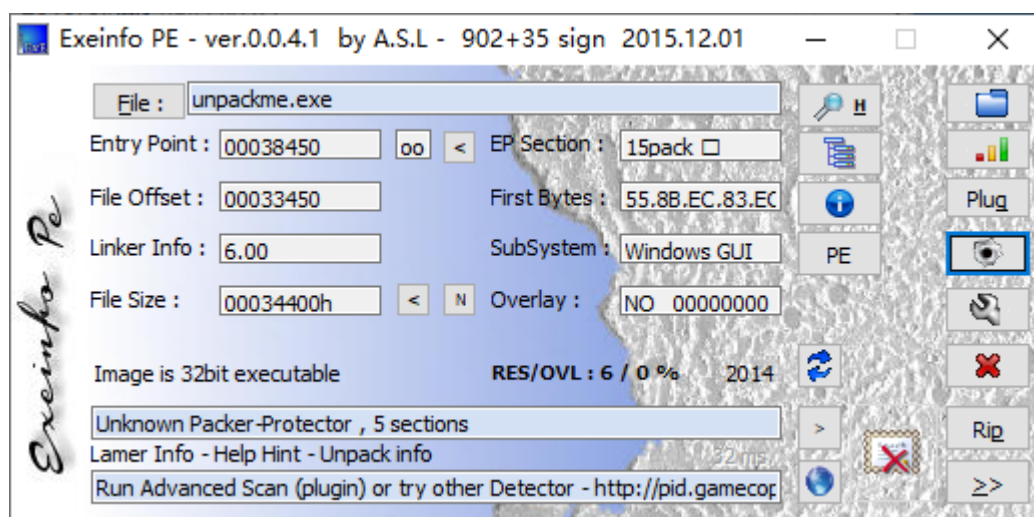


查壳
OD脱壳
解密IAT
修复导入表
破解Crackme

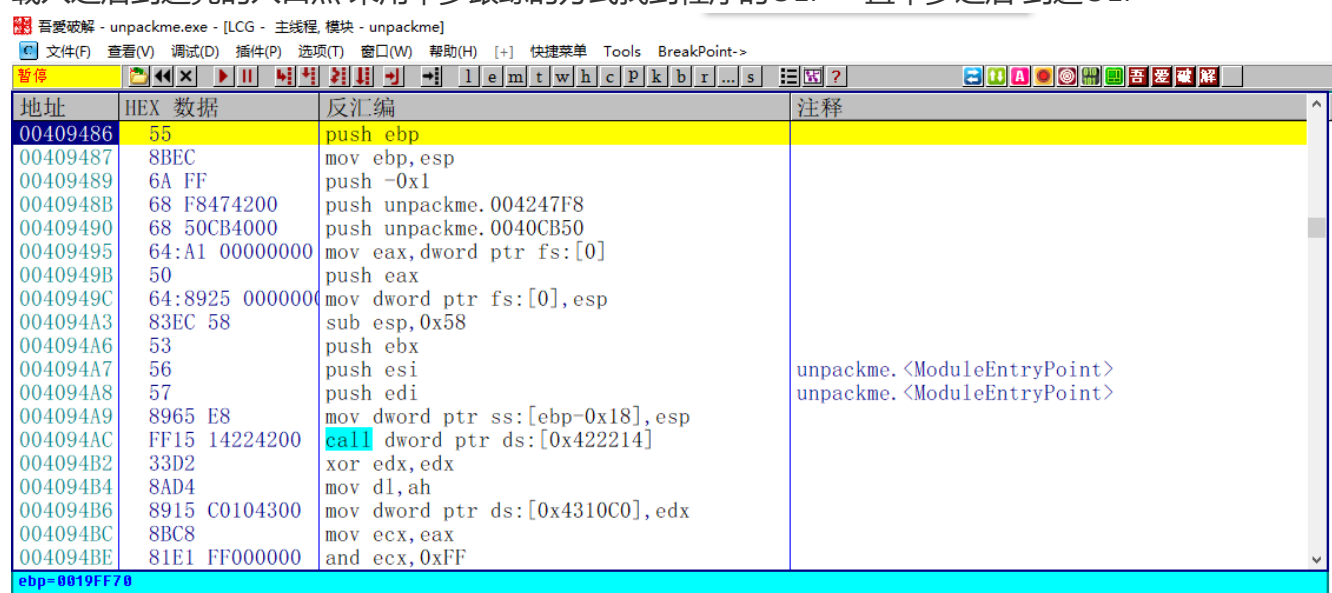
查壳



目标程序链接器版本是6.0 可能是VC6写的程序 PEiD查壳没有查出来，应该是别人自写的一个壳

OD脱壳

载入之后到达壳的入口点 采用单步跟踪的方式找到程序的OEP 一直单步之后 到达OEP



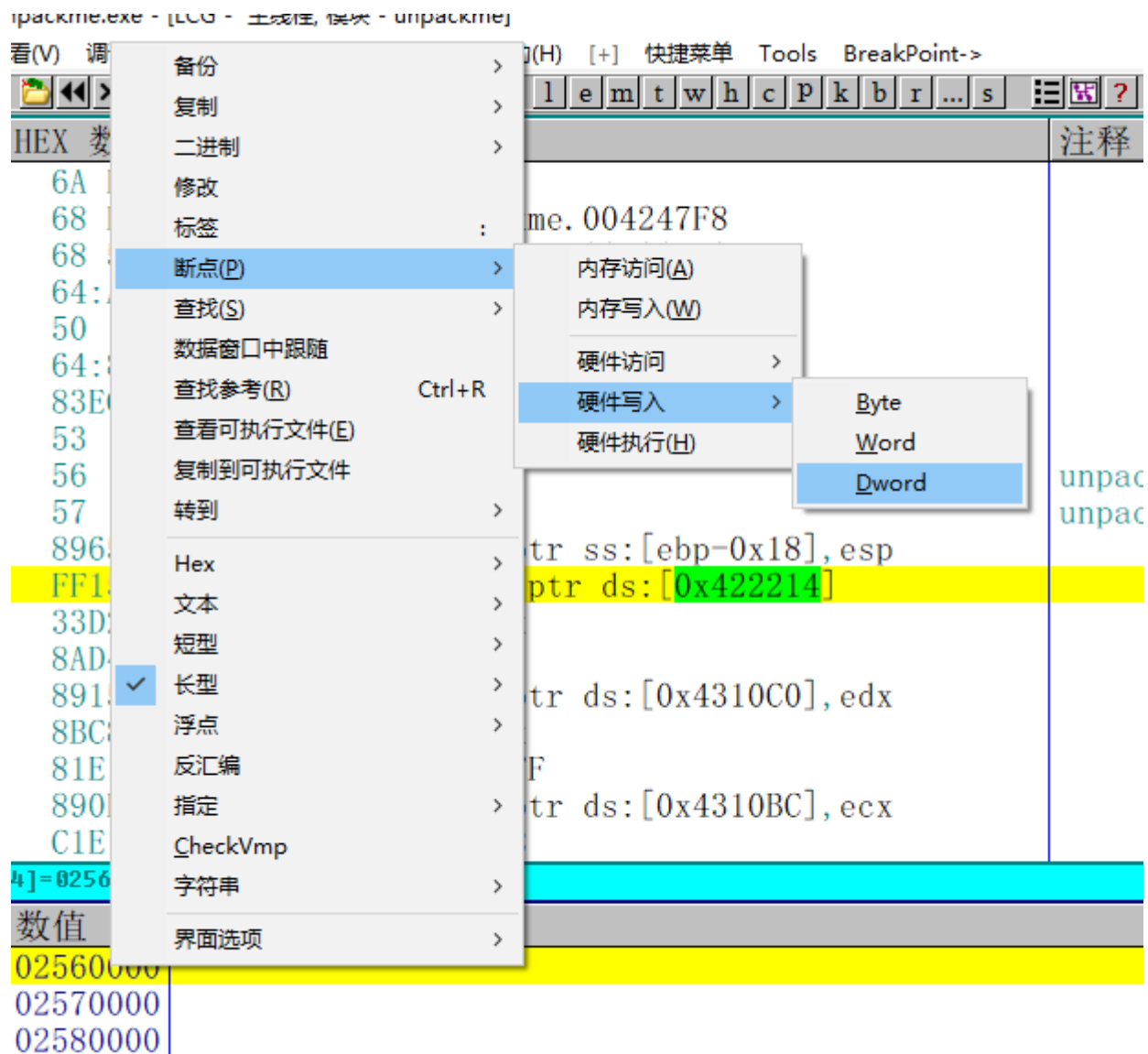
在入口处 我们发现程序符合VC6的入口特征 但是第一个call并不是GetVersion 查看内存 发现IAT被加密了 这个时候 如果直接dump内存 程序就无法运行 必须先将IAT解密 解密IAT的方法就是找到写入

004094A7	56	push esi	unpackme.<ModuleEntryPoin
004094A8	57	push edi	unpackme.<ModuleEntryPoin
004094A9	8965 E8	mov dword ptr ss:[ebp-0x18],esp	
004094AC	FF15 14224200	call dword ptr ds:[0x422214]	
004094B2	33D2	xor edx,edx	
004094B4	8AD4	mov dl,ah	
004094B6	8915 C0104300	mov dword ptr ds:[0x4310C0],edx	
004094BC	8BC8	mov ecx,eax	
004094BE	81E1 FF000000	and ecx,0xFF	
004094C4	890D BC104300	mov dword ptr ds:[0x4310BC],ecx	
004094CA	C1E1 08	shl ecx,0x8	
ds:[00422214]=02560000			
地址	数值	注释	地址
00422214	02560000		001
00422218	02570000		001
0042221C	02580000		001
00422220	02590000		001
00422224	025A0000		001
00422228	025B0000		001
0042222C	025C0000		001
00422230	025D0000		001

