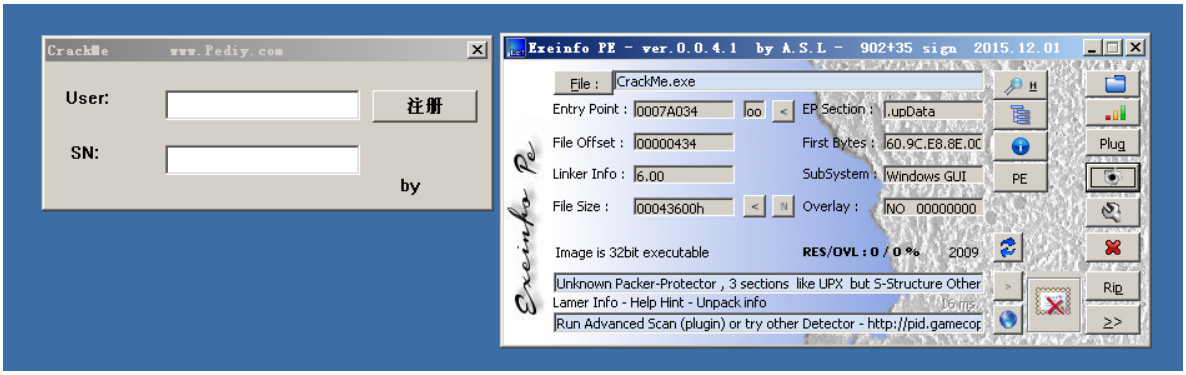
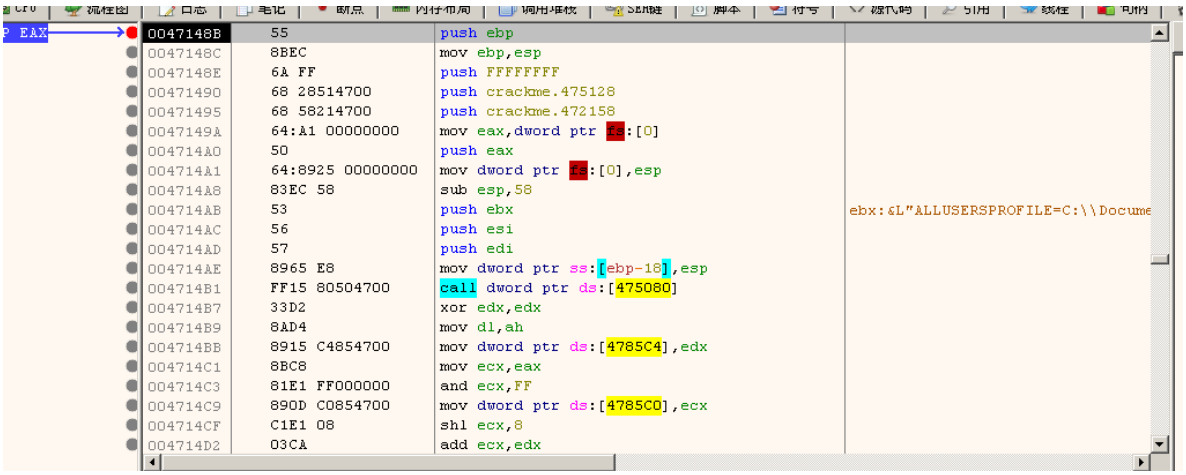
脱壳实例

