实验：使用ESP定律脱壳实验

一、实验目的：

1. 通过实验理解ESP定律基本原理及其应用
2. 掌握PE文件分析常用工具的使用
3. 掌握UPX壳手动脱壳方法
4. 掌握x32dbg调试工具硬件断点的使用方法

二、实验环境：

操作系统：Windows 7/8/10

实验对象：DiskProbe.exe

实验软件：ExeinfoPE，x32dbg

三、实验内容：

1. 实验背景

软件逆向工程和恶意代码分析过程中，经常分析到经过**加壳**的PE文件，因此必须首先进行**脱壳**操作。加壳和脱壳是相反的两种操作，加壳是对PE文件数据进行变换操作，从而实现对文件数据的压缩或加密操作；而脱壳则是加壳的逆向操作，就是将加壳的程序数据还原到加壳之前的原始数据状态。为了能够让加了壳的程序正常运行（操作系统的PE加载机制并不能识别加了壳以后的数据），加壳后的程序在运行时会先进行自身脱壳，然后再执行原始的程序。这个特性也是下文中采用手动的方式脱壳的基本思路来源。

加壳操作一般使用专用的加壳程序实现对目标程序的加壳，而脱壳相对比较复杂。在实际工作中，先使用工具软件查出壳的类型，比如ExeinfoPE工具，然后再使用对应的脱壳程序进行自动化脱壳，此类方法的关键是准确判断加壳类型并找到对应的脱壳程序，操作相对简单，这里不进行介绍。

如果遇到比较特殊的某种“壳”，或无法使用工具自动脱壳的时候，可以采用人工手动脱壳方法，本次实验就是介绍了一种手动脱壳技术，基于ESP定律的脱壳。实验主要使用动态调试工具，如ollydbg或x96dbg（软件包含x64dbg和x32dbg，分别针对64位和32位的程序）等工具，通过动态分析，获得程序**真实的入口代码地址OEP（Original Entry Point）**，最后使用相关工具将内存中的程序保存到文件中，实现最终的脱壳操作。

> 加壳后的软件在运行时，必须先将加了可的代码指令“解密/释放”到内存中，这时只要找到指令的起点，即OEP地址，那么就可以使用一些工具软件将内存中已经“解密”的程序转存（dump）出来，保存在本地文件，实现了脱壳。转存过程中涉及到PE格式修复，地址重定位修改等一系列操作，通常都是使用工具软件自动完成。

2. 实验内容

实验目标程序一款磁盘文件系统检测和数据恢复软件。软件作者在软件的编写过程中对程序执行的时间做了限制，如果系统时间超过了设定的期限，程序将显示提示信息并退出程序，如图 1所示。

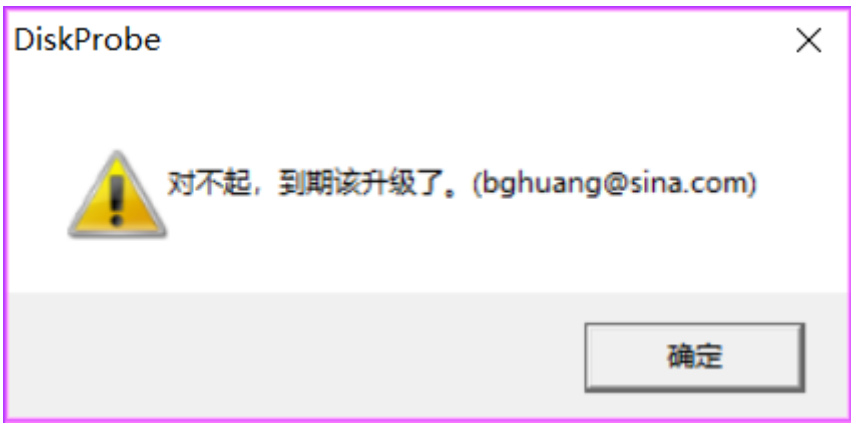

图 1 程序执行时间限制

为了实现破解该软件的升级提示对话框，首先要对目标进行脱壳。本次实验就是通过学习使用动态调试工具，利用ESP栈平衡定律实现对加壳的软件进行脱壳操作,最终得到脱壳后的DiskProbe.exe。

