一个MODULEENTRY32结构里面, 信息包括了该进程的映像基址, 进程的映像大小, 进程的句柄, 进程的完整路径。当然还有一些其他的信息, 不过这些我们不用管。

得到了进程的句柄, 以及映像基址和大小之后, 软件就可以对它进行操作了。软件会根据这个数据读取出进程的内存数据, 然后将数据保存到磁盘文件。

那么它为什么能脱壳呢? 其实用它进行脱壳的过程, 一般是我们先让其他软件将加壳程序调试到OEP处停止, 这个时候程序在内存中已经被壳给解压缩或者解密了, 这个时候的进程的内存数据就是没加壳时候的数据。然后将这些数据Dump下来保存为文件。得到的就是没加壳的原程序文件了。其实就是先让壳对程序自行解压或解密, 然后将解压或解密后的程序数据Dump下来保存为文件。



这个是程序的界面，打开程序时要注意用管理员权限打开，不然可能用OD调试的进程在这里可能看不到。点击选项，然后调整设置成下面这样：



里面有个选项：从磁盘粘贴文件头。意思是Dump下来的数据里面的PE文件头是直接从磁盘原始文件中复制的。

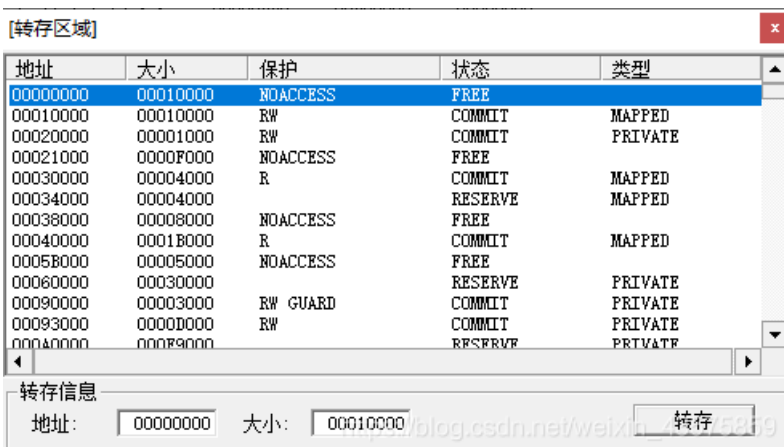
当我们要Dump一个进程的内存数据的时候，右键目标进程，然后选择完整转存就可以了。



但是，既然有这种Dump技术，那就肯定还有反Dump技术，因为LordPE软件是通过MODULEENTRY32结构里面的数据来获取进程的信息的。所以，一些程序的壳会修改里面的modBaseSize和modBaseAddr字段，往里面填入错误的值。而因为如果修改modBaseAddr的值，会让系统出现问题，所以一般修改的都是modBaseSize字段。这样我们Dump下来的就是不完整的数据。

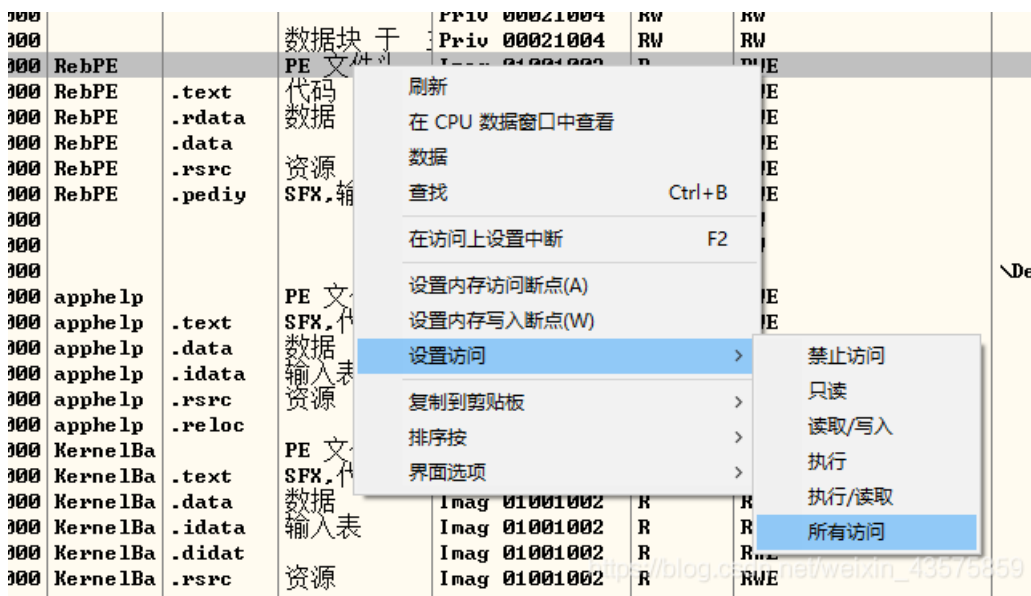
出现这种问题其实解决方法很简单，可以让软件直接到PE文件头里面的SizeOfImage字段读取映像的真正大小。做法就是右键目标几进程，然后选择修正镜像大小。然后再Dump就可以了。