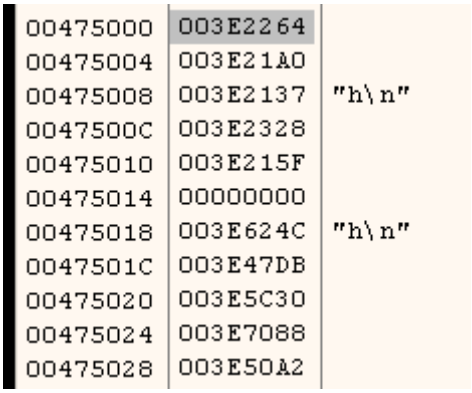
实例

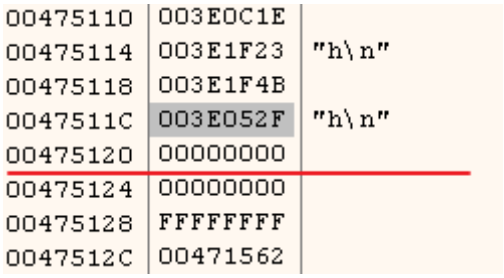


一个压缩壳，ESP定律秒杀，入口点：0047148b

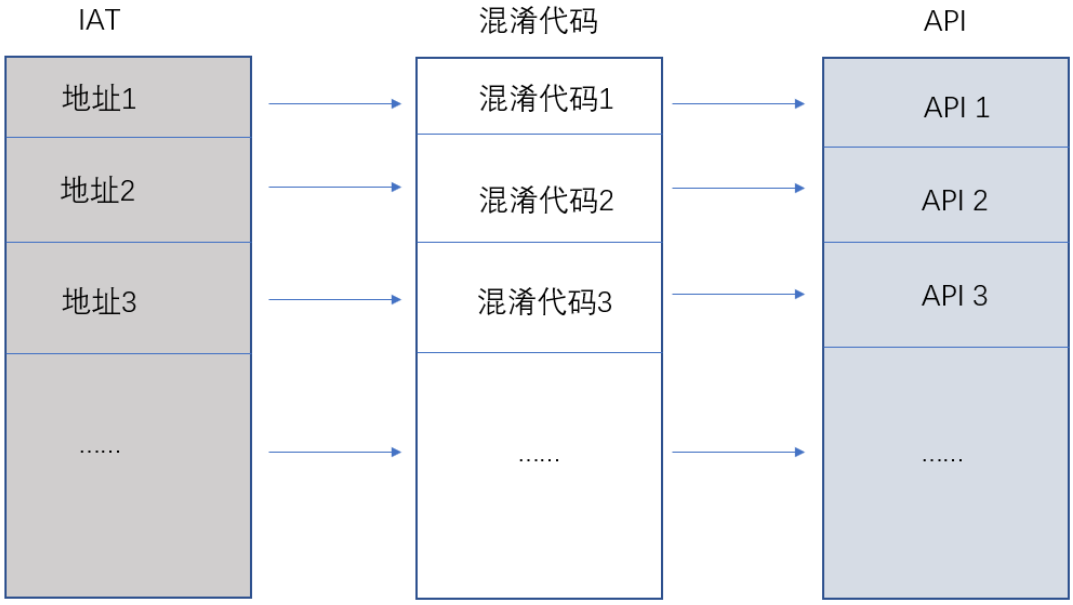


定位IAT 00475000，大小 0x120





根据分析得知，IAT所填写的地址并不是API的地址，中间过程加了混淆代码（在堆上），大致如图所示

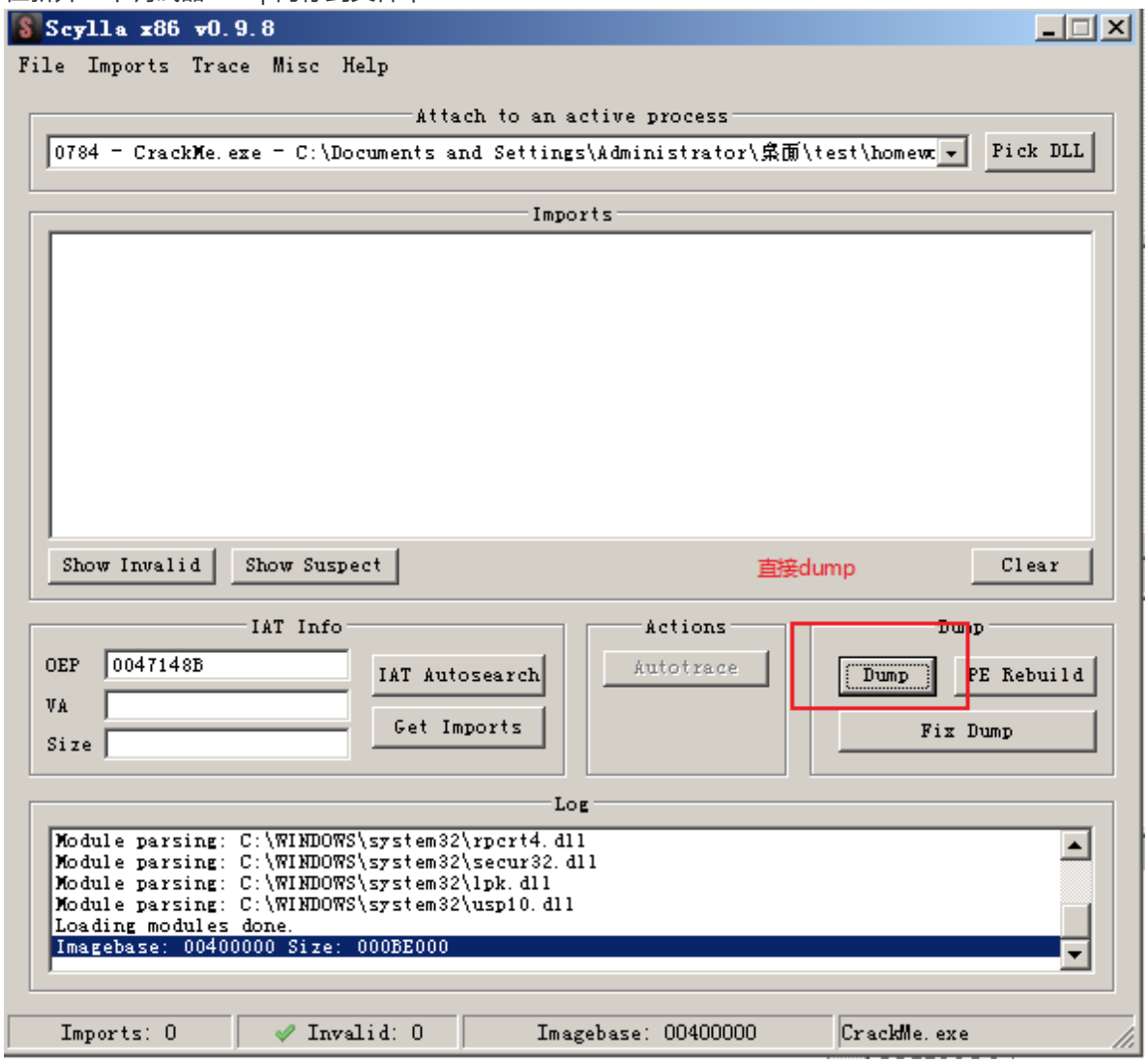


且混淆代码最后都会通过`ret`指令从栈中返回并调用正确的API，可用通过编写脚本来得到正确的IAT

```
from x64dbgpy.pluginsdk._scriptapi import *

IAT = 0x00475000     # IAT地址
IATSize = 0x120      # IAT大小
ItemCount = IATSize / 4 # 计算出IAT项个数
RetCode = 0xC3       # ret指令硬编码

# 遍历IAT
for i in range(0, ItemCount):
    ObfuscatedAddr = ReadDword(IAT + i * 4) # 获取混淆代码地址
    if ObfuscatedAddr == 0:
        continue

    # 设置EIP为混淆代码处
    SetEIP(ObfuscatedAddr)

    # 单步寻找ret指令
    while True:
        StepIn()
        if ReadByte(GetEIP()) == RetCode:
            # 获取栈上API地址写回IAT
            APIAddr = ReadDword(GetESP())
            WriteDword(IAT + i * 4, APIAddr)
            break
```