四、实验步骤：

1. PE文件基本信息分析。

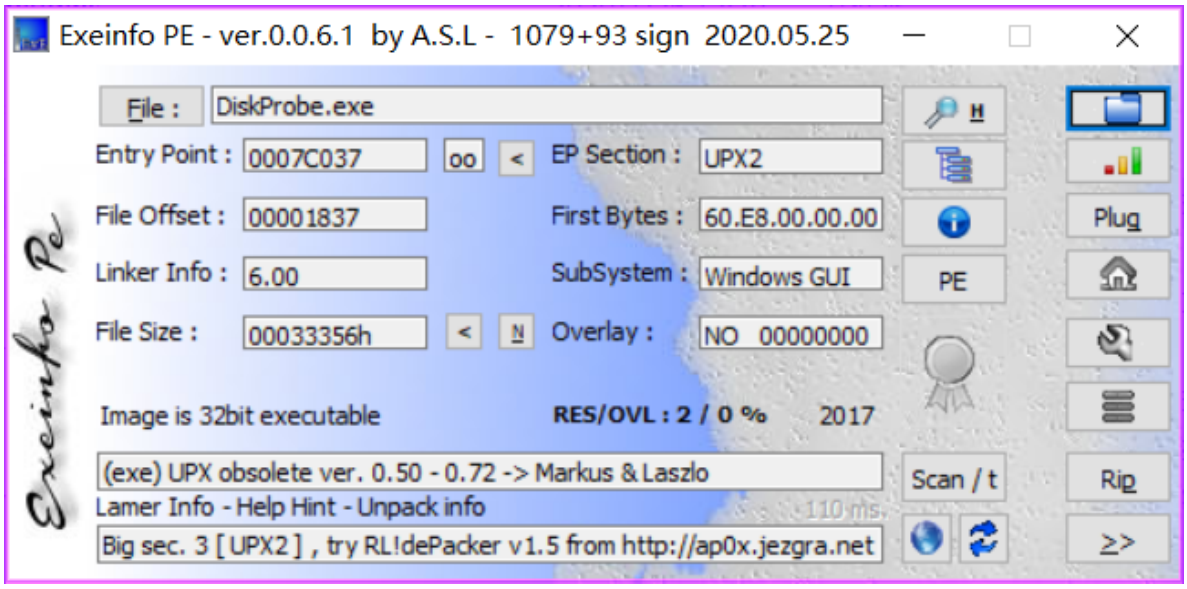

图 2 使用ExeinfoPE工具检查“壳”的类型

为了实现对PE文件的逆向分析和破解，通常首先需要获得PE文件的一些基本信息，如图 2所示的界面中，对DiskProbe.exe分析，发现其文件经过了UPX软件的压缩加壳，软件初步分析给出加壳软件版本是0.50-0.72。单击旁边**“Scan/t”**按钮，分析结果为UPX 0.70。

> 图2中可以看到diskprobe.exe程序OEP是0007c037H，代码入口点所在的节是UPX2等信息。这里UPX实际上是使用UPX加壳软件加壳后的软件的一个节的名称，具有很明显的特征性。这里查看到的OEP是加壳以后的程序运行的程序开始点，而不是程序真实的OEP，脱壳过程就是要找的这个OEP。

2. 手动脱壳基本原理

实践工作中会使用脱壳软件实现自动化脱壳,快捷方便效率高，如本题目可以使用LinxerUnpacker实现全自动脱壳。但本次实验目标是通过使用手动脱壳加深对脱壳的一般过程的理解。

手动脱壳通常的关键是找到软件OEP（Original Entry Point）也就是加壳的软件中真正的软件执行的代码起点，又称“原始代码入口点”。按照查找OEP的方法可以将手动脱壳分为“手动单步跟踪法”、“ESP定律方式”和“内存二次断点法”。

本文重点采用ESP定律法进行脱壳操作。

（1）手动单步跟踪法

手动单步追踪法是使用调试器的单步调试，如使用调试器的动态调试功能，如快捷键F8（单步执行）和F4（执行到指定位置）。通过对反编译代码的分析和执行，查找OEP。这种方式的缺点是效率低下，对分析人员的综合能力要求较高。

（2）内存二次断点法

内存二次断点法是利用经过加壳的程序在执行的时候需要分别解压缩不同区段(Segment)中内容，按照解压缩的先后顺序的特点，通过分别在资源段和代码段添加断点的方式，通过2次断点，就能确定OEP的大概位置。

- 首先在资源段上下一个断点，那么在解压缩代码的时候，程序依旧可以正常的运行，因为是先解压缩代码再解压缩资源的。当开始解压缩资源的时候，程序就会中断下来，因为已经在资源段上下了一个断点，此时中断的时候，代码已经全部解压，资源段正在准备解压...
- 然后在代码段上下一个断点，再继续运行，此时程序开始解压缩资源和其他的一些内容，等到全部解压缩完成以后，解压缩程序就需要跳转到OEP上运行，即跳转到代码段上运行，此时中断下来。
- 最后手动F8往下跟几步，就到达跨段跳转到OEP的指令位置。

（3）ESP定律法

ESP定律方式相对以上的方式更方便高效，因此本次实验将采用这个方法。

- 什么是ESP定律?