IAT的位置 然后将函数原本未加密的地址写回去 即可完成解密

解密IAT

所以 我们在IAT处下硬件写入断点 找到写入IAT的地方



重启程序 一直按F9找到加密IAT的地方

地址	HEX 数据	反汇编	注释
004385CF	8945 DB	mov dword ptr ss:[ebp-0x25],eax	
004385D2	FF15 B4924300	call dword ptr ds:[0x4392B4]	kernel32.VirtualAlloc
004385D8	f30f6f45 d0	movdqu xmm0,dword ptr ss:[ebp-0x30]	
004385DD	8B55 CC	mov edx,dword ptr ss:[ebp-0x34]	unpackme. 00400000
004385E0	f30f7f00	movdqu dqword ptr ds:[eax],xmm0	
004385E4	f30f6f45 e0	movdqu xmm0,dword ptr ss:[ebp-0x20]	
004385E9	f30f7f40 10	movdqu dqword ptr ds:[eax+0x10],xmm0	
004385EE	8907	mov dword ptr ds:[edi],eax	
004385F0	8B4E 04	mov ecx,dword ptr ds:[esi+0x4]	
004385F3	83C6 04	add esi,0x4	
004385F6	8BFE	mov edi,esi	unpackme. 00422214
004385F8	85C9	test ecx,ecx	
004385FA	^ 0F85 72FFFFFF	jnz unpackme. 00438572	
00438600	8B75 C4	mov esi,dword ptr ss:[ebp-0x3C]	unpackme. 00428C10
00438603	83C6 14	add esi,0x14	
00438606	8975 C4	mov dword ptr ss:[ebp-0x3C],esi	unpackme. 00422214
00438609	837E F0 00	cmp dword ptr ds:[esi-0x10],0x0	
0043860D	^ 0F85 3DFFFFFF	jnz unpackme. 00438550	
00438613	8B75 CC	mov esi,dword ptr ss:[ebp-0x34]	unpackme. 00400000

一般加密IAT的地方都会有一个规律 `mov dword ptr ds:[edi],eax` 都是类似这样的格式 找到IAT之后 一般有两种处理方式 一是重启程序 在 `LoadLibraryA/W GetProcAddress` 下API断点 二是继续单步跟踪 找到IAT原本的地址 因为一般加密IAT的地方是个循环 这里我们采取第二种方法 单步几次之后 我们发现程序使用 `GetProcAddress` 获取到了函数地址 之后对这个地址进行了异或

解密破解 - unpackme.exe - [L]C - 主线程, 模块 - unpackme.exe					
文件(F) 查看(V) 调试(D) 插件(P) 选项(T) 窗口(W) 帮助(H) [+] 快捷菜单 Tools BreakPoint->					
[X] [X]					

之后又申请了一块空间 将加密后的地址填充到这块空间里

地址	HEX 数据	反汇编	注释
004385B5	C645 EF C3	mov byte ptr ss:[ebp-0x11],0xC3	
004385B9	FF15 CC924300	call dword ptr ds:[0x4392CC]	kernel32.GetProcAddr
004385BF	6A 40	push 0x40	
004385C1	68 00300000	push 0x3000	
004385C6	6A 20	push 0x20	
004385C8	35 15151515	xor eax,0x15151515	
004385CD	6A 00	push 0x0	
004385CF	8945 DB	mov dword ptr ss:[ebp-0x25],eax	
004385D2	FF15 B4924300	call dword ptr ds:[0x4392B4]	kernel32.VirtualAlloc
004385D8	f30f6f45 d0	movdqu xmm0,dqword ptr ss:[ebp-0x30]	
004385DD	8B55 CC	mov edx,dword ptr ss:[ebp-0x34]	unpackme.00400000
004385E0	f30f7f00	movdqu dqword ptr ds:[eax],xmm0	
004385E4	f30f6f45 e0	movdqu xmm0,dqword ptr ss:[ebp-0x20]	
004385E9	f30f7f40 10	movdqu dqword ptr ds:[eax+0x10],xmm0	
004385EE	8907	mov dword ptr ds:[edi],eax	
004385F0	8B4E 04	mov ecx,dword ptr ds:[esi+0x4]	
004385F3	83C6 04	add esi,0x4	
004385F6	8BFE	mov edi,esi	unpackme.0042221C
004385F8	85C9	test ecx,ecx	
eax=02580000			

