

FSG2.0 脱壳：


一、FSG2.0 壳特征：

反汇编窗口搜索 `jmp dword ptr ds:[ebx+C]` 下断点，且大跳。




二、工具：

虚拟机：win10、win7

1)、X64dbg+Scylla 插件。

 x64dbg 中文版安装程序(Dec 21 2023).rar 中自带 Scylla 转储文件和修复 PE 文件 IAT 工具。

2)、X64dbg（插件 OllyDumpEx）+ Scylla.exe+ ImportREConstructor 1.7.exe。

 x32dbg.exe  Scylla_x86.exe  ImportREC.美化版.exe

3)、PEID：检测加壳情况

二、总结：

步骤：

检测被脱壳文件信息：加壳种类和版本、EP、节情况

选择合适工具脱壳：专脱工具（自动）、OD 转储文件和修复 IAT（手工）

手工脱壳：找 OEP+修复 IAT

工具原理：

动态调试器：OD：通过实时执行和监控程序来调试和分析其行为，帮助逆向工程和故障排查。

转储文件插件：Scylla、OllydumpEX、Ollydump：在运行时从内存中提取解压后的程序代码并保存为 PE 文件。

修复 PE 文件插件：Scylla、ImportREConstructor：重建和修复 PE 文件的导入表和 IAT，确保程序在转储后能正常运行。

问题：

- (1) FSG2.0 加壳程序为什么不能在 win10 系统中运行？
FSG2.0 加壳导致代码揉进了 PE 文件头，且 win10 开启安全保护机制 CFG 不允许 PE 文件头中代码执行，从而导致加壳程序运行失败。
- (2) 如何发现代码揉进了 PE 文件头里面？
可以通过对比程序入口点（EP）地址和 PE 文件头的大小来判断。如果入口

点的地址位于 PE 文件头范围内 (通过 `SizeOfHeaders` 字段判断), 则说明代码可能被揉进了 PE 文件头中。正常情况下, EP 应位于 `.text` 段的开始, 而不是文件头。

(3) 如何找到 OEP?

在反汇编窗口中搜索指令 `jmp dword ptr ds:[ebx+C]` (这类间接跳转指令经常出现在 FSG 2.0 加壳的解压代码中)。在找到该指令后设置断点, 程序运行时会触发跳转, 通常这会跳转到原始入口点 (OEP)。

(4) 判断不止一个 dll, 那还有其他那些 dll 呢?

通过已查出的 dll 函数在 OD 调试器内存窗口中定位函数位置: `ImageBase+偏移`。

(5) 内存窗口中发现 `user32.dll`。为什么没有被 `ImportREConstructor` 工具识别呢?

导入表中存在无效指针或错误数据: 值 `FFFFFFFF` 和 `7FFFFFFF` 可能是无效的地址或占位符, 这表明导入表可能在某个阶段被破坏或被篡改。修复工具在遇到这种无效数据时, 通常会忽略它们。解决办法: 将值 `FFFFFFFF` 和 `7FFFFFFF` 替换为 0。

三、示例演示:

4-1、Ollydbg 动态调试: 找 OEP+dump 文件, Scylla 插件修复 PE

4-2: x64dbg (Scylla 插件)

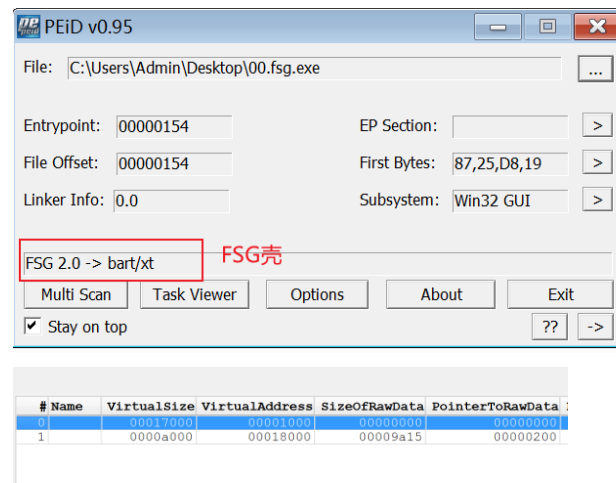
4-3: X64dbg (插件 OllyDumpEx) +ImportREConstructor 1.7。(!!!!

dll 识别不完全, 导入表插入无效值干扰工具识别) ----重点



待脱壳文件：00.fsg.exe

1、看文件信息：fsg 2.0 加壳，2 个节但无节名，由 EP 知代码被“揉进”了 PE 头。



*当 **Entry Point** 为 0x154 时，说明程序的第一条执行指令位于 **PE 文件头** 范围内，而不是在正常的代码段中。这种现象通常表明程序被加壳或混淆，代码被“揉进”了 PE 头，这意味着执行的是壳程序的一部分代码，而不是真正的应用程序代码。

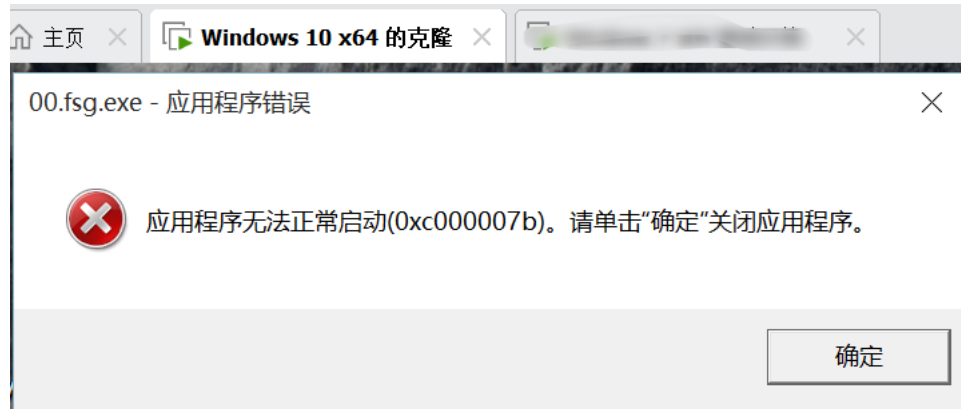
在没有壳或未被混淆的正常 PE 文件中，入口点通常会在 .text 段（代码段）中。比如，常见的 EXE 文件的 **Entry Point** 通常会在 0x1000 或之后的地址，具体位置由节表决定。

PE 文件结构的一般情况：

- **DOS 头** 通常在前 64 字节左右。
- **PE 头** 包含 IMAGE_NT_HEADERS 和 IMAGE_SECTION_HEADER，通常大小在 200 字节左右，具体取决于节的数量。
- **IMAGE_DOS_HEADER (DOS 头)：大小：64 字节**
- **IMAGE_NT_HEADERS (PE 头)：大小：248 字节 (32 位)，272 字节 (64 位)**
- **IMAGE_SECTION_HEADER (节头) 大小：40 字节 (每个节)**

所以，0x154 地址属于 PE 头或其附近的区域，而不是 .text 段中正常的代码地址。

2、win10 中双击运行 00.fsg.exe 失败：win10 默认使用 CFG 保护

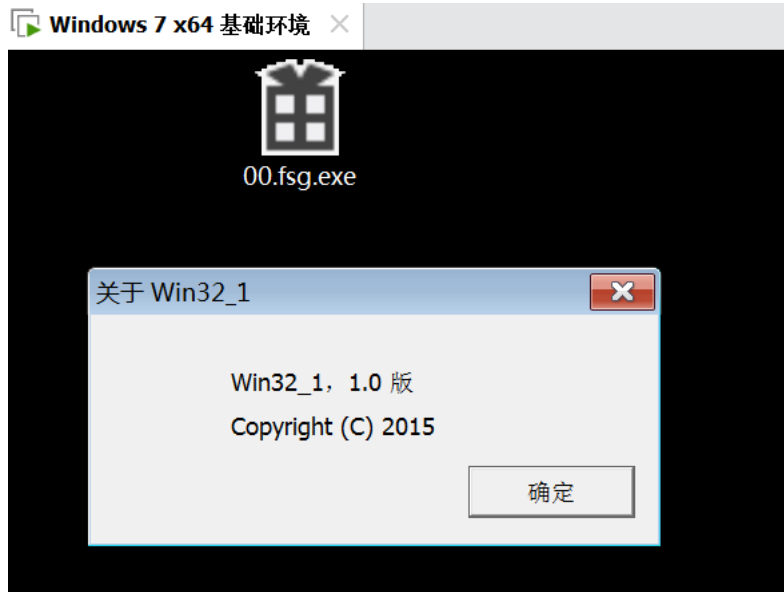


由前面文件信息已知代码被杂糅进了 PE 头，且 win10 系统引入了安全保护机制，使得待脱壳程序在 win10 系统中运行失败。win7 以前旧的系统没有引入此保护机制，可以试着用 win7 运行程序。