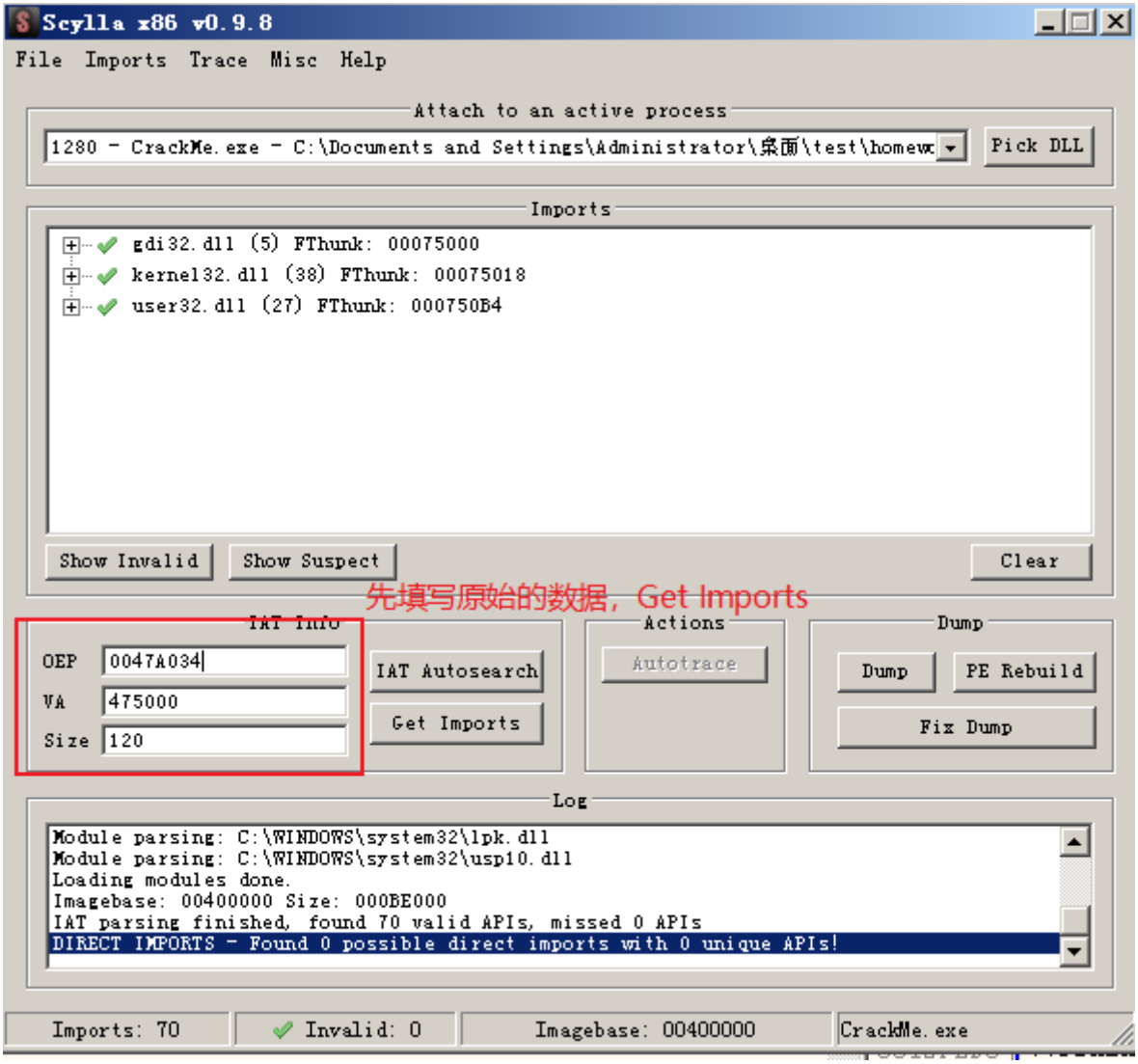
```
00475000 77EF5D77 gdi32.SetTextColor
00475004 77EF5EDB gdi32.SetBkMode
00475008 77EFED78 gdi32.GetTextExtentPoint32A
0047500C 77EFAF1D gdi32.TextOutA
00475010 77EFDDA1 gdi32.GetTextMetricsA
00475014 00000000
00475018 7C801D7B kernel32.LoadLibraryA
0047501C 7C80AE40 kernel32.GetProcAddress
00475020 7C939B80 ntdll.RtlReAllocateHeap
00475024 7C809AF1 kernel32.VirtualAlloc
00475028 7C9300A4 ntdll.RtlAllocateHeap
```