我们只需要将加密前的IAT填充回去就完成了解密

地址	HEX 数据	反汇编	注释
004385B5	C645 EF C3	mov byte ptr ss:[ebp-0x11],0xC3	
004385B9	FF15 CC924300	call dword ptr ds:[0x4392CC]	
004385BF	6A 40	push 0x40	
004385C1	68 00300000	push 0x3000	
004385C6	6A 20	push 0x20	
004385C8	35 15151515	xor eax,0x15151515	
004385CD	6A 00	push 0x0	
004385CF	8945 DB	mov dword ptr ss:[ebp-0x25],eax	
004385D2	FF15 B4924300	call dword ptr ds:[0x4392B4]	
004385D8	f30f6f45 d0	movdqu xmm0,dqword ptr ss:[ebp-0x30]	
004385DD	8B55 CC	mov edx,dword ptr ss:[ebp-0x34]	
004385E0	f30f7f00	movdqu dqword ptr ds:[eax],xmm0	
004385E4	f30f6f45 e0	movdqu xmm0,dqword ptr ss:[ebp-0x20]	
004385E9	f30f7f40 10	movdqu dqword ptr ds:[eax+0x10],xmm0	
004385EE	8907	mov dword ptr ds:[edi],eax	将这一句复制到xor eax,0x15151515处,然后nop掉
004385F0	8B4E 04	mov ecx,dword ptr ds:[esi+0x4]	
004385F3	83C6 04	add esi,0x4	
004385F6	8BFE	mov edi,esi	unpackme.<ModuleEntryPoint>
004385F8	85C9	test ecx,ecx	unpackme.<ModuleEntryPoint>

修改后的代码如下

吾爱破解 - unpackme.exe - [LCG - 主线程, 模块 - unpackme]			
文件(F) 查看(V) 调试(D) 插件(P) 选项(T) 窗口(W) 帮助(H) [+] 快捷菜单 Tools BreakPoint->			
暂停			
地址	HEX 数据	反汇编	注释
004385BF	6A 40	push 0x40	
004385C1	68 00300000	push 0x3000	
004385C6	6A 20	push 0x20	
004385C8	8907	mov dword ptr ds:[edi],eax	
004385CA	90	nop	
004385CB	90	nop	
004385CC	90	nop	
004385CD	6A 00	push 0x0	
004385CF	8945 DB	mov dword ptr ss:[ebp-0x25],eax	
004385D2	FF15 B4924300	call dword ptr ds:[0x4392B4]	
004385D8	f30f6f45 d0	movdqu xmm0,dqword ptr ss:[ebp-0x30]	
004385DD	8B55 CC	mov edx,dword ptr ss:[ebp-0x34]	
004385E0	f30f7f00	movdqu dqword ptr ds:[eax],xmm0	
004385E4	f30f6f45 e0	movdqu xmm0,dqword ptr ss:[ebp-0x20]	
004385E9	f30f7f40 10	movdqu dqword ptr ds:[eax+0x10],xmm0	
004385EE	90	nop	
004385EF	90	nop	
004385F0	8B4E 04	mov ecx,dword ptr ds:[esi+0x4]	
004385F3	83C6 04	add esi,0x4	

之后单步到OEP处

吾爱破解 - unpackme.exe - [LCG - 主线程, 模块 - unpackme]			
文件(F) 查看(V) 调试(D) 插件(P) 选项(T) 窗口(W) 帮助(H) [+] 快捷菜单 Tools BreakPoint->			
暂停			
l e m t w h c p k b r ... s			
地址 HEX 数据 反汇编 注释			
00409486	55	push ebp	
00409487	8BEC	mov ebp,esp	
00409489	6A FF	push -0x1	
0040948B	68 F8474200	push unpackme.004247F8	
00409490	68 50CB4000	push unpackme.0040CB50	
00409495	64:A1 00000000	mov eax,dword ptr fs:[0]	
0040949B	50	push eax	
0040949C	64:8925 00000000	mov dword ptr fs:[0],esp	
004094A3	83EC 58	sub esp,0x58	
004094A6	53	push ebx	
004094A7	56	push esi	unpackme.<ModuleEntryPoint>
004094A8	57	push edi	unpackme.<ModuleEntryPoint>
004094A9	8965 E8	mov dword ptr ss:[ebp-0x18],esp	
004094AC	FF15 14224200	call dword ptr ds:[0x422214]	kernel32.GetVersion IAT已经修复
004094B2	33D2	xor edx,edx	
004094B4	8AD4	mov dl,ah	
004094B6	8915 C0104300	mov dword ptr ds:[0x4310C0],edx	
004094BC	8BC8	mov ecx,eax	
004094BE	81E1 FF000000	and ecx,0xFF	
ebp=0019FF78			
地址 数值 注释			
00422000	75E8D1D0	advapi32.RegCloseKey	
	0019FF64	00000000	

可以发现 IAT解密已经完成 然后dump文件 修复IAT即可

在手动修复IAT的时候要注意 当单步跟踪找到了写入IAT和获取IAT地址时 应该重新加载程序 在程序写入IAT之前完成对代码的修改 因为如果在单步跟踪的时候直接修改 这个时候程序已经加密了部分的代码 此时再去修复IAT就会出问题 但是如果用写脚本的方式就不需要担心这种问题

修复导入表



然而 当我们使用ImportREC修复导入表时 我们发现 软件并没有能完整的识别所有的IAT信息 只搜索到两个模块的导入信息 在遇到这种情况一般采取的方法有两种 一种是将ImportREC自动分析的RVA取整 Size增加到1000 这样也能比较完整地分析全IAT表 只是会分析出很多无用函数 还有一种是手动查看IAT区域的基址和大小 然后填入ImportREC的RVA和Size中 这里我们采取第二种方法 先在OD中观察IAT表的起始位置 是422000