Control Flow Guard (CFG) 控制流保护

- **概述：**CFG 是 Windows 10 中引入的一种运行时保护机制，旨在防止恶意软件通过修改代码流控制（如函数指针和返回地址）来劫持程序的执行路径。CFG 会对每个合法的函数调用目标进行检查，防止执行跳转到非法位置（如 PE 头或混入其中的恶意代码）。
- **影响：**如果程序试图执行不在合法调用目标表中的代码（比如 PE 头中的混淆代码），CFG 会阻止执行并终止程序运行。
- **如何生效：**CFG 由编译器支持并通过操作系统进行保护，Windows 10 默认启用该机制。

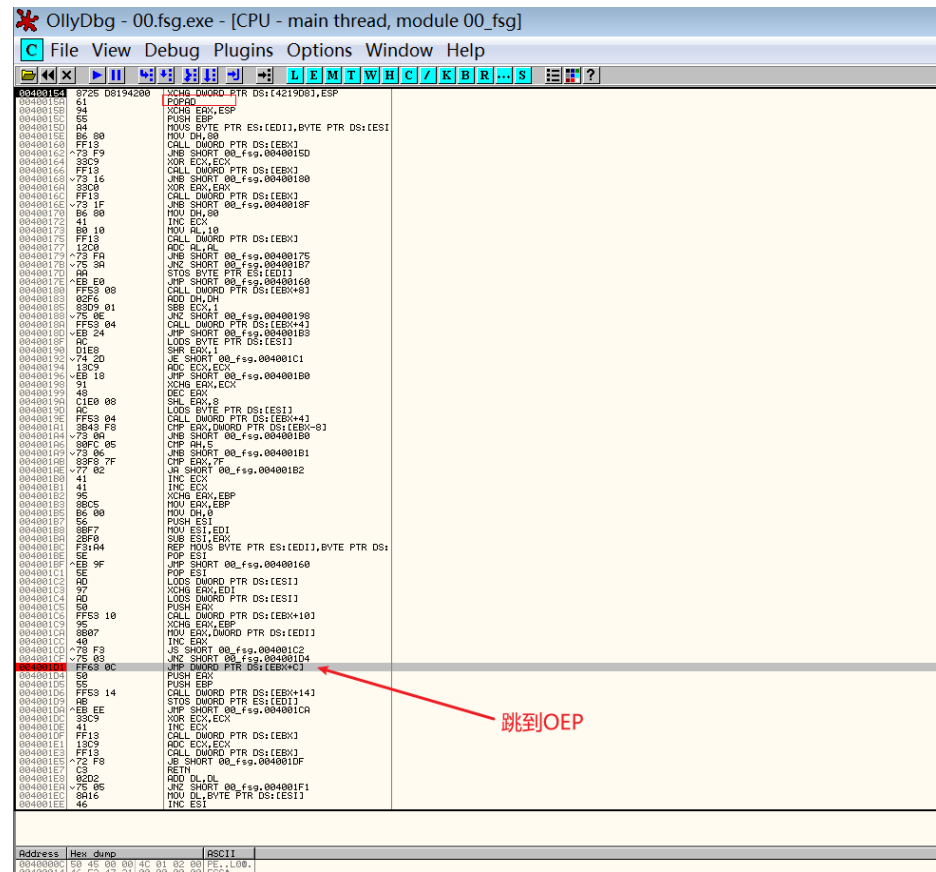
3、win7 中运行 00.fsg.exe 成功：