每一个函数的调用在堆栈里都必须开辟一个栈帧，而这个栈帧是建立在原有的栈基础上的，所以当代码执行进入函数的时候，开辟新的栈帧前必须将原来的栈信息进行保存。

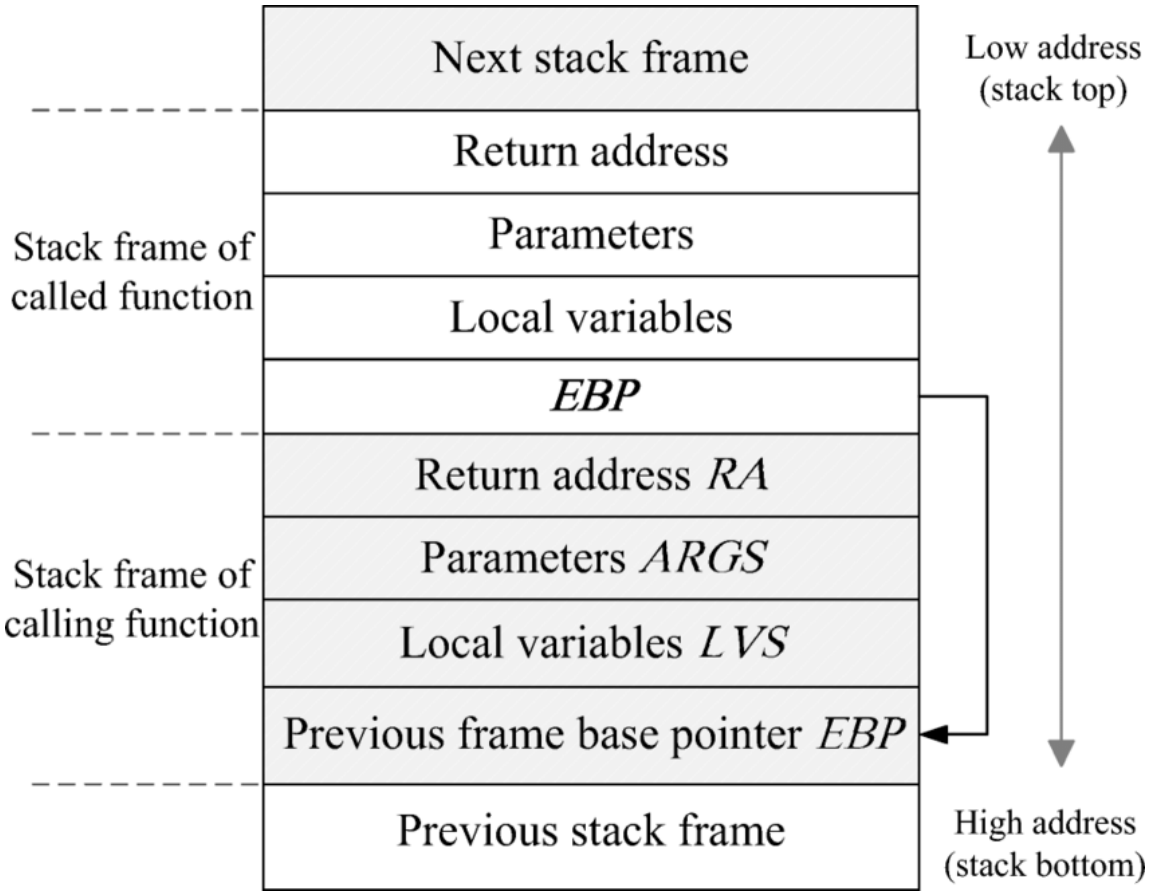

图3 栈帧在内存中布局

如图3所示的栈帧布局中，新开辟的栈帧是叠加在原来函数栈帧”上“面的。为了后面的描述方便，“上”面的函数栈帧这里成为子函数栈帧或被调用函数栈帧，“下面”的栈帧称为父函数栈帧或调用函数栈帧。实际以上的描述也表明了以上2个栈帧以及函数之间调用关系。

> 注意：ESP寄存器值指向当前栈的栈顶，EBP寄存器指向当前栈的栈底。一般使用[ESP+4]表示ESP向栈底方向移动4字节的栈中的数值。图3中，*stack frame of calling function:源栈帧；stack of called function:被调用函数开辟的新栈帧*。

被调用函数的栈帧在“上面”，调用函数栈帧在“下面”。当代码执行从调用函数进入被调用函数时，在被调用函数（called function）中需要执行一条PUSH EBP指令，将调用函数（calling function）的EBP保存在栈中，如图3中EBP是被调用栈帧的栈底，并指向了调用栈的栈底。请同学们思考为什么只需要将EBP入栈保存，而不需要保存ESP?

在函数退出前，函数的栈需要还原到函数调用之前的状态，这通常会执行对栈的逆向操作。也就是当前函数返回前，必须使用一系列的指令让ESP指向图3中EBP位置，然后会执行一个POP EBP指令，这样就能还原EBP和ESP的数值，然后在执行函数最后的ret返回指令，就能使用栈低的返回地址完全退出子函数。

在以上函数进入以及函数退出的过程中，分别对EBP执行了PUSH和POP操作。而这两个操作的时候，ESP寄存器的值时相等的。也就是说函数执行前后的栈是“平衡”的。更通俗的说就是函数执行前ESP会指向一个值，函数执行完毕后ESP也应该重新指向这个数值。

而经过加壳的程序在运行时，类似是先调用了脱壳函数，然后再执行脱壳后的代码。所以只要重点关注ESP的两次访问，就能确定脱壳函数的调用，从而找到脱壳后的代码。

- 如何使用ESP定律?

由ESP定律的原理可得到函数执行完成后，ESP指针会进行还原为先前的状态（第二次指向栈中某个位置）。因此可以对ESP寄存器下一个硬件断点，使得ESP还原为调用函数之前状态的时候触发中断，代码中段的位置就是“壳”的解压缩的函数执行完毕的位置，这样就能方便的得到已经”脱壳“的程序代码在内存中的位置。

> 假设将一个壳的解压缩的方式当作一个函数的调用。如经过UPX加壳的程序在运行的时候，使用调试器能较为方便的找到UPX解压缩的函数，此时它会保存EBP，然后移动ESP，给函数留出足够的栈帧，当所有的解压缩完成后，它需要把所有的栈帧再进行还原，使得ESP恢复为调用函数之前的状态，此时会触发以上所述的硬件断点，而代码中断的位置正好为解压缩函数执行结束位置，而解压缩完成后，就会进行跳转到OEP，此时我们只需要F8（单步执行）往下跟几步，就能很快的找到OEP了!