地址	数值	注释
00422000	75E8D1D0	advapi32.RegCloseKey
00422004	6F63FCA0	apphelp.6F63FCA0
00422008	6F63FA00	apphelp.6F63FA00
0042200C	6F63F820	apphelp.6F63F820
00422010	00000000	
00422014	80000011	
00422018	00000000	
0042201C	751B4330	gdi32.SetMapMode
00422020	751B4220	gdi32.SetViewportOrgEx
00422024	751B7E60	gdi32.OffsetViewportOrgEx
00422028	751B4C40	gdi32.SetViewportExtEx
0042202C	751BC790	jmp 到 gdi32ful.ScaleViewportExtEx
00422030	751B4C00	gdi32.SetWindowExtEx
00422034	751BC7B0	jmp 到 gdi32ful.ScaleWindowExtEx
00422038	751B6E50	gdi32.IntersectClipRect
0042203C	751B5B20	gdi32.DeleteObject
00422040	6F604870	apphelp.6F604870
00422044	751BC310	jmp 到 gdi32ful.GetViewportExtEx

然后找到结束位置 422504

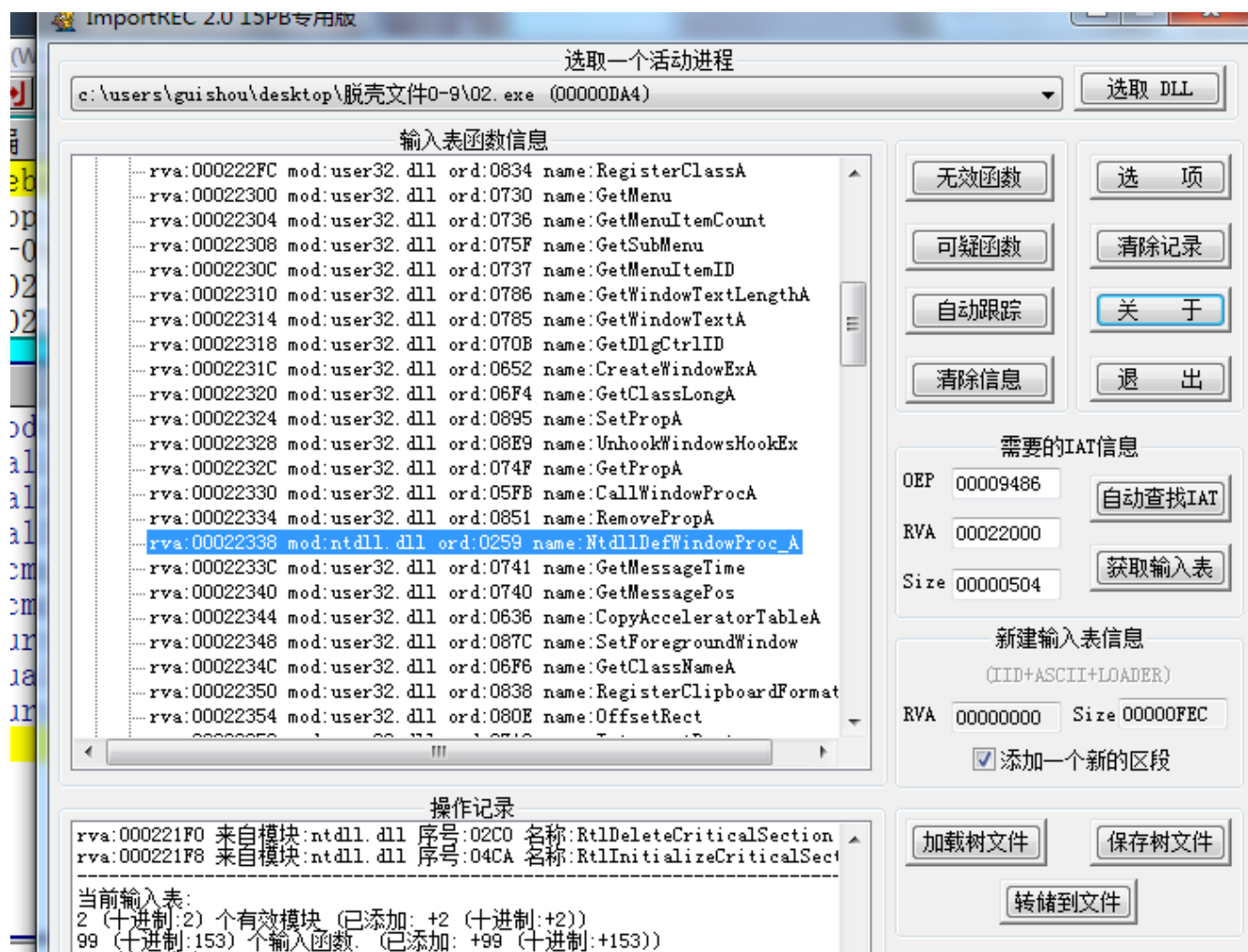
004224D4	7637EC20	combase.CoTaskMemFree	00
004224D8	75FFF010	ole32.CreateILockBytesOnHGlobal	00
004224DC	75FFF180	ole32.StgCreateDocfileOnILockBytes	00
004224E0	75FFF2A0	ole32.StgOpenStorageOnILockBytes	00
004224E4	763CD560	combase.CoGetClassObject	00
004224E8	76368250	combase.CLSIDFromString	00
004224EC	7633EB90	combase.CLSIDFromProgID	00
004224F0	75FF0100	jmp 到 combase.CoRegisterMessageFilter	00
004224F4	7633F0C0	combase.CoRevokeClassObject	00
004224F8	75FE0370	ole32.OleFlushClipboard	00
004224FC	75FDC7B0	ole32.OleIsCurrentClipboard	00
00422500	75FE20C0	ole32.OleUninitialize	00
00422504	00000000		00
00422508	80000008		00
0042250C	00000000		00
00422510	00422830	unpackme.00422830	00
00422514	00422518	unpackme.00422518	00