加壳程序还可能用一种技术来防止用户Dump进程数据，就是在程序中使用VirtualProtect函数来设置PE文件头为不可读，这样我们用软件Dump进程的时候就会出现访问错误。在软件界面右键目标进程，然后点击区域转存，就出现了下图：



这里只是举个例子，这个程序还没有进行脱壳调式。可以看到有些区域显示的是NOACCESS。如果加壳程序做了我们上面所说的处理的话，PE头也会出现这种情况。

这个时候我们可以打开OD，按下“Alt+M”或者点击蓝色小框框里面的M打开内存映像，然后在PE文件头那里右键，选择设置访问->所有访问。



下面就试一下用LordPE进行脱壳，用到的加壳程序是书本《加密与解密》里面的例子：RebPE。

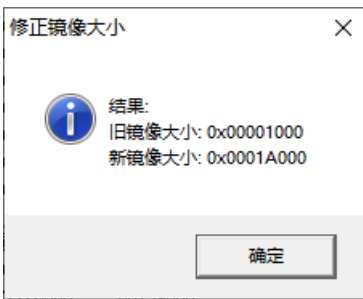
首先我们用OD打开目标程序：

地址	HEX 数据	反汇编	注释
00413000	60	PUSHAD	
00413001	E8 C2000000	CALL RebPE.004130C8	
00413006	2E:3001	XOR BYTE PTR CS:[EAX],AL	
00413009	0000	ADD BYTE PTR DS:[EAX],AL	
0041300B	0000	ADD BYTE PTR DS:[EAX],AL	
0041300D	0000	ADD BYTE PTR DS:[EAX],AL	
0041300F	0000	ADD BYTE PTR DS:[EAX],AL	
00413011	003E	ADD BYTE PTR DS:[ESI],BH	
00413013	3001	XOR BYTE PTR DS:[ECX],AL	
00413015	002E	ADD BYTE PTR DS:[ESI],CH	
00413017	3001	XOR BYTE PTR DS:[ECX],AL	
00413019	0000	ADD BYTE PTR DS:[EAX],AL	
0041301B	0000	ADD BYTE PTR DS:[EAX],AL	
0041301D	0000	ADD BYTE PTR DS:[EAX],AL	
0041301F	0000	ADD BYTE PTR DS:[EAX],AL	
00413021	0000	ADD BYTE PTR DS:[EAX],AL	
00413023	0000	ADD BYTE PTR DS:[EAX],AL	
00413025	0000	ADD BYTE PTR DS:[EAX],AL	
00413027	0000	ADD BYTE PTR DS:[EAX],AL	
00413029	0000	ADD BYTE PTR DS:[EAX],AL	
0041302B	0000	ADD BYTE PTR DS:[EAX],AL	
0041302D	0020	ADD BYTE PTR DS:[EAX],AH	
0041302F	5F	POP EDI	

很明显是被加壳了。然后我们单步执行一下之后，在ESP寄存器中的内存地址上设置内存访问断点，然后运行程序，跟踪到OEP的地方：

0040112F	90	NOP	
00401130	55	PUSH EBP	
00401131	8BEC	MOV EBP,ESP	
00401133	6A FF	PUSH -0x1	
00401135	68 B8504000	PUSH RebPE.004050B8	
0040113A	68 FC1D4000	PUSH RebPE.00401DFC	
0040113F	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	SE 处理程序安装
00401145	50	PUSH EAX	
00401146	64:8925 00000000	MOV DWORD PTR FS:[0],ESP	
0040114D	83EC 58	SUB ESP,0x58	
00401150	53	PUSH EBX	
00401151	56	PUSH ESI	
00401152	57	PUSH EDI	
00401153	8965 E8	MOV DWORD PTR SS:[EBP-0x18],ESP	
00401156	FF15 28504000	CALL DWORD PTR DS:[0x405028]	kernel32.GetVersion
0040115C	33D2	XOR EDX,EDX	
0040115E	8AD4	MOV DL,AH	
00401160	8915 08854000	MOV DWORD PTR DS:[0x408508],EDX	
00401166	8BC8	MOV ECX,ECX	