- 为什么需要使用ESP定律?

经过加壳的程序，在运行时首先会调用自身的”解密程序“实现脱壳操作，然后在执行经过”解密“的原始程序的代码。虽然能在分析代码中找到调用解密程序call指令，但是却不太容易找到解密后执行指令的位置，这时候利用栈帧的平衡原则，使用ESP定律就能方便找到解密后的指令，继续就可以找到OEP，就能实现手动的脱壳。

3. ESP定律脱壳实验步骤

（1）加载分析程序

打开x32dbg软件，将需要分析的程序diskprobe.exe拖入软件主窗口，此时会自动加载并进入调试分析状态。

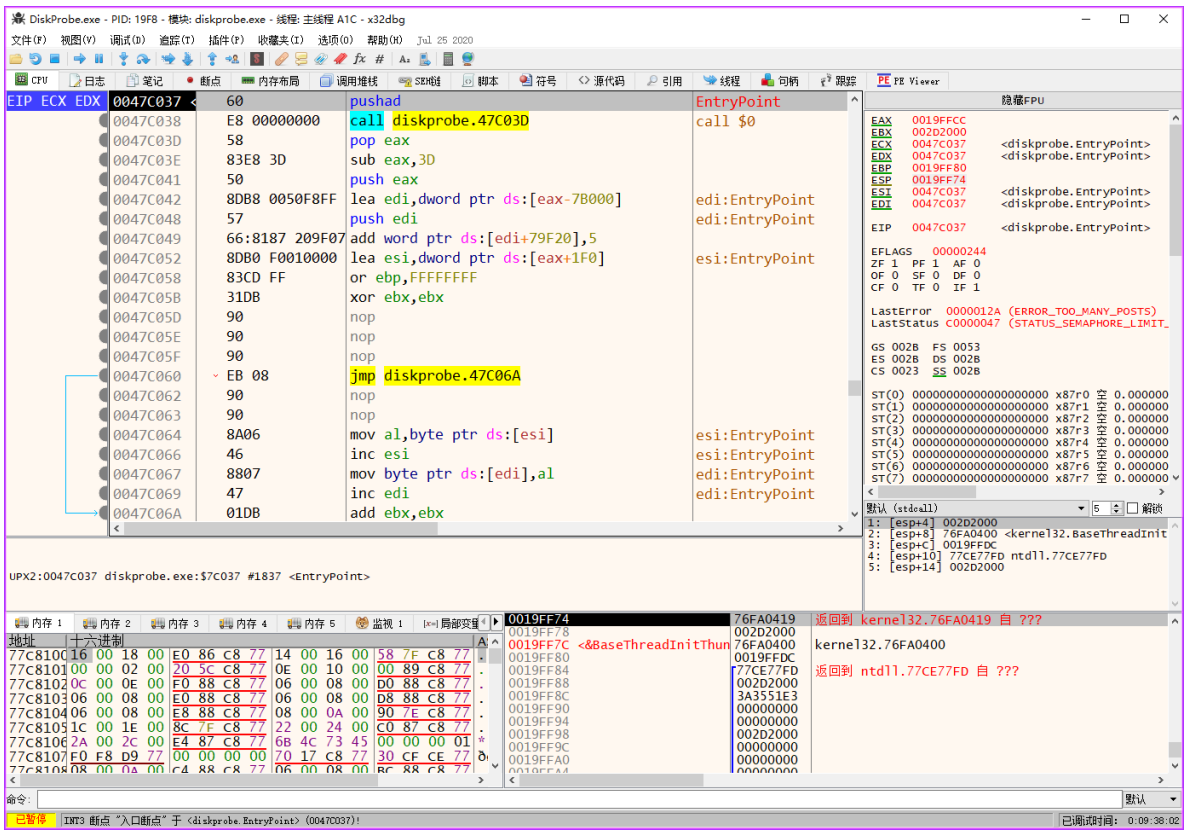

图 4 进入壳函数

x32dbg程序打开后默认显示的代码是系统库函数调用的代码，并不是分析的diskprobe中代码，按 `alt+F9` 快捷键可以快速执行到"用户代码"，也就diskprobe程序的代码。程序会停在如图4所示的代码位置，经过分析前两行的代码，推测 call diskprobe.47c04d 这段代码是进入"壳"的自释放代码入口。