由此得出IAT表的大小 是504 重新在ImportREC中指定RVA和Size 点击获取输入表

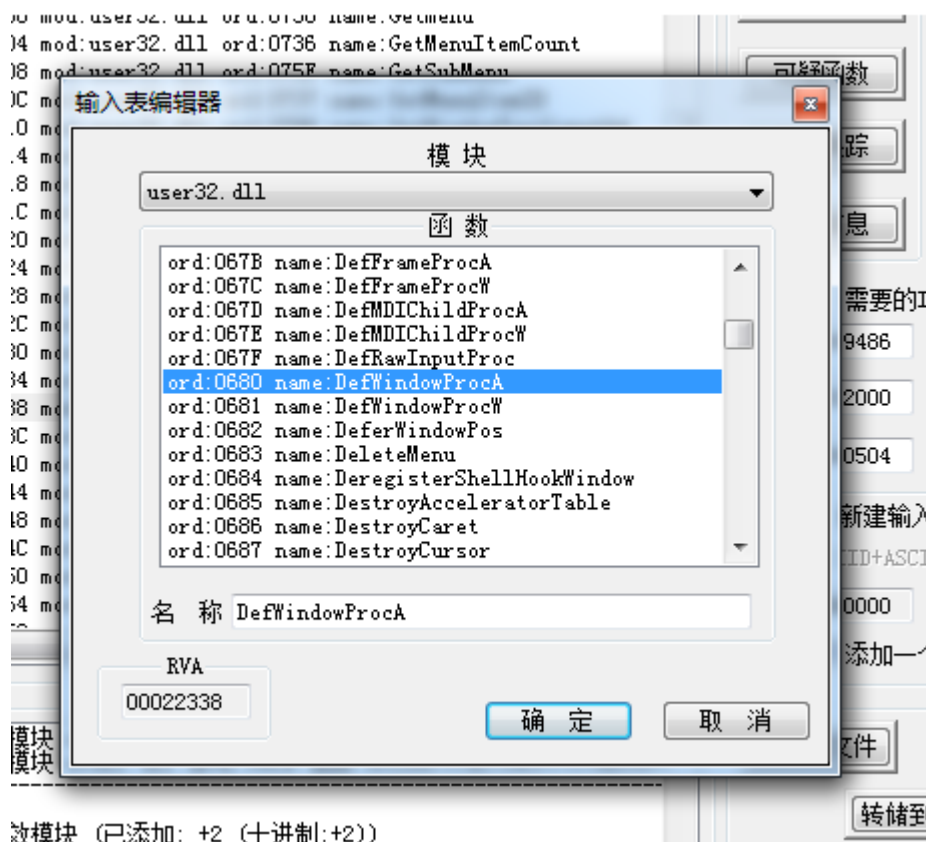


在获取到的函数地址中 会有一部分无效的 一般无效的看能否修复 如果可以修复就修复 如果不能修复就剪切掉 在分析的无效函数中 有一种转发函数 被分析成了无效函数 如图中的