这个时候进程在内存中已经脱壳完成了，现在我们就用LordPE来Dump数据了。用管理员权限打开软件，然后右键目标进程，先选择修正镜像大小：



软件成功修正了镜像的大小。所以说这个程序也修改了**MODULEENTRY32**结构里面的值。

然后我们再选择完整转存，保存到磁盘文件。

最后我们点击运行一下：



居然出现了一个错误。这就很崩溃了。

但是办法总比困难多，一番查询就会知道，加壳程序会对程序的IAT做手脚，破坏原程序的输入表是加密外壳必须具备的功能。所以上面这个程序虽然我们把它的外壳去掉了，但是它的IAT已经被破坏了，所以无法调用系统API。

一些外壳会改变原程序IAT里面的内容，加壳的时候，由于壳会自建一个输入表，并且让原来的PE文件的数据目录里面输入表指针指向自建的输入表。这样PE文件装载器会对自建的输入表进行填写。而原来的输入表被外壳变形后储存。程序运行的时候外壳将变形后的输入表进行扫描，然后重新填写所有函数的地址。

要注意：因为IAT填写完之后，输入表的其他部分就不重要了，程序依靠IAT里面的函数地址就能正常运行，并且壳只是填了IAT里面的内容，对于输入表其他部分，可以说是已经破坏了，也就是说，加壳程序运行的时候，原程序的输入表已经不在，内存里面就只有一个IAT。

所以像我们上面那样Dump数据后，文件里面的没有输入表的，自然程序运行不了。所以我们要重建输入表。

还有一些外壳，填写到IAT里面的并不是真正的API地址。它填的可能是外壳的HOOK API地址，这样程序每次调用API就会跳转到外壳中运行，这样外壳又可以获得控制权。