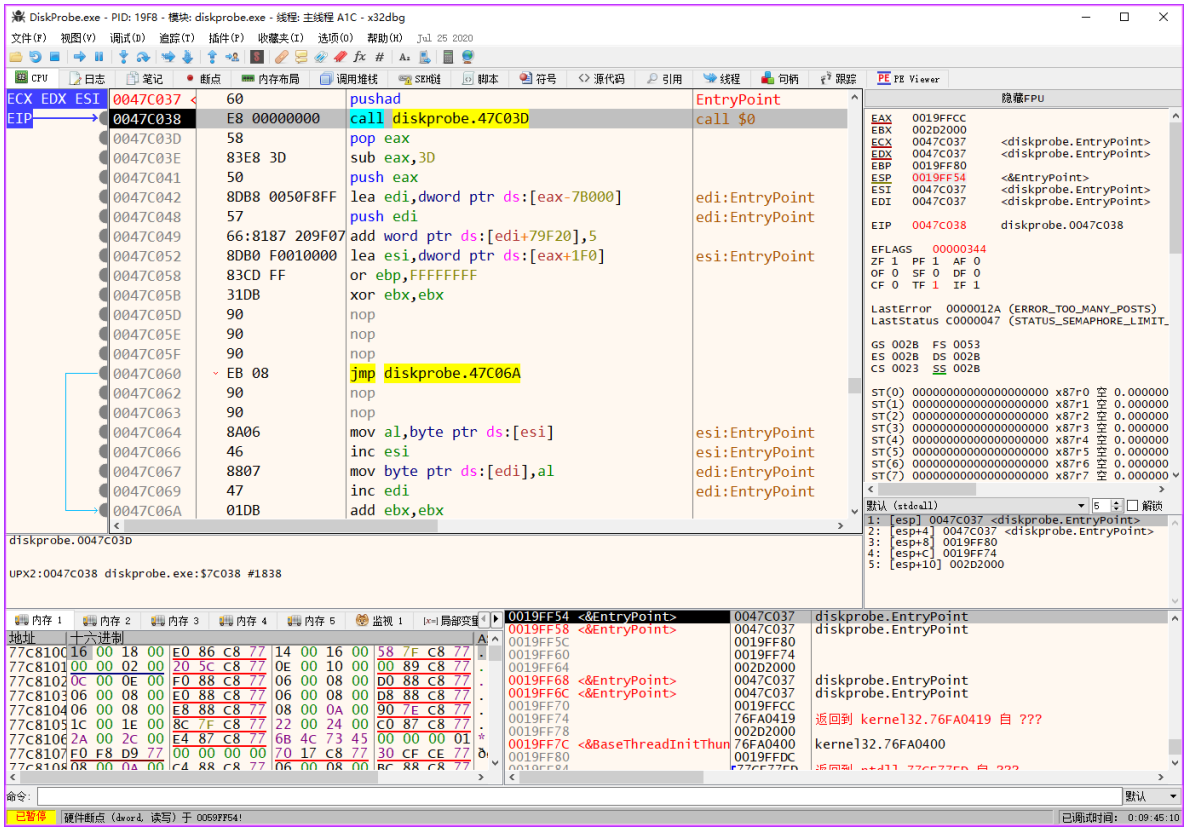

图 5 查看ESP值

按 `F7` 快捷键执行一条指令，使EIP寄存器指向 call diskprobe.47c04d 这条指令，如图5。按照ESP定律，当call指令执行完毕后，必定会再次访问当前ESP所指向的栈地址，此时可以记录下栈顶也就是ESP的地址为0019FF54。

（2）添加断点

对当前ESP地址加一个**访问硬件断点**，程序执行完对"壳"的解密后，必然会触发这个中断。x32dbg中增加硬件断点，可以直接在栈地址上单击右键，选择硬件访问断点。也可以在命令行执行以下命令添加硬件断点，在程序界面下方的命令输入栏，输入以下命令。

```
bph esp, r, 4
```