NtdllDefwindowProc_A

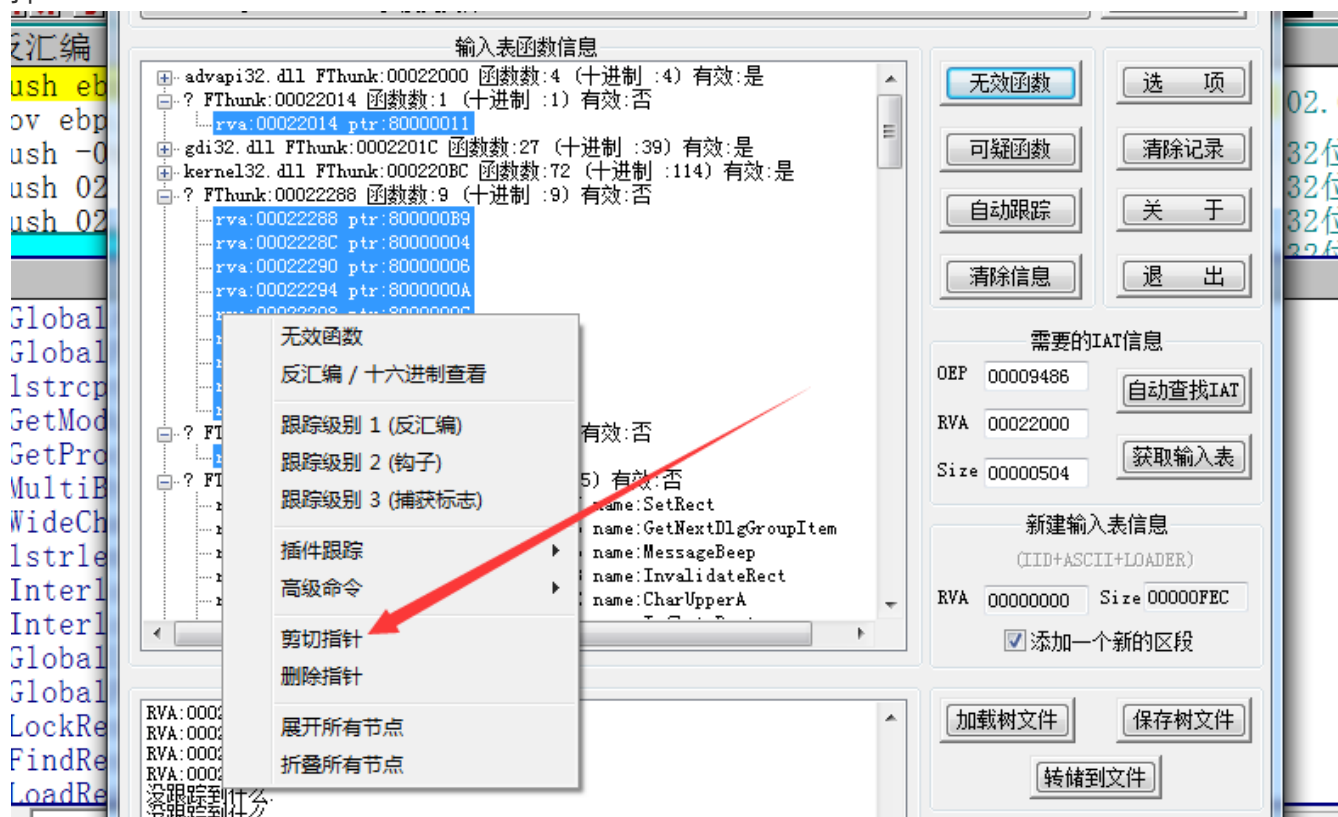


这种无效函数的成因其实是函数被转发了 上面的函数在user32中的函数名的 DefWindowProcA 而在 user32模块中这个函数地址空间并没有实际的代码 而只是一个函数名 类似的函数还有几个 这种函数可以使用ImportREC修复 双击函数名 然后先选择模块 再填入函数名