这个时候我们就要用到Import REC（全称：Import REConstructor）这个软件了。

这个软件可以帮助我们重建程序的输入表，并且用起来很简单。

运行这个软件前我们要确保几个事情：

- 目标文件已经被像我们上面那样给Dump下来了。
- 目标文件正在运行
- 我们事先知道目标程序的真正OEP或者IAT的偏移量和大小。

所以如果完成了上面的Dump的步骤之后，我们再将原来的没有Dump的程序用OD打开，然后跟踪到OEP处：

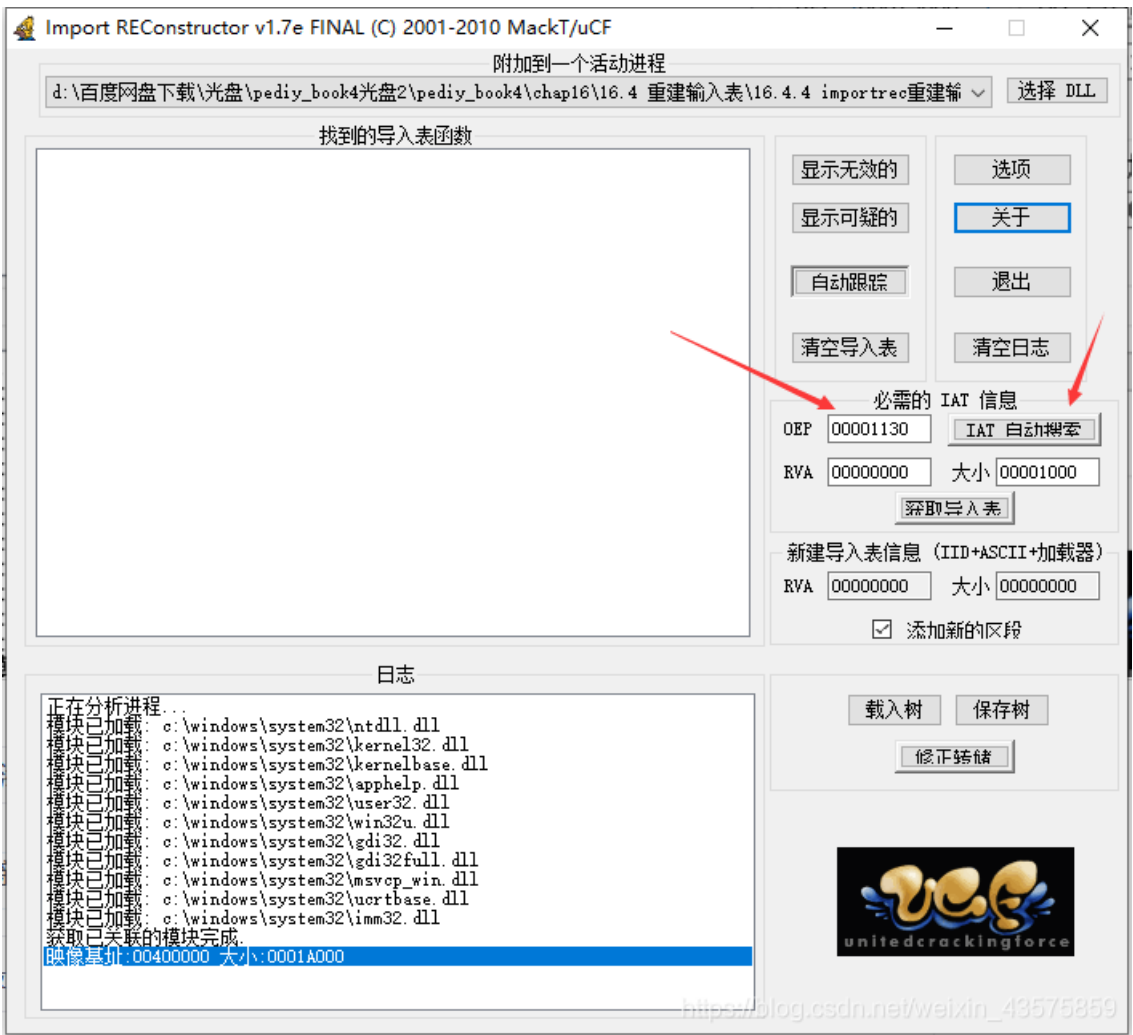
0040112F	90	NOP	
00401130	55	PUSH EBP	
00401131	8BEC	MOV EBP,ESP	
00401133	6A FF	PUSH -0x1	
00401135	68 B8504000	PUSH RebPE.004050B8	
0040113A	68 FC1D4000	PUSH RebPE.00401DFC	
0040113F	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	SE 处理程序安装