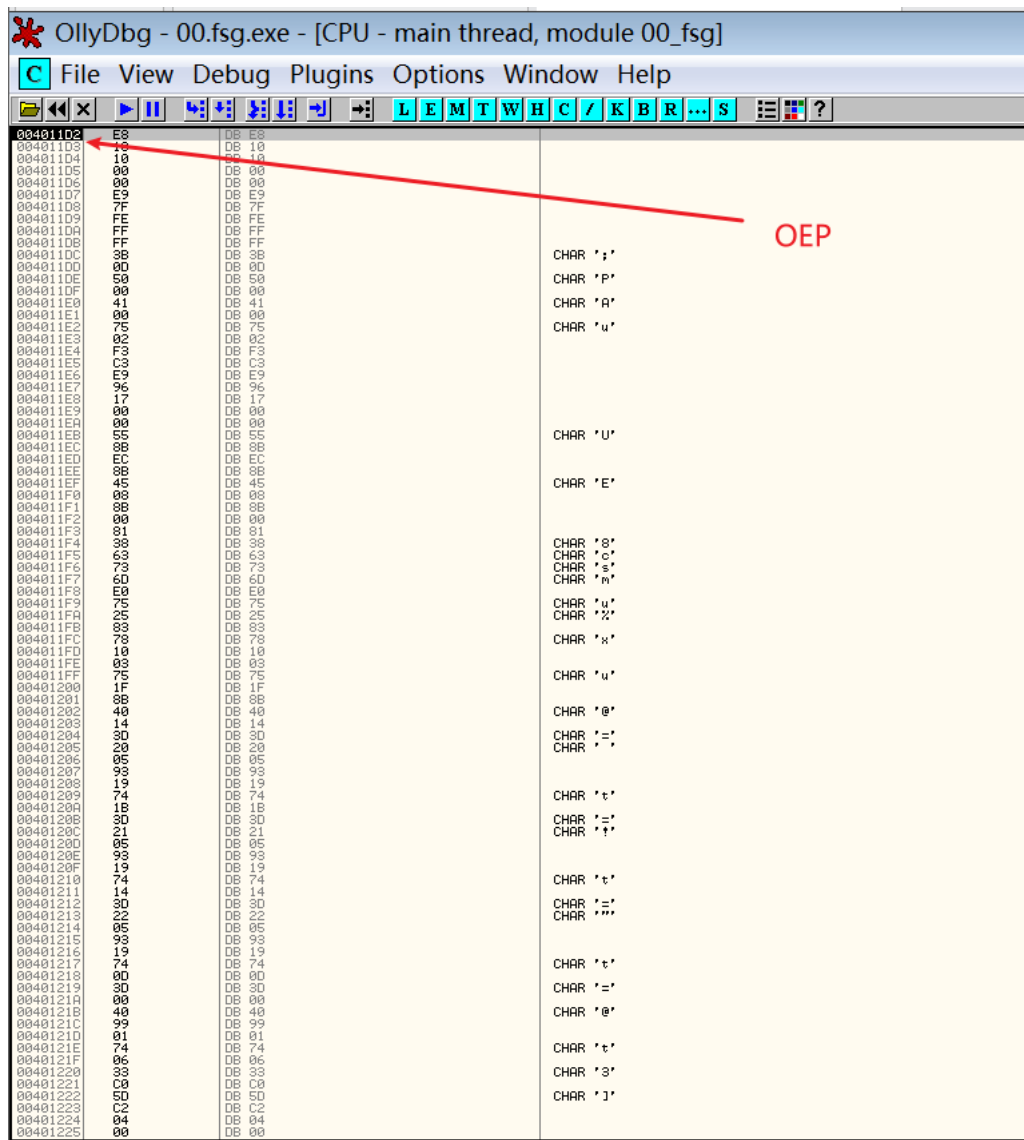


4-1、Ollydbg 动态调试：找 OEP+dump 文件

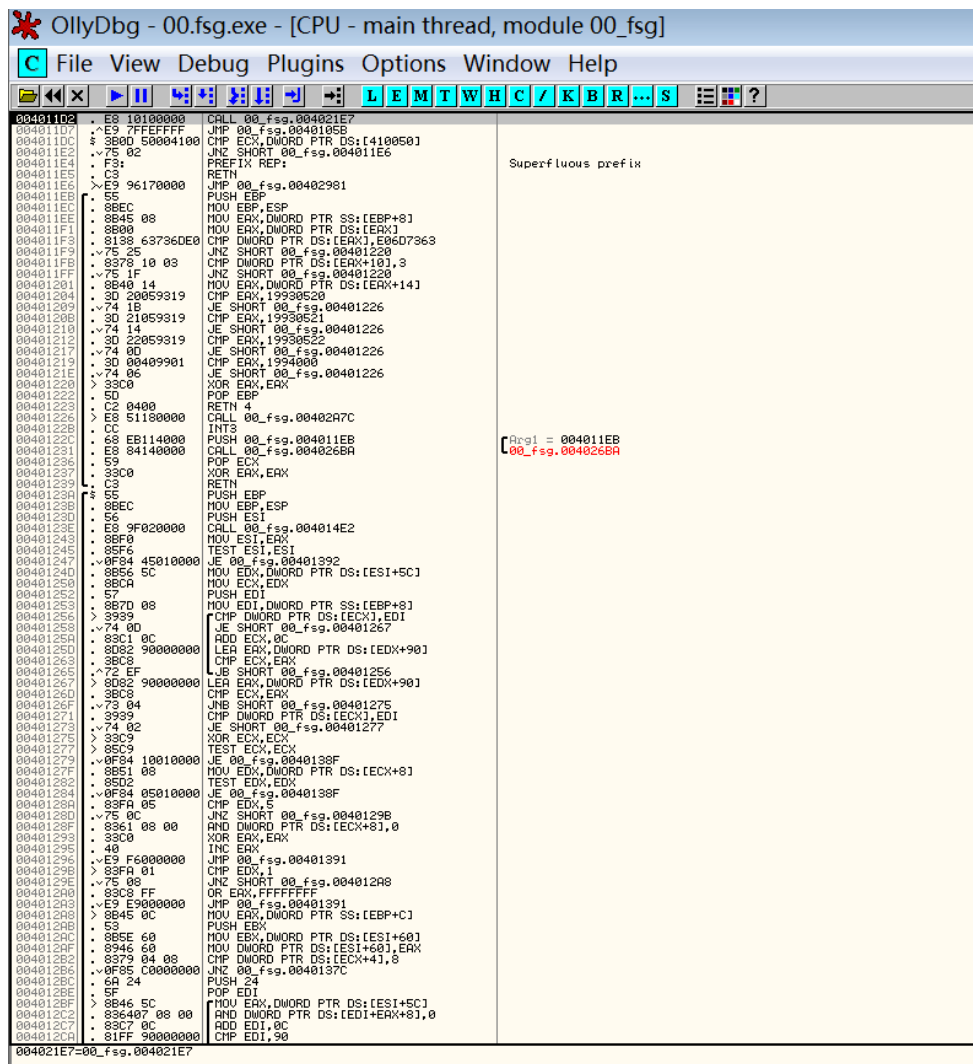
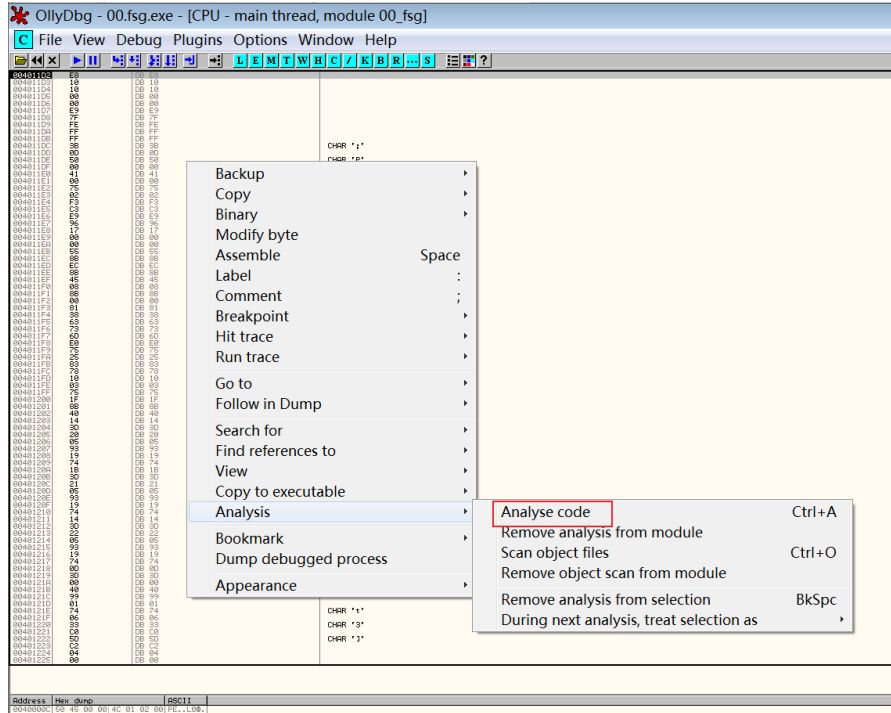
怎么找到 OEP 的？

反汇编窗口搜索 jmp dword ptr ds:[ebx+C] 下断点，且大跳。

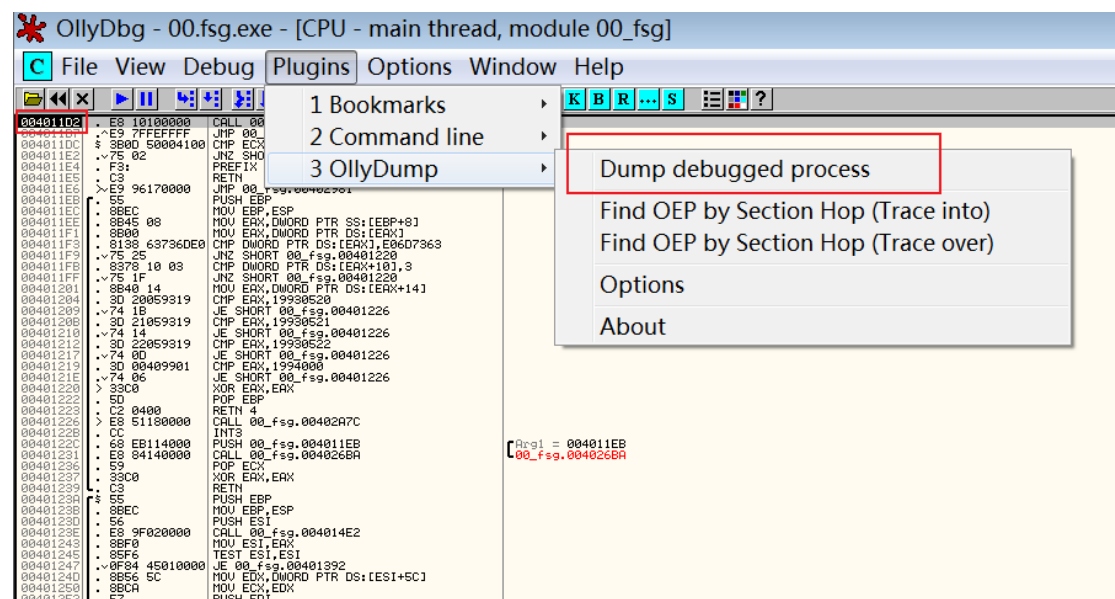




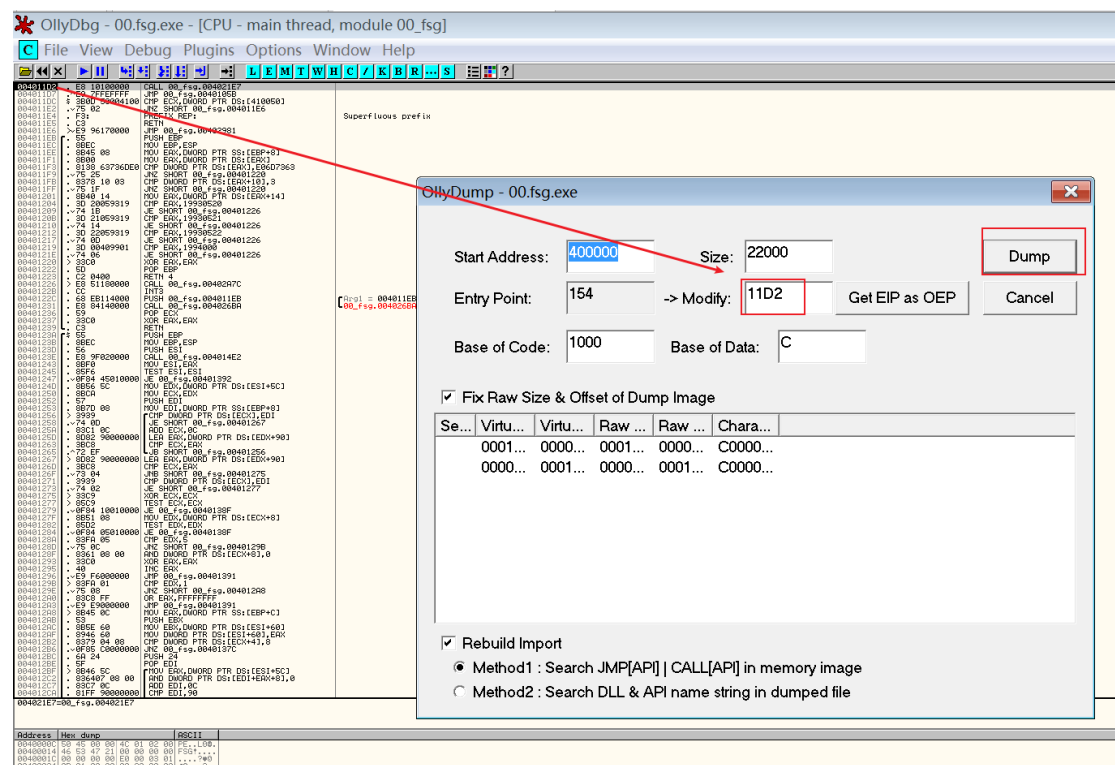
出现未被识别代码，鼠标右键手动分析：



Dump 出文件:



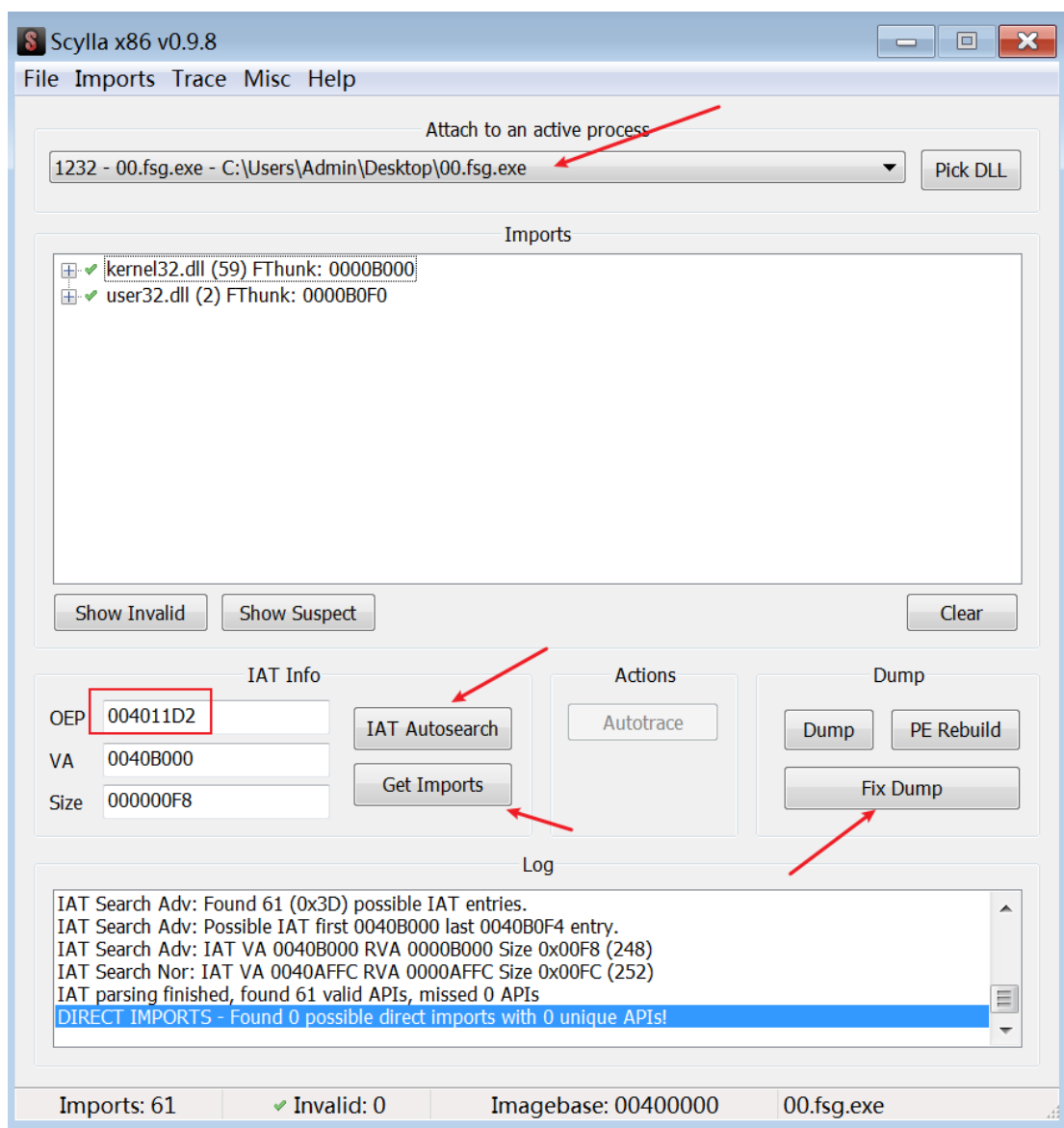
确认 OEP 地址:



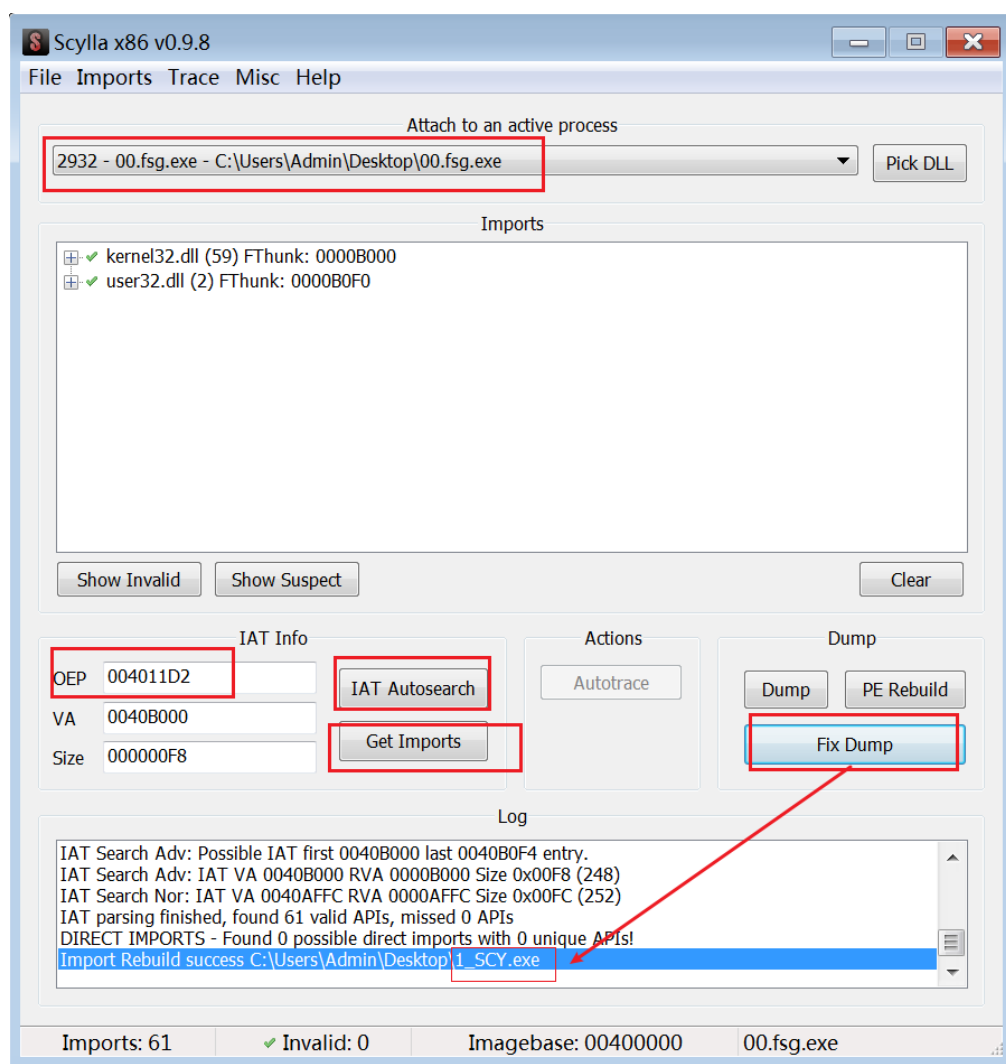
运行 dump 文件: 失败



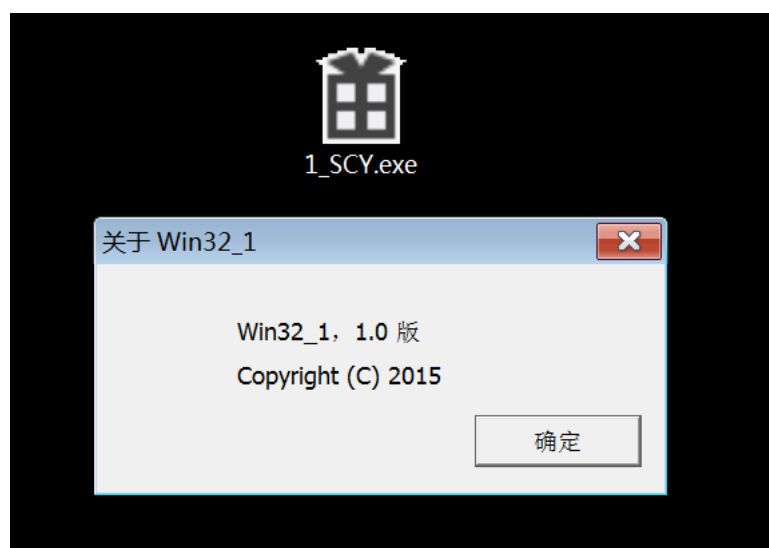
5-1、修复脱壳后的文件 1.exe:

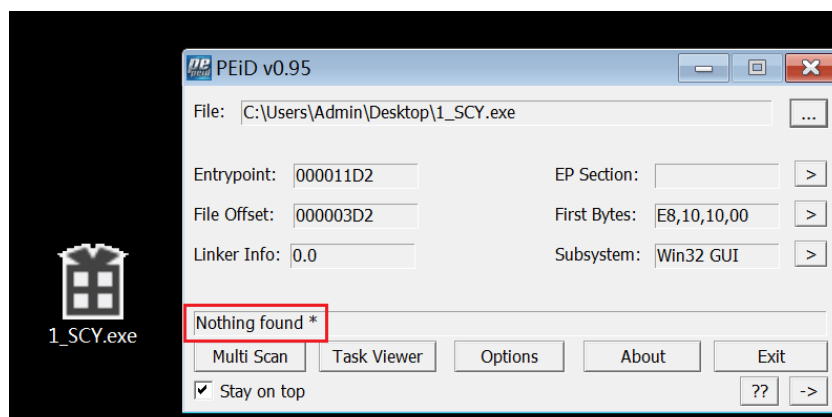


选择 1.exe 进行修复：



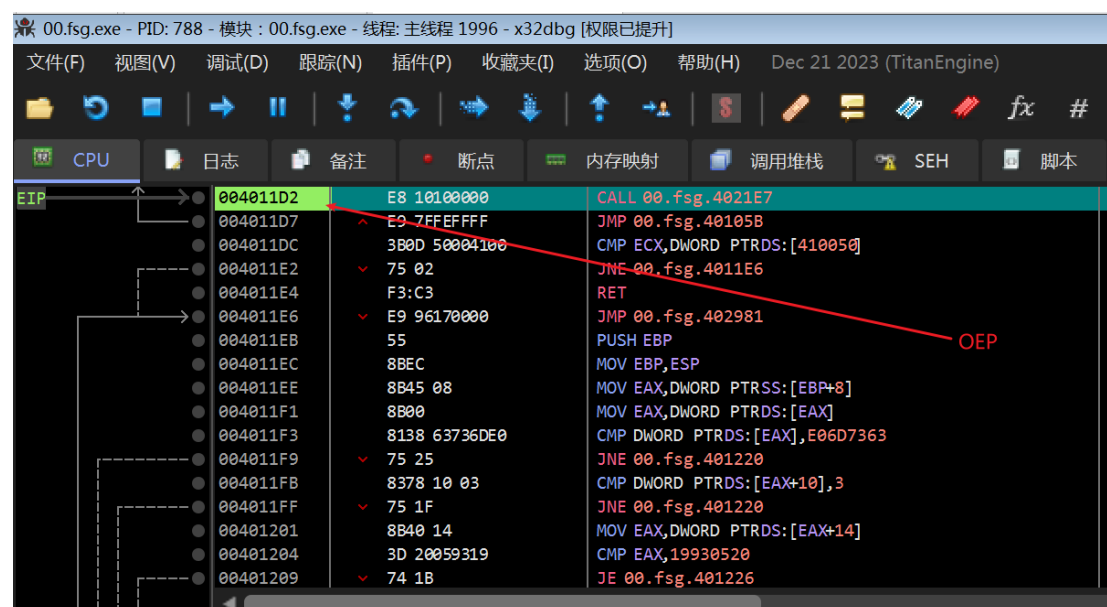
5-3、修复成功：PEID 检测无壳。



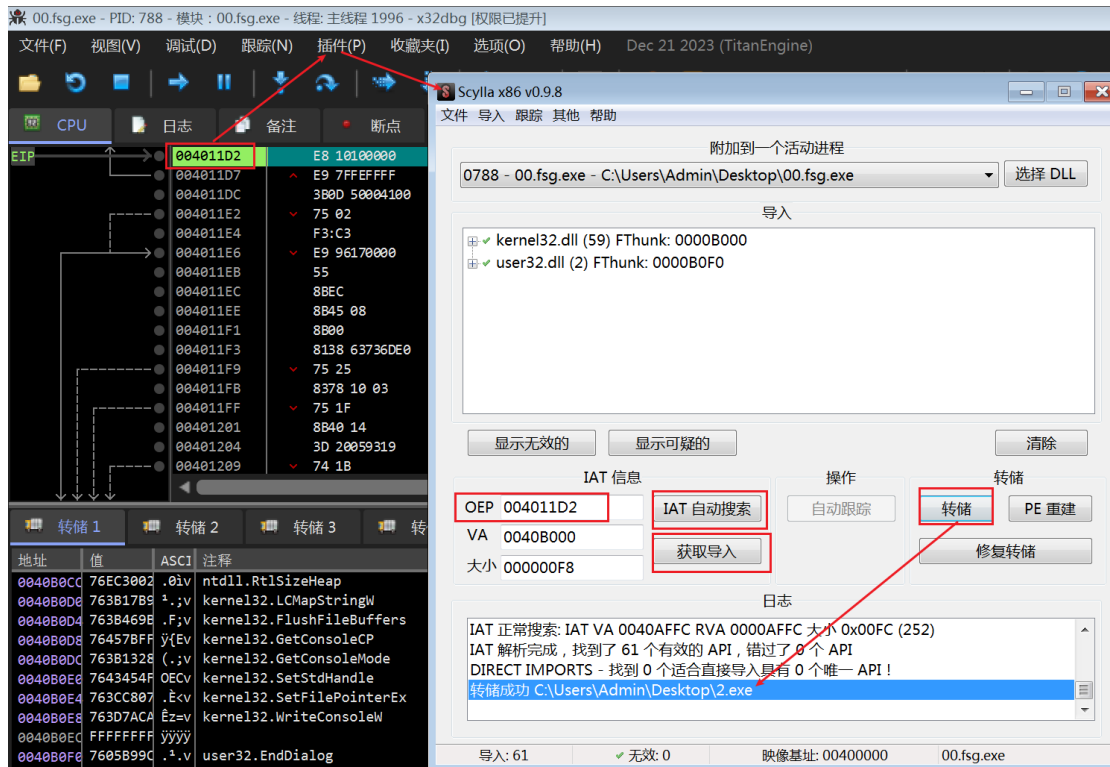


4-2: x64dbg (Scylla 插件)

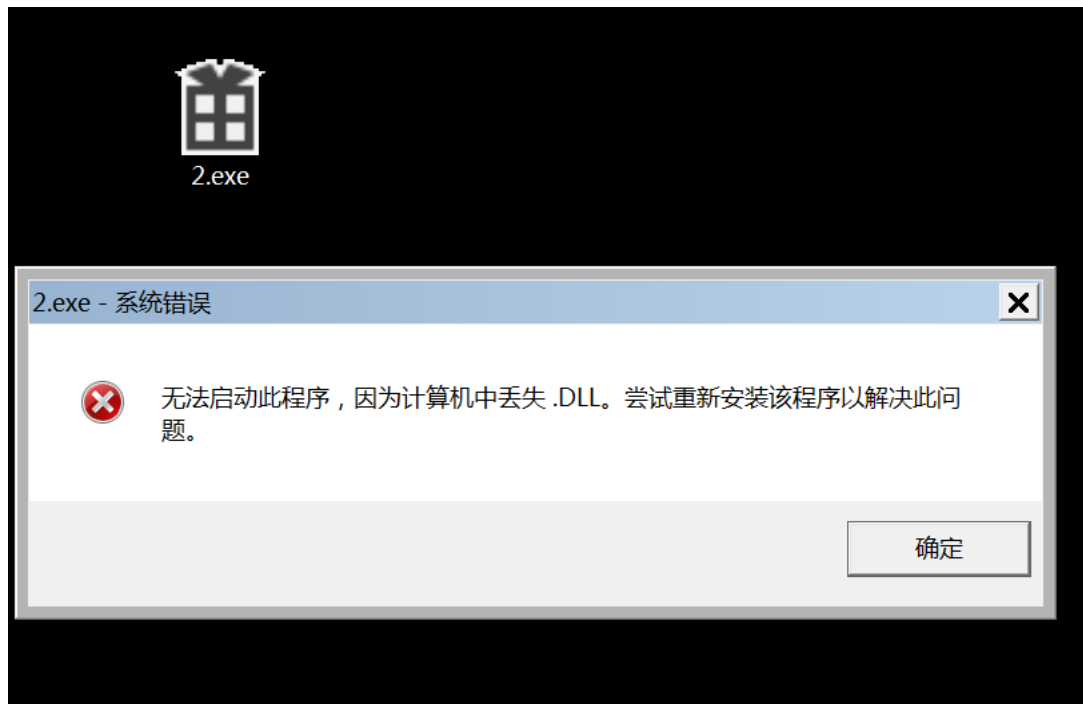
跳到 OEP:



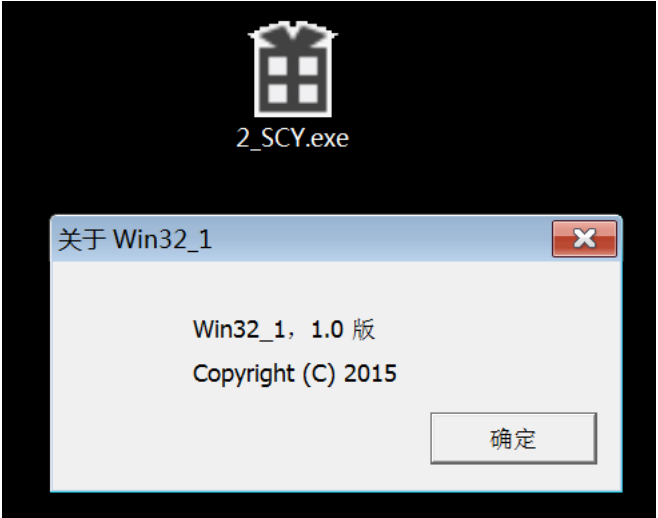
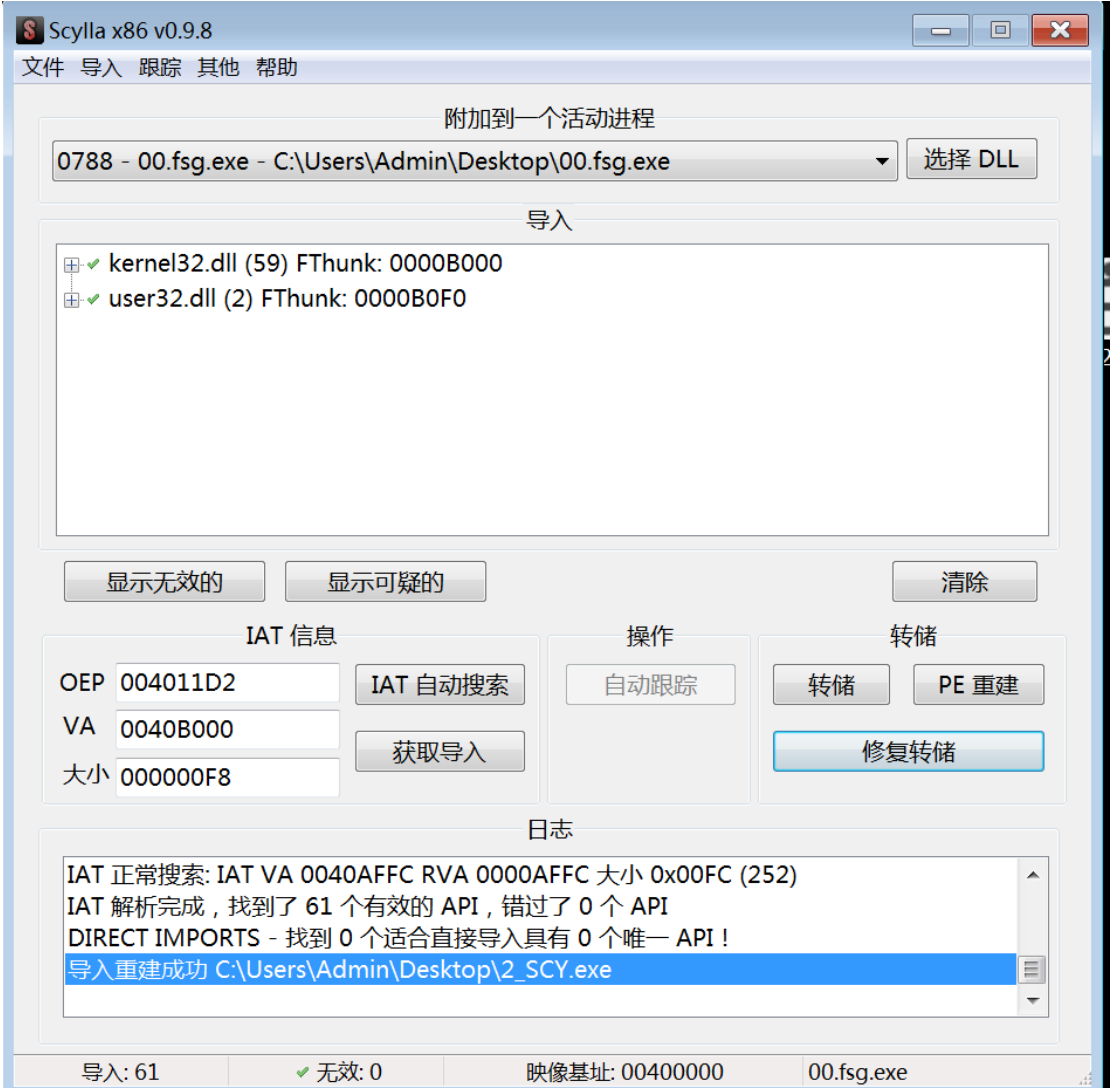
转储成 2.exe 文件:

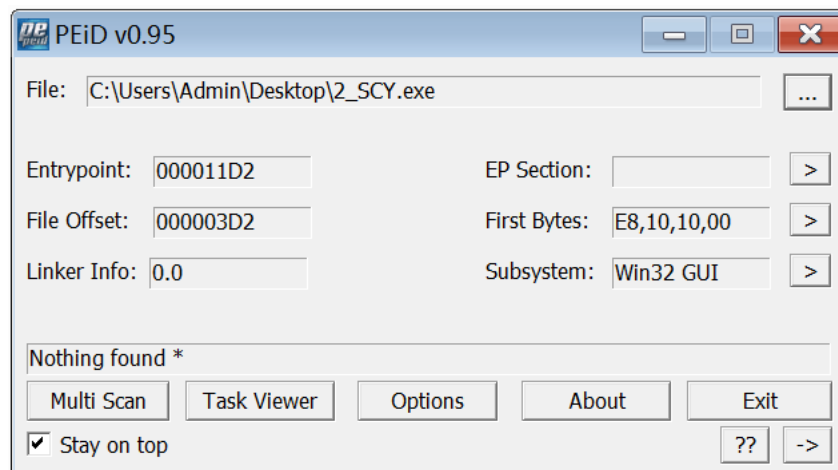


运行 2.exe 文件：失败



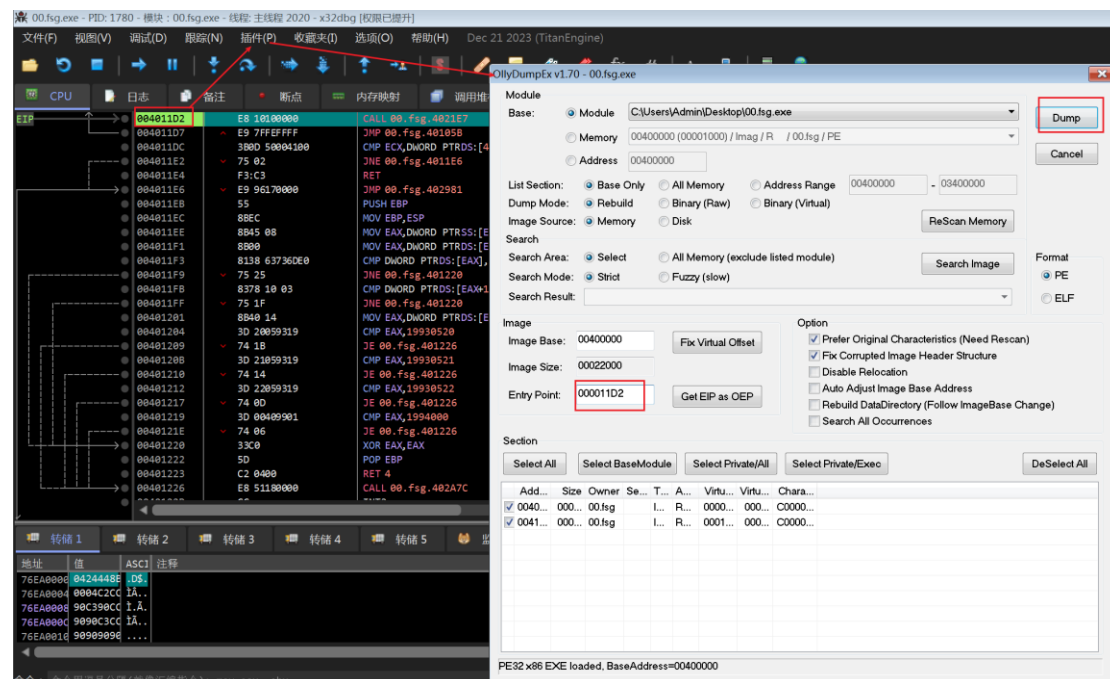
5-2、修复：成功

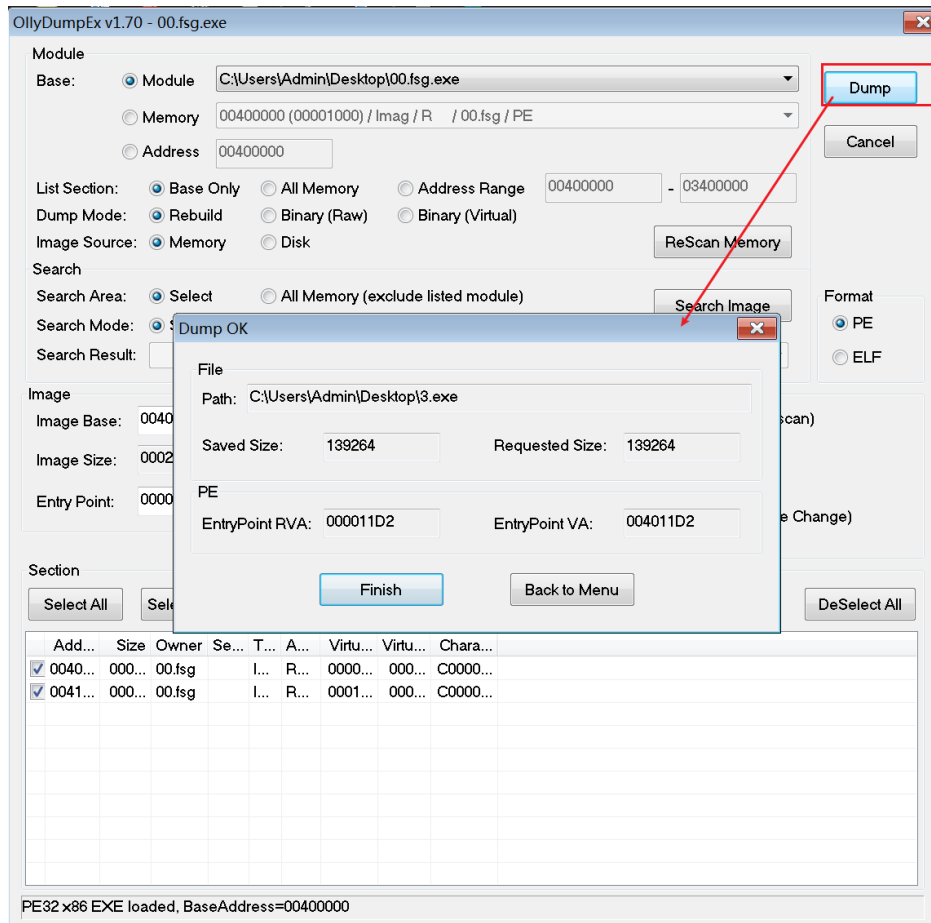




4-3: X64dbg (插件 OllyDumpEx) + ImportREConstructor 1.7.exe。(!!!! dll 识别不完全，导入表插入无效值干扰工具识别)

插件 dump 文件：



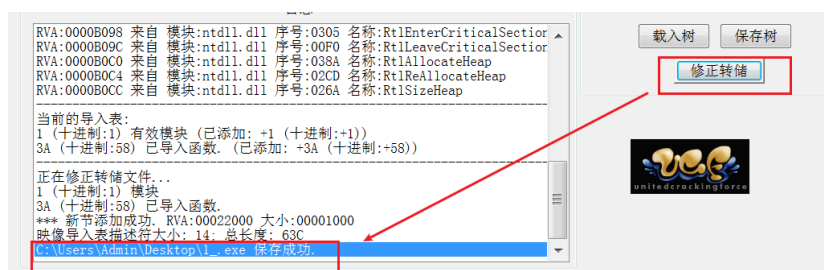
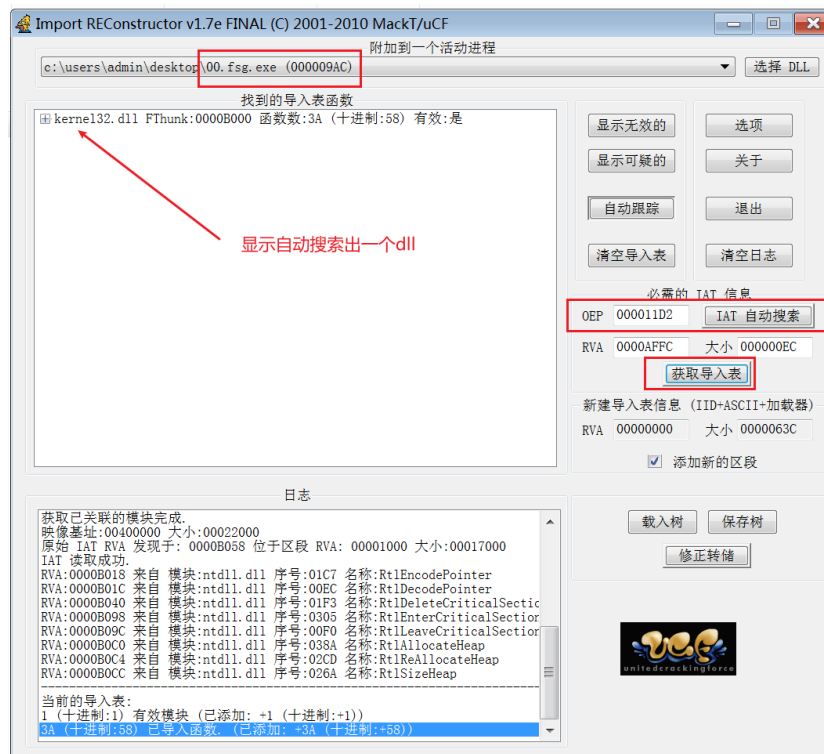


运行 3.exe: 失败

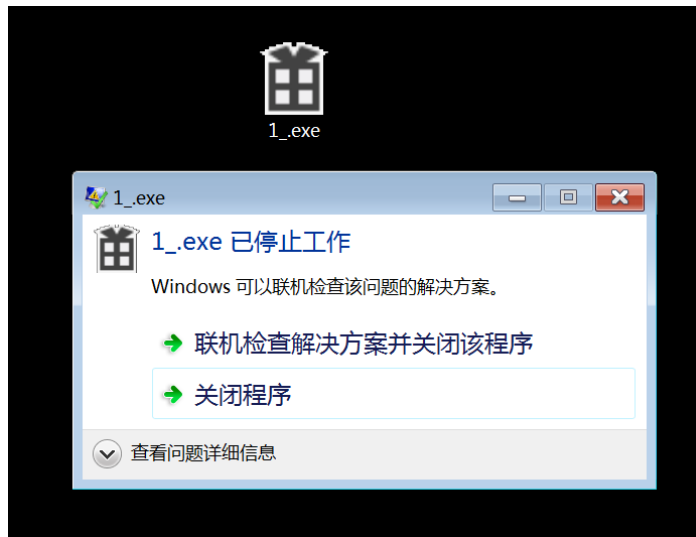


使用 ImportREConstructor 修复 1.exe 文件导入表和 IAT 表:

只有一个 dll 显然不可能。

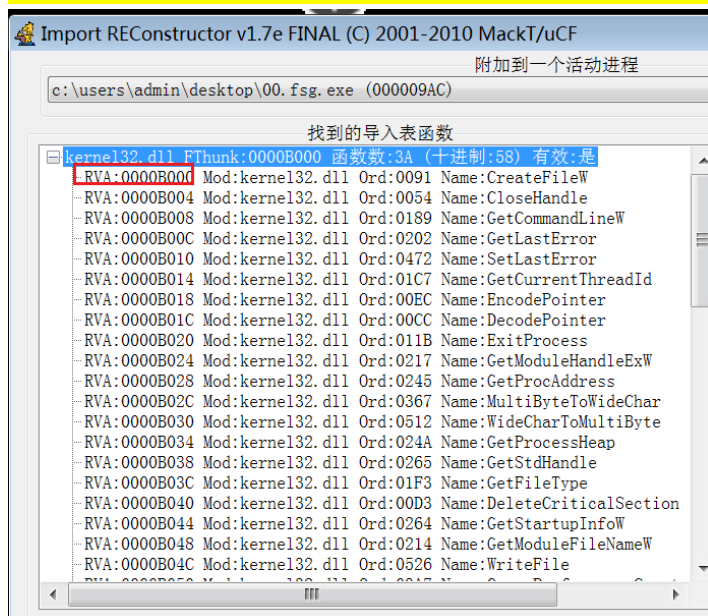


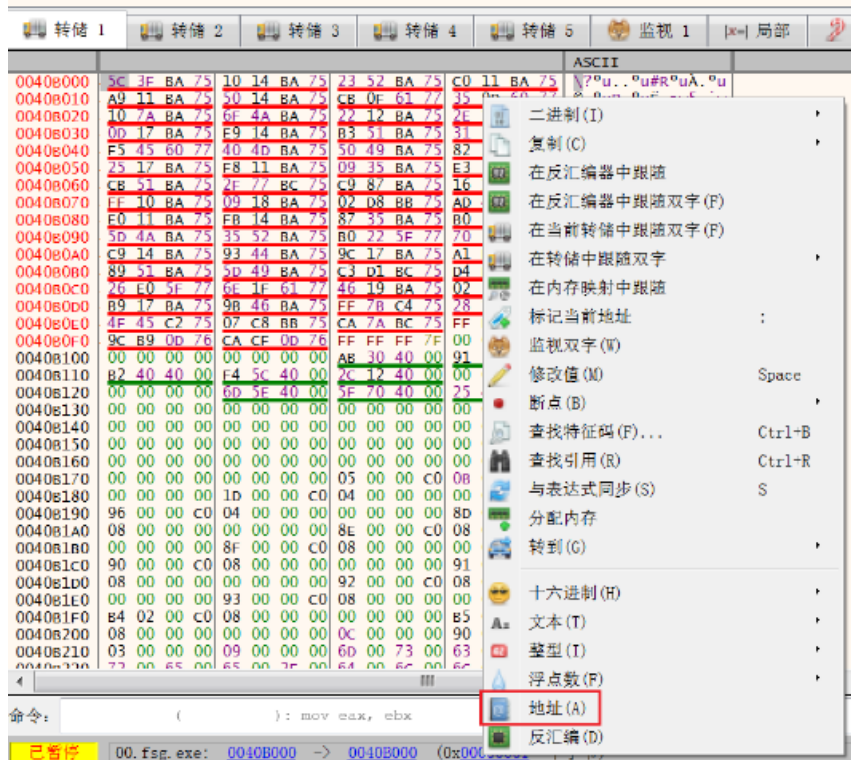
修正转储后：文件运行失败。查原因发现缺少了 dll。



判断不止一个 dll，那还有其他那些 dll 呢？

通过已查出的 dll 函数在 OD 调试器内存中定位函数位置：ImageBase+偏移





转储 1	转储 2	转储 3	转储 4	转储 5	监视 1	局部	结构体
		ASCII					
0040B000	75BA3F5C	??u	kernel32.CreateFileW				
0040B004	75BA1410	..u	kernel32.CloseHandle				
0040B008	75BA5223	#Ru	kernel32.GetCommandLineW				
0040B00C	75BA11C0	A.u	kernel32.GetLastError				
0040B010	75BA11A9	@.u	kernel32.SetLastError				
0040B014	75BA1450	P.u	kernel32.GetCurrentThreadId				
0040B018	77610FCB	E.aw	ntdll.RtlEncodePointer				
0040B01C	77609D35	S.w	ntdll.RtlDecodePointer				
0040B020	75BA7A10	.zu	kernel32.ExitProcess				
0040B024	75BA4A6F	oJu	kernel32.GetModuleHandleExW				
0040B028	75BA1222	".u	kernel32.GetProcAddress				
0040B02C	75BA192E	..u	kernel32.MultiByteToWideChar				
0040B030	75BA170D	..u	kernel32.WideCharToMultiByte				
0040B034	75BA14E9	e.u	kernel32.GetProcessHeap				
0040B038	75BA51B3	3Qu	kernel32.GetStdHandle				
0040B03C	75BA3531	15u	kernel32.GetFileType				
0040B040	776045F5	oE`w	ntdll.RtlDeleteCriticalSection				
0040B044	75BA4D40	@Mu	kernel32.GetStartupInfoW				
0040B048	75BA4950	PIu	kernel32.GetModuleFileNameW				
0040B04C	75BA1282	..u	kernel32.WriteFile				
0040B050	75BA1725	%u	kernel32.QueryPerformanceCounter				
0040B054	75BA11F8	o.u	kernel32.GetCurrentProcessId				
0040B058	75BA3509	.5u	kernel32.GetSystemTimeAsFileTime				
0040B05C	75BA51E3	âQu	kernel32.GetEnvironmentStringsW				
0040B060	75BA51CB	EQu	kernel32.FreeEnvironmentStringsW				
0040B064	75BC772F	/w4u	kernel32.UnhandledExceptionFilter				
0040B068	75BA87C9	E.u	kernel32.SetUnhandledExceptionFilter				
0040B06C	75BA1916	..u	kernel32.InitializeCriticalSectionAndSpinCount				
0040B070	75BA10FF	y.u	kernel32.Sleep				
0040B074	75BA1809	..u	kernel32.GetCurrentProcess				
0040B078	75BBD802	.0»u	kernel32.TerminateProcess				
0040B07C	75BA49AD	.Iu	kernel32.TlsAlloc				
0040B080	75BA11E0	à.u	kernel32.TlsGetValue				
0040B084	75BA14FB	û.u	kernel32.TlsSetValue				
0040B088	75BA3587	.5u	kernel32.TlsFree				
0040B08C	75BA34B0	°4u	kernel32.GetModuleHandleW				

往下拉：发现 user32.dll。为什么没有被工具识别呢？

导入表中存在无效指针或错误数据：值 FFFFFFFF 和 7FFFFFFF 可能是无效的地址或占位符，这表明导入表可能在某个阶段被破坏或被篡改。修复工具在遇到这种无效数据时，通常会忽略它们。

转储 1	转储 2	转储 3	转储 4	转储 5	监视 1	局部
		ASCII				
0040B0A8	75BA179C	..°u	kernel32.GetACP			
0040B0AC	75BCD1A1	jN%u	kernel32.GetOEMCP			
0040B0B0	75BA5189	.Q°u	kernel32.GetCPInfo			
0040B0B4	75BA495D]I°u	kernel32.LoadLibraryExW			
0040B0B8	75BCD1C3	ÄN%u	kernel32.RtlUnwind			
0040B0BC	75BCD1D4	ÔN%u	kernel32.OutputDebugStringW			
0040B0C0	775FE026	&a_w	ntdll.RtlAllocateHeap			
0040B0C4	77611F6E	n.aw	ntdll.RtlReAllocateHeap			
0040B0C8	75BA1946	F.°u	kernel32.GetStringTypeW			
0040B0CC	77603002	.0`w	ntdll.RtlSizeHeap			
0040B0D0	75BA17B9	.°u	kernel32.LCMapStringW			
0040B0D4	75BA469B	.F°u	kernel32.FlushFileBuffers			
0040B0D8	75C47BFF	ÿ{Äu	kernel32.GetConsoleCP			
0040B0DC	75BA1328	(.°u	kernel32.GetConsoleMode			
0040B0E0	75C2454F	OEÄu	kernel32.SetStdHandle			
0040B0E4	75BBC807	.È»u	kernel32.SetFilePointerEx			
0040B0E8	75BC7ACA	Êz%u	kernel32.WriteConsoleW			
0040B0EC	FFFFFFFF	ÿÿÿÿ				
0040B0F0	760DB99C	.°v	user32.EndDialog			
0040B0F4	760DCFCA	Êi.v	user32.ShowDialogParamW			
0040B0F8	7FFFFFFF	ÿÿÿÿ				
0040B0FC	00000000				
0040B100	00000000				
0040B104	00000000				
0040B108	004030AB	<0@.				
0040B10C	00403B91	.;@.				
0040B110	004040B2	²@@.				
0040B114	00405CF4	ô\@.				
0040B118	0040122C	,.@.				
0040B11C	00000000				

user32.dll中函数

解决：修改无效值为 0.

转储 1	转储 2	转储 3	转储 4	转储 5	监视 1	局部
		ASCII				
0040B0A8	75BA179C	..°u	kernel32.GetACP			
0040B0AC	75BCD1A1	jN%u	kernel32.GetOEMCP			
0040B0B0	75BA5189	.Q°u	kernel32.GetCPInfo			
0040B0B4	75BA495D]I°u	kernel32.LoadLibraryExW			
0040B0B8	75BCD1C3	ÄN%u	kernel32.RtlUnwind			
0040B0BC	75BCD1D4	ÔN%u	kernel32.OutputDebugStringW			
0040B0C0	775FE026	&a_w	ntdll.RtlAllocateHeap			
0040B0C4	77611F6E	n.aw	ntdll.RtlReAllocateHeap			
0040B0C8	75BA1946	F.°u	kernel32.GetStringTypeW			
0040B0CC	77603002	.0`w	ntdll.RtlSizeHeap			
0040B0D0	75BA17B9	.°u	kernel32.LCMapStringW			
0040B0D4	75BA469B	.F°u	kernel32.FlushFileBuffers			
0040B0D8	75C47BFF	ÿ{Äu	kernel32.GetConsoleCP			
0040B0DC	75BA1328	(.°u	kernel32.GetConsoleMode			
0040B0E0	75C2454F	OEÄu	kernel32.SetStdHandle			
0040B0E4	75BBC807	.È»u	kernel32.SetFilePointerEx			
0040B0E8	75BC7ACA	Êz%u	kernel32.WriteConsoleW			
0040B0EC	00000000				
0040B0F0	760DB99C	.°v	user32.EndDialog			
0040B0F4	760DCFCA	Êi.v	user32.ShowDialogParamW			
0040B0F8	00000000				
0040B0FC	00000000				
0040B100	00000000				
0040B104	00000000				
0040B108	004030AB	<0@.				
0040B10C	00403B91	.;@.				
0040B110	004040B2	²@@.				
0040B114	00405CF4	ô\@.				

重新加载文件 dll，并修正转储文件:



5-3 修复：成功。