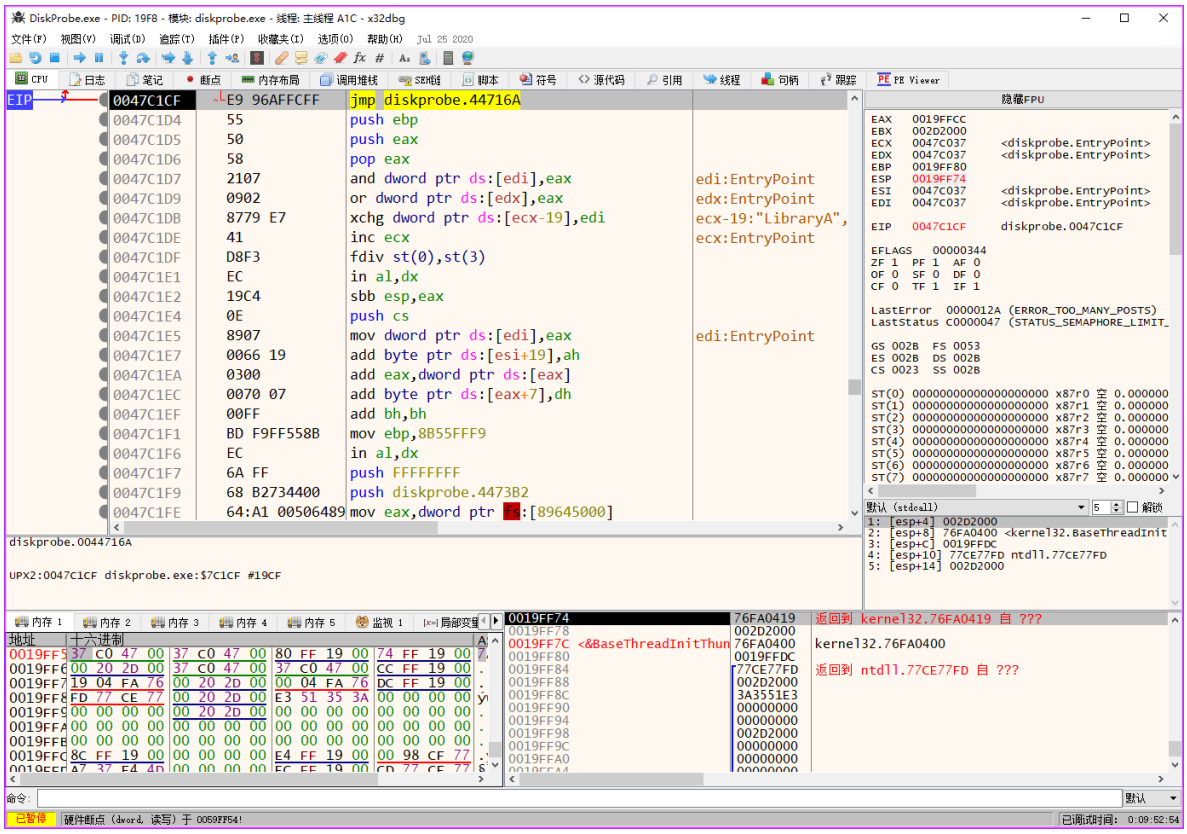

图6 硬件中断到函数结束

设置好硬件中断后，按F9继续执行代码到中断位置。注意该断点是访问断点，因此第一次中断是进入函数前，第二次中断是执行完函数，这里需要找到解密后的代码执行，所以需要查看第二次中断，如图 6所示。

注意到中断中如下代码：

```
0047C1CF  | E9 96AFFCFF      | jmp diskprobe.44716A                  |
0047C1D4  | 55               | push ebp                              |
0047C1D5  | 50               | push eax                              |
0047C1D6  | 58               | pop eax                               |
```

其中jmp diskprobe.447167A是一个跨节（section）的长跳转，说明前面的指令一直是在解密加了壳的指令代码，而解密后的指令代码放在了另外的节中，所以执行就必须一个远距离的跨节跳转指令衔接。

> 这里看出是跨节跳转，主要是比较当前地址和目标地址是否在同一个代码节中。而具体的节的区域，可以通过查看节表获得。比较方便的是使用PE Viewer插件查看。
>
> 本实验中，查询到

Name	VirtualAddress	VirtualSize
UPX0	00001000	00079000
UPX1	0007a000	00001266
UPX2	0007c000	00031b56

将以上的Virtual Address加上基址0x400000,可以判定出 47C1CF在UPX2节，而跳转目标地址44716A在UPX0节中，所以是跨节长跳转。

(3) 导出脱壳后程序

图6程序中断在jmp指令，此时按快捷键 `F7` 可以单步执行到目标地址，代码如下：

```
0044716A  | 55              | push ebp                              |
0044716B  | 8BEC            | mov ebp,esp                           |
0044716D  | 6A FF           | push FFFFFFFF                         |
0044716F  | 68 48DE4400     | push diskprobe.44DE48                 |
00447174  | 68 D4724400     | push <JMP.&_except_handler3>          |
00447179  | 64:A1 00000000  | mov eax,dword ptr fs:[0]              |
```

此时判断0044716A是OEP，只需将内存中从OEP开始的内存中数据“DUMP”到文件中就可以实现脱壳。但实际dump时，必须按照PE文件的格式要求修复其中的各项数据，纯手工操作需要修改很多参数，复杂低效率，通常这些操作会使用一些工具。在x32dbg中的可以使用scally插件自动完成。