得到真正的OEP（箭头所指）。

然后我们打开软件，在下拉栏里面找到目标进程：

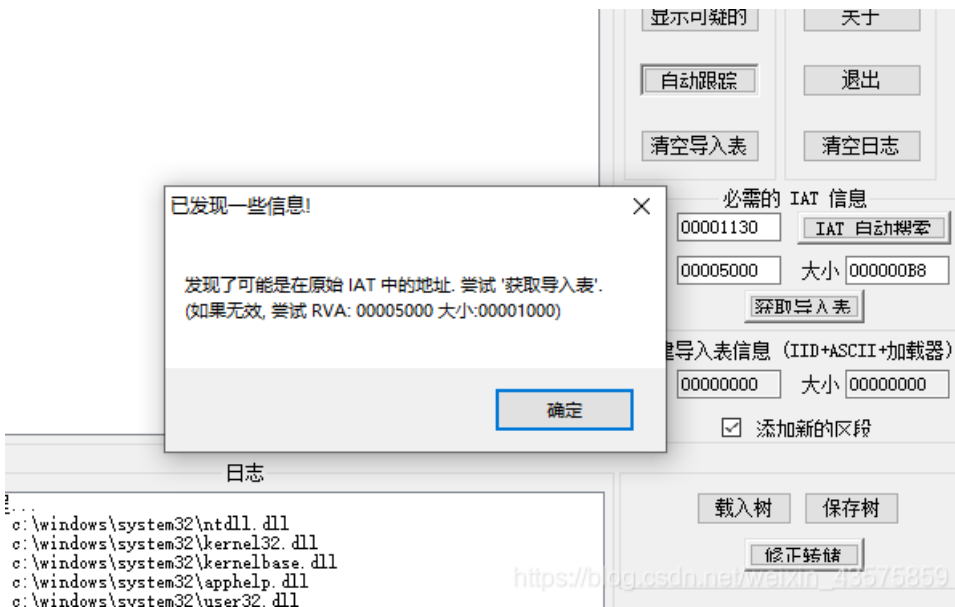


然后在OEP框框那里填入正确的OEP的RVA，然后再点击IAT自动搜索，让软件自动搜索IAT的偏移量和大小：

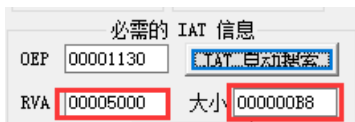


如果OEP发挥作用，那么软件会弹出一个窗口：

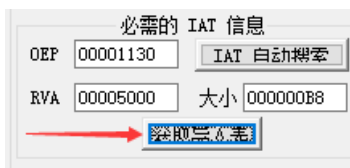




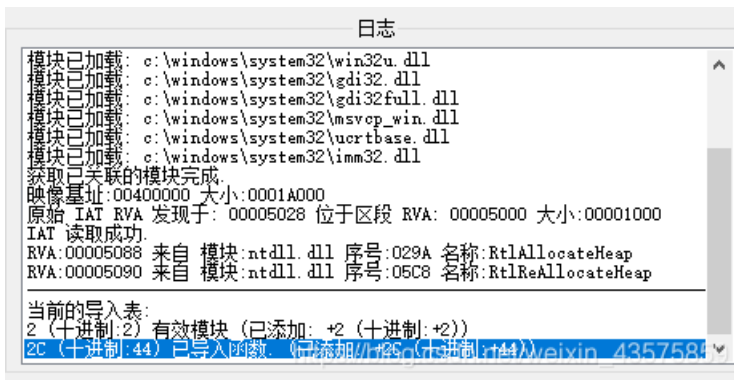
如果说我们没有正确的OEP，或者说软件没有找到IAT的偏移量，那么我们需要手动填入IAT的RVA和SIZE：



完成上面步骤之后，我们点击获取导入表：

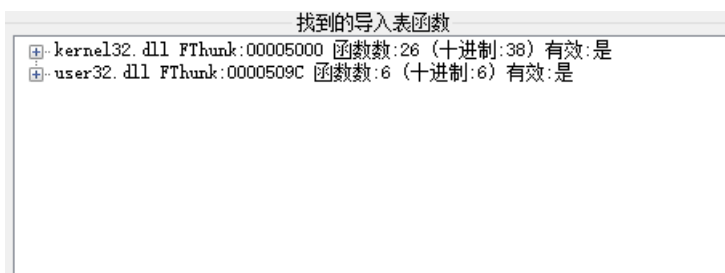


日志框：



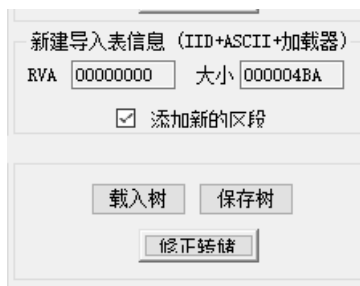
这样就是API都读取成功了。

如果有读取不成功的，就会在下面这个框中显示 有效：否。



这里我们的API都能够识别，如果有不能识别的，右键这个框的空白处，然后选择跟踪级别1，然后点击“显示无效的”，如果还是有无效的，那么再右键空白处，选择跟踪级别2。

最后就到了修复程序的时候了，在下面这个地方的复选框中选中添加新节区：



这样软件会把新的输入表放在这个新的节区里面，这样会不可避免导致程序变大，我们也可以将输入表建在程序空白处，这样我们就得自己填写RVA了。

最后点击修正转储，选择我们之前Dump下来的文件进行转储，软件会生成一个新的文件，OEP也会被自动修正，这个文件就是最后修正了输入表的文件，点击运行：



完全没有问题。至此就脱壳完成了。

加壳程序：[https://download.csdn.net/download/weixin\\_43575859/12337598](https://download.csdn.net/download/weixin_43575859/12337598)

LordPE和Import REC在吾爱破解的爱盘里面都能找到。