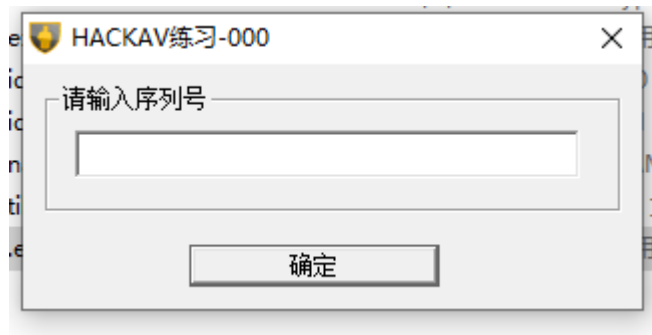


如果还有无效函数 就右键剪切

无效模块 (已添加: +2 (十进制: +2))
掉



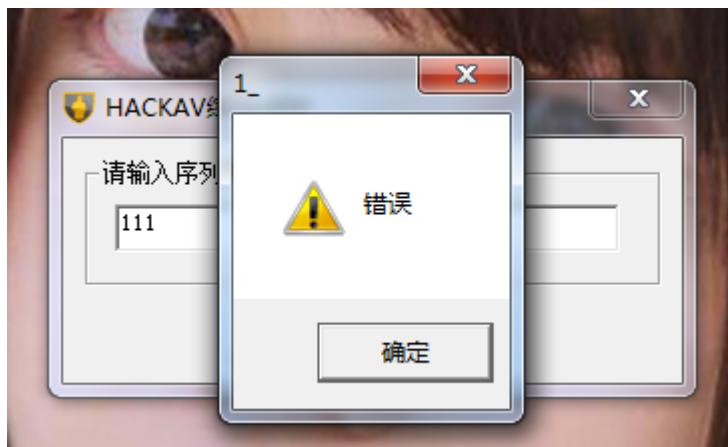
当没有无效函数的时候 就可以转储文件了, 程序正常运行



注意，我的运行环境是W7 64位，脱壳后的程序放到我的W10物理机上的跑不起来的

破解Crackme

既然是个Crackme，就顺手把他干掉吧



根据字符串提示找到来自401557的地址

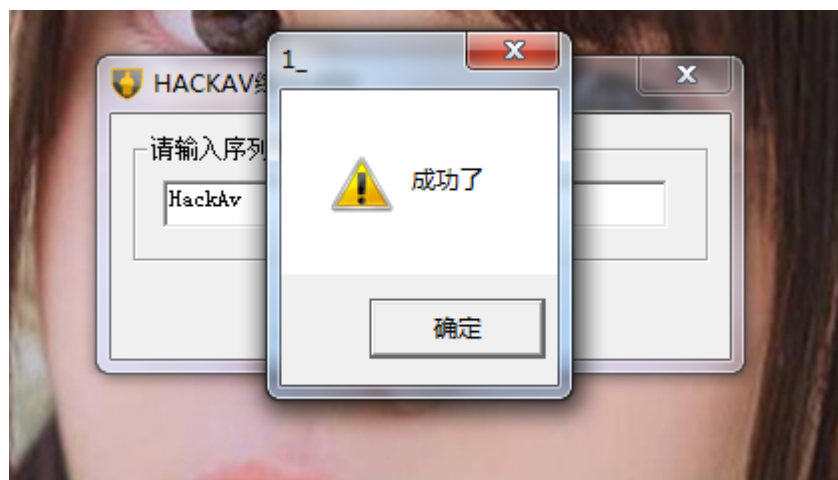
地址	HEX	数据	反汇编	注释
00401567	.	E8 98850100	call 1_.00419B04	
0040156C	.	5F	pop edi	kernel32.773C33AA
0040156D	.	5E	pop esi	kernel32.773C33AA
0040156E	.	5B	pop ebx	kernel32.773C33AA
0040156F	.	C3	ret	
00401570	>	68 C0B04200	push 1_.0042B0C0	错误
00401575	.	E8 2DB80100	call 1_.0041CDA7	
0040157A	.	6A 00	push 0x0	
0040157C	.	8BCF	mov ecx,edi	
0040157E	.	E8 81850100	call 1_.00419B04	
0042B0C0=1_.0042B0C0 (ASCII "错误")				
跳转来自 00401557				
地址	数值	注释	地址	数值
00422000	779C461D	advapi32.RegCloseKey	0018FF8C	773C33AA

地址	HEX 数据	反汇编	注释
00401548	> 33C0	xor eax,eax	kernel32.BaseThreadInitThun
0040154A	.. EB 05	jmp short 1_.00401551	kernel32.BaseThreadInitThun
0040154C	> 1BC0	sbb eax,eax	kernel32.BaseThreadInitThun
0040154E	. 83D8 FF	sbb eax,-0x1	kernel32.BaseThreadInitThun
00401551	> 85C0	test eax,eax	kernel32.BaseThreadInitThun
00401553	. 6A 00	push 0x0	
00401555	. 6A 00	push 0x0	
00401557	.. 75 17	jnz short 1_.00401570	
00401559	. 68 C8B04200	push 1_.0042B0C8	成功了
0040155E	. E8 44B80100	call 1_.0041CDA7	
00401563	. 6A 00	push 0x0	
00401565	. 8BCF	mov ecx,edi	
00401567	. E8 98850100	call 1_.00419B04	
0040156C	. 5F	pop edi	kernel32.773C33AA

然后就来到了成功的地方

址	HEX 数据	反汇编	注释
401515	. 6A 01	push 0x1	
401517	. E8 E8850100	call 1_.00419B04	
40151C	. 8B77 5C	mov esi,dword ptr ds:[edi+0x5C]	
40151F	. B8 D0B04200	mov eax,1_.0042B0D0	HackAv
401524	> 8A10	mov dl,byte ptr ds:[eax]	
401526	. 8A1E	mov bl,byte ptr ds:[esi]	
401528	. 8ACA	mov cl,dl	
40152A	. 3AD3	cmp dl,bl	
40152C	.. 75 1E	jnz short 1_.0040154C	
40152E	. 84C9	test cl,cl	
401530	.. 74 16	je short 1_.00401548	
401532	. 8A50 01	mov dl,byte ptr ds:[eax+0x1]	
401535	. 8A5E 01	mov bl,byte ptr ds:[esi+0x1]	
401538	. 8ACA	mov cl,dl	
40153A	. 3AD3	cmp dl,bl	
40153C	.. 75 0E	jnz short 1_.0040154C	

接着往上翻，发现这里有一个HackAv,直接输入



提示成功，破解完成

需要相关文件可以到我的Github下载:<https://github.com/TonyChen56/Unpack-Practice>