- Dump程序文件

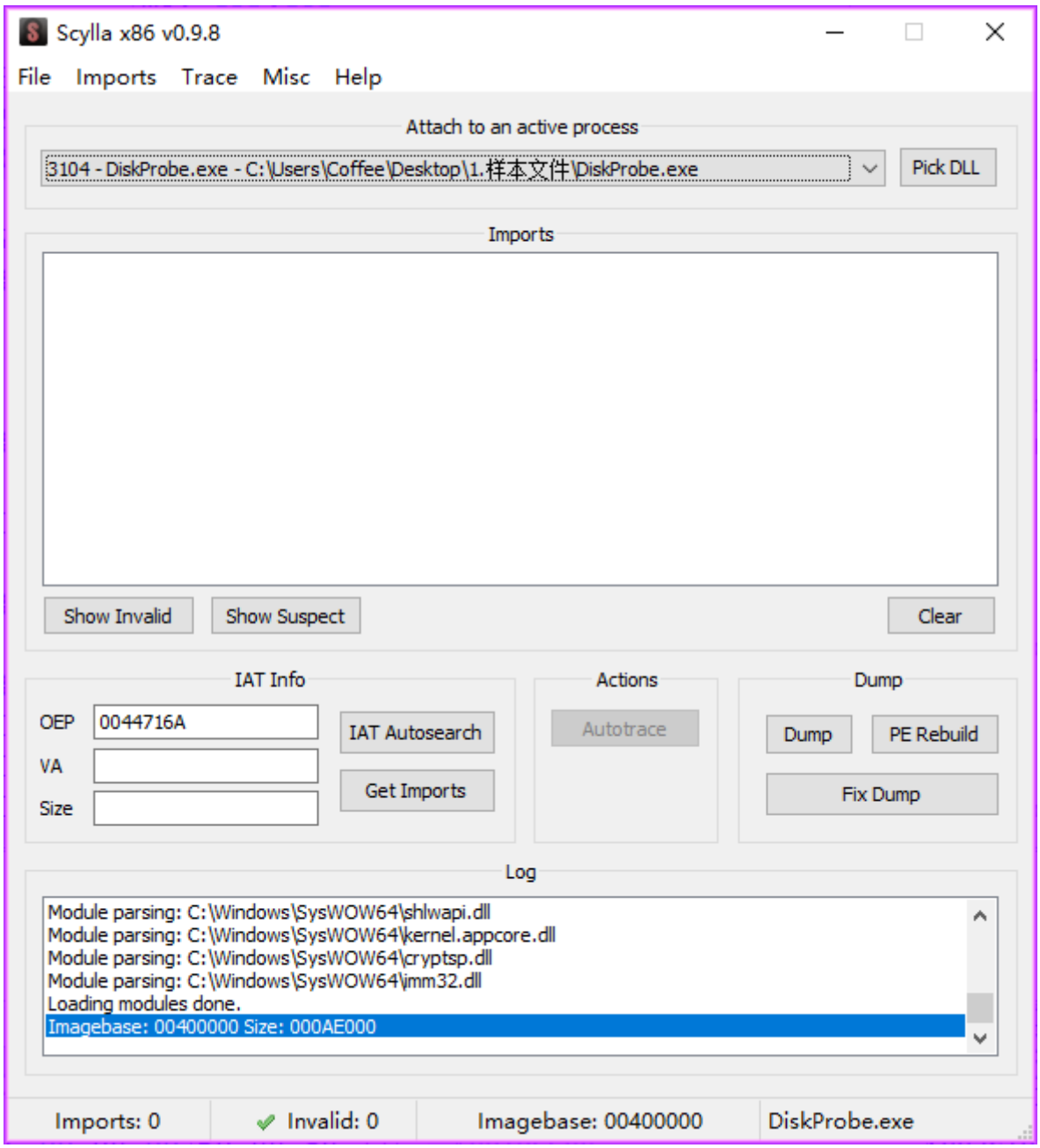

图7 启动scally插件

在x32dbg的插件菜单中选择scally插件启动，如图 7所示。在OEP中填入上一节中找到的地址。依次单击IAT Autosearch和 Get Imports按钮。