在新开一个调试器dump内存到文件中

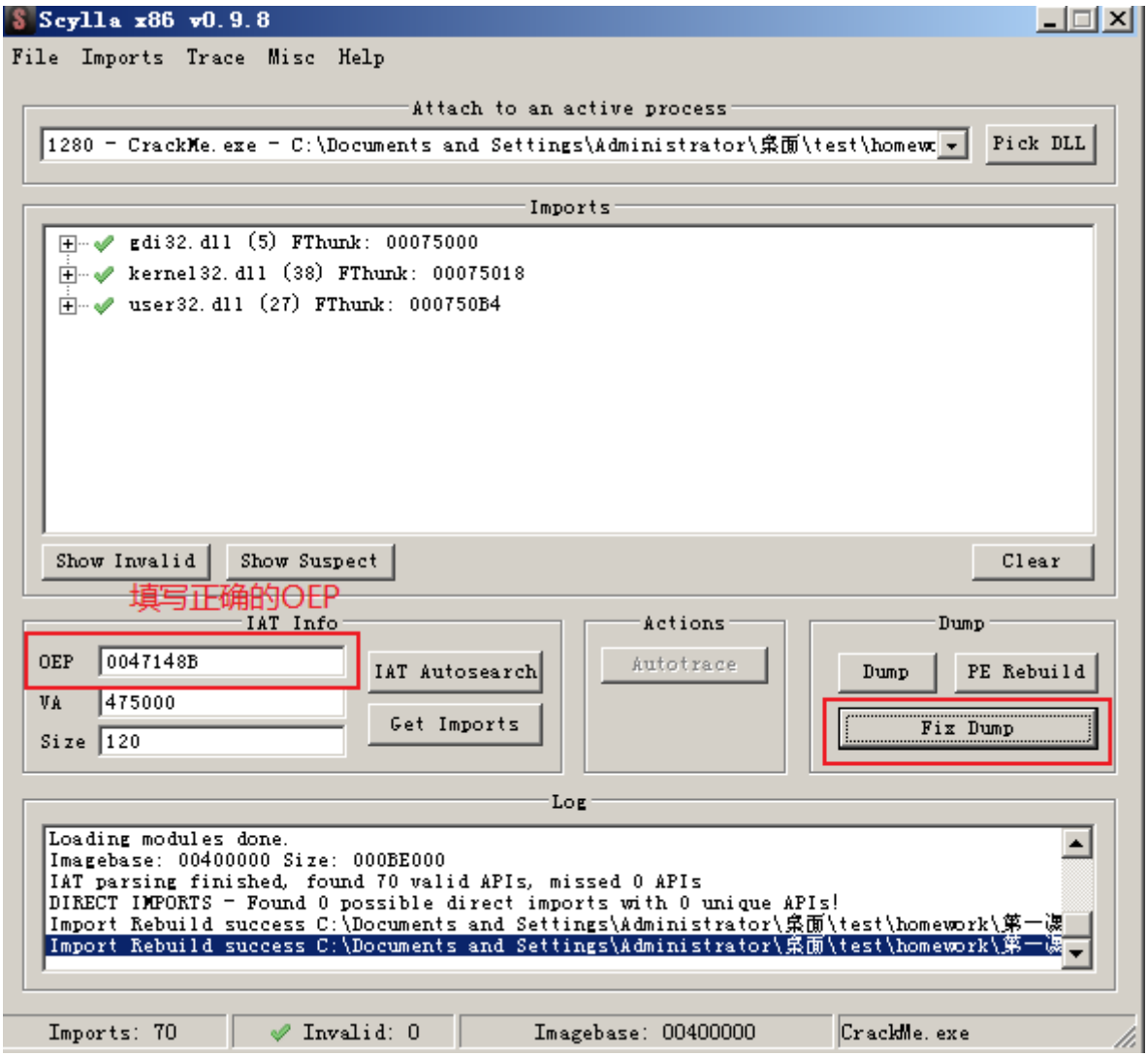


关闭新开的调试器，在之前的调试器中对dump后的文件进行导入表修复

先获取正确的IAT



然后对dump文件进行修复



脱壳完工

代码混淆

代码混淆可用通过对一段执行的序列进行分片打乱顺序，而后通过 `jmp` 来跳转到各个分片中执行

```
; 未混淆
code segment 1
code segment 2
code segment 3
code segment 4
....

; 混淆后
    jmp code1
code2:
    code segment 2
    jmp code3

code4:
    code segment 4
    jmp codeN

code1:
```

```
    code segment 1
    jmp code2
code3:
    code segment 3
    jmp code4

codeN:
...
```

对于`jmp label1`的等价指令为

```
    call label1
    ...
    ...
    ...
label1:
    ...
    ...
    ...
    pop reg     ; 栈中的call的返回地址弹栈
    ...
    ...
```

同样等价于

```
    call label1
    ...
    ...
    ...
label1:
    ...
    ...
    ...
    lea esp, [esp + 4]      ; 栈中的call的返回地址弹栈
    ...
    ...
```