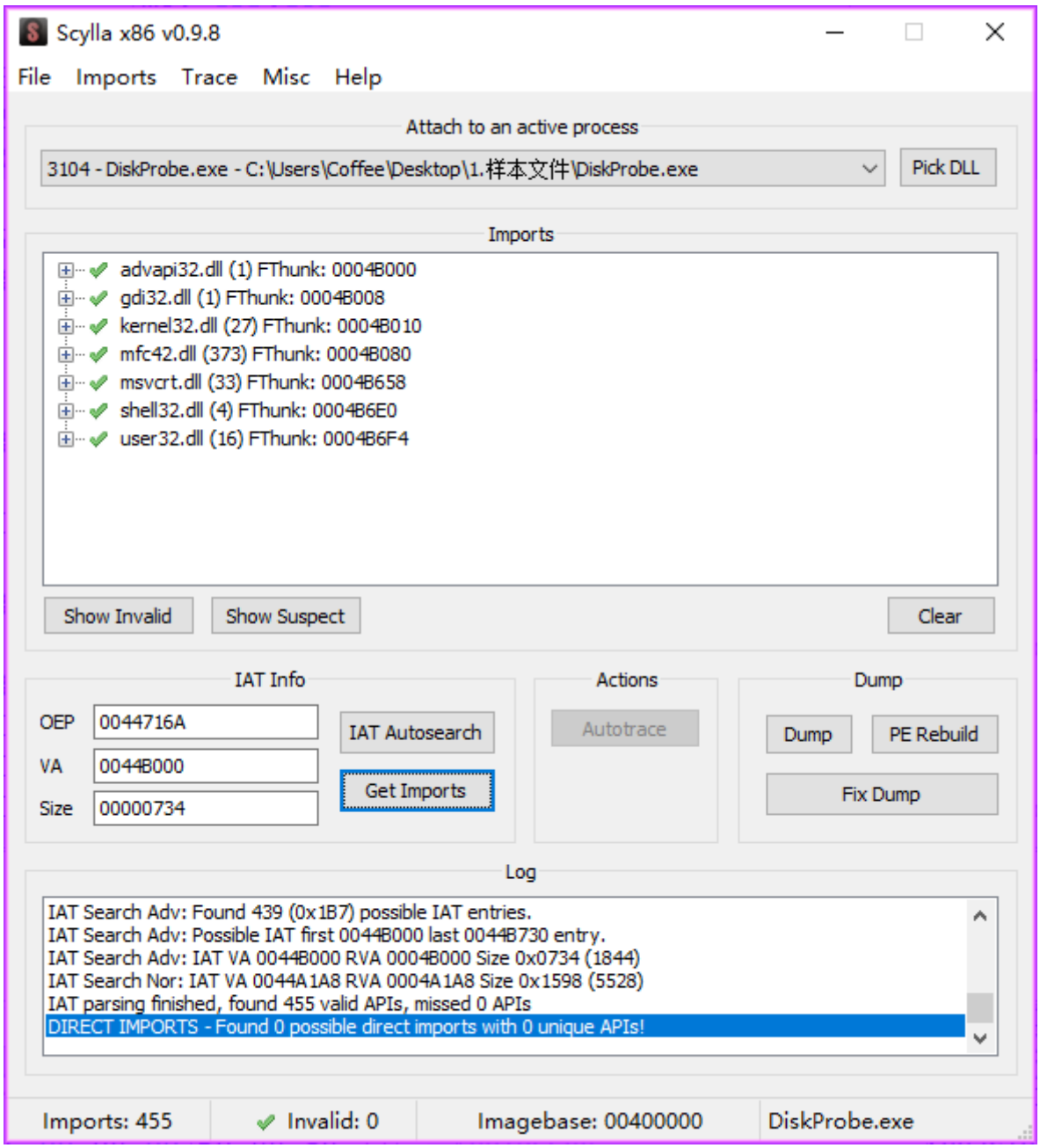

图8 Scally插件

如果一切正常，插接显示如图8。最后单击Dump按钮将数据保存。

- 导入并修复程序文件

  Dump出的文件需要进一步的修复才可以使用。

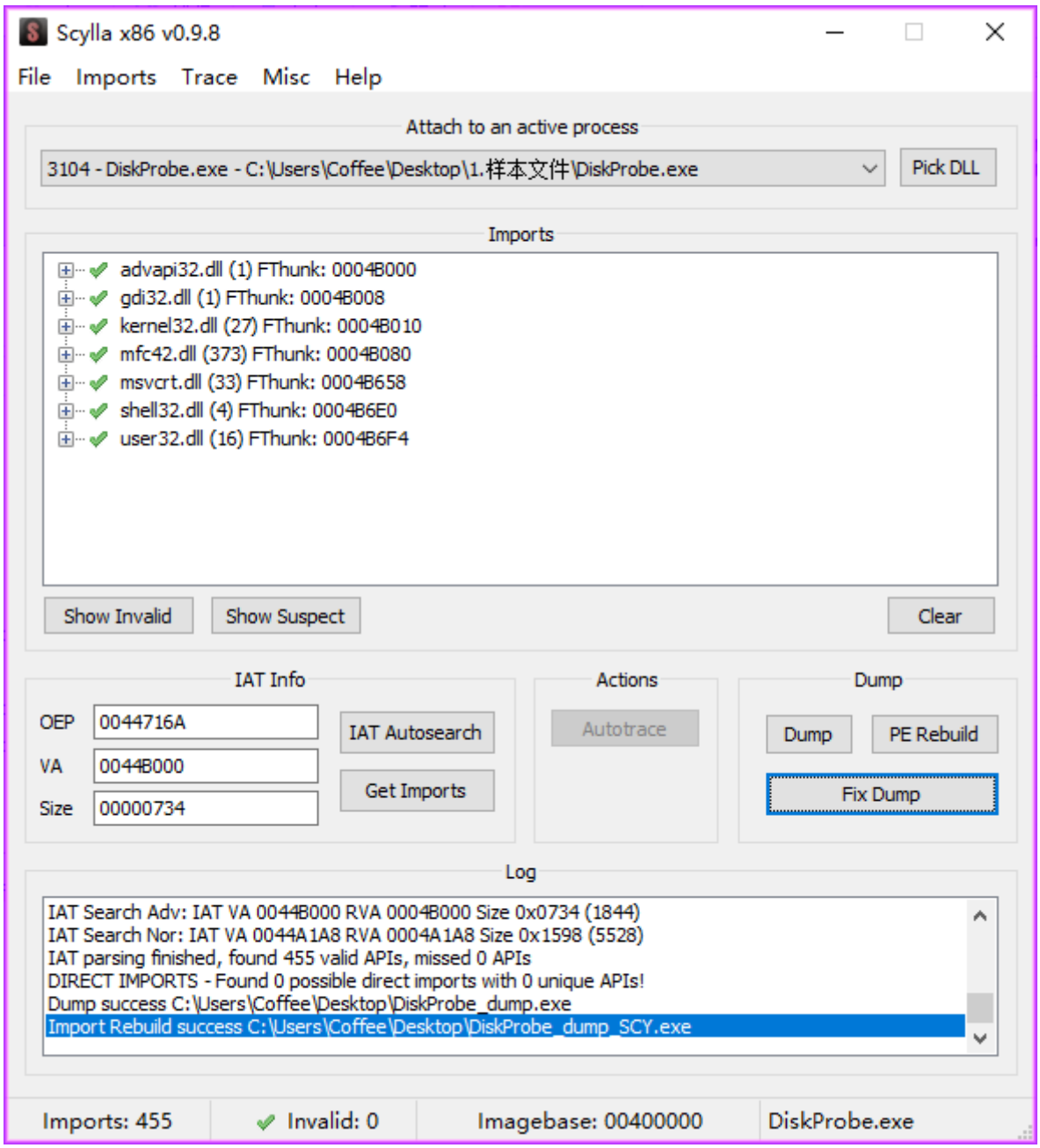

图9 Fix Dump

点击Fix Dump按钮，然后选中上一步Dump出的文件，会自动修复保存，如图9所示，修复的文件是“DiskProbe_dump_SCY.exe”，保存在桌面文件夹中。

最后运行修复的文件，如果能正常打开说明手动脱壳成功。

五、实验结果：

1. x32dbg加载目标程序后，用户代码的第一条指令截图。
2. PUSHAD指令地址（ ）
3. 第二次硬件中断的地址（ ）
4. OEP地址（ ）
5. 全部脱壳后程序有（ ）个节（Section）

六、思考题：

1. 如何在硬件断点后找到OEP地址?

2. 请尝试查找专用工具对本实验中的程序进行自动化脱壳，给出实验软件截图。