第八章 软件脱壳技术

北京邮电大学
崔宝江

第八章 软件脱壳技术

- 一. 壳的概念
- 二. 脱壳的相关工具
- 三. 逆向分析

北邮网安学院 崔宝江

一. 壳的概念

- 1 基础概念
- 2 壳的原理和分类
- 3 加壳程序的运行流程

一. 壳的概念

- 1 基础概念
  - “壳”就是专门压缩/加密的工具，通过在压缩/加密的过程中加入保护性代码，程序文件会失去原来的程序结构，改变代码的表现形式，增加被篡改和反编译的难度，达到保护程序内部逻辑的效果。
  - 加壳
    - 对可执行文件进行压缩/加密的过程
  - 脱壳
    - 而对已经加壳后的程序进行解压缩/解密的过程

一. 壳的概念

2 壳的原理和分类

1）压缩壳

- 以减小软件体积和改变软件可执行代码的特征为目的
- 压缩壳的主要目的对程序进行压缩，对程序的保护不是该类壳的重点。
- 使用压缩壳可以帮助缩减 PE 文件的大小，隐藏了 PE 文件内部代码和资源，便于网络传输和保存。
- 目前兼容性较好的压缩壳主要有**ASPack**、**UPX**和**PECompact**等。

一. 壳的概念

2 壳的原理和分类

（2）加密壳

- 以保护软件为目的，根据用户输入的密码用相应的加密算法对原程序进行加密
- 加密壳最主要的功能就是保护程序免受逆向分析，在加壳中运用了多种防止代码逆向分析的技术。
- 加密壳被大量用于对安全性要求高，且对破解敏感的应用程序，同时也会有一些恶意应用通过加密壳来躲避杀毒软件的查杀。
- 目前常用的加密壳主要有**ASProtect**、**Armadillo**、**EXECryptor**以及**Themida**等。

一. 壳的概念

- 3 加壳程序的运行流程
  - 加壳程序的运行过程中，由于壳修改了原程序的执行文件结构，从而壳代码能比原程序逻辑更早的获得控制权。
  - 加壳程序的运行流程
    - （1）保存入口参数
    - （2）获取所需要的API地址
    - （3）解密原程序的各个区块的数据
    - （4）初始化程序的IAT表
    - （5）对重定位项进行处理
    - （6）跳转到程序的原入口点

一. 壳的概念

- （1）保存入口参数
  - 壳程序会先于源程序逻辑获得控制权，执行壳部分的代码势必会改变各个寄存器的值。
  - 因此，需要先保存各寄存器的值，当壳代码执行完毕后，再将寄存器的值恢复，并开始执行源程序的逻辑。
  - 保存和恢复寄存器的值通常采用的是**pushad/popad、pushfd/popfd**指令。
    - **pushad:** 将所有的**32**位通用寄存器压入堆栈，其入栈顺序是:**EAX,ECX,EDX,EBX,ESP,EBP,ESI,EDI**
    - **Pushfd:**将**32**位标志寄存器**EFLAGS**压入堆栈

一. 壳的概念

- （2）获取所需要的API地址
  - 正常的程序导入表中存放了从外部加载的函数信息，其中包括DLL名称、函数名称、函数地址等。
  - 为了防止从这些信息猜测出来程序的功能，外壳的导入表中只有GetProcAddress、GetModuleHandle和LoadLibrary这几个函数。
    - 利用函数LoadLibrary可以将DLL文件映射到进程的地址空间
    - 函数GetModuleHandleA(W)可以获得DLL模块句柄
    - 函数GetProcAddress可以获得指定函数的真实地址。
  - 利用这些信息可以动态获取API函数的真实地址，并调用这些API函数，同时还隐藏了导入表中的函数信息。

一. 壳的概念

- （3）解密原程序的各个区块的数据
  - 加壳过程一般都会对源程序的各个块进行加密，在壳执行完毕后，为了能正确执行源程序的代码，需要对加密后的各个块进行解密。

一. 壳的概念

- （4）初始化程序的IAT表
  - 壳一般都修改了原程序文件的输入表，为了让源程序正常运行，外壳需要自己模仿Windows系统的工作来填充输入表中相关的数据。
    - 输入表（导入表）是记录PE文件中用到的dll的集合，一个dll库在输入表中占用一个元素信息的位置，一个元素信息描述了该输入dll的具体信息
    - 导入的函数就是被程序调用，但其执行代码又不在程序中的函数，这些函数的代码位于一个或者多个dll 中
    - PE 文件被装入内存时，Windows 装载器通过输入表才能将 DLL 装入

一. 壳的概念

- （4）初始化程序的IAT表
  - 有些壳程序还会将IAT表中的数据填充为HOOK-API代码的地址，这样程序在调用这些API时会先执行HOOK-API代码，可以达到监控程序行为的目的
    - IAT（Import Address Table）导入地址表
    - PE 文件被装入内存时，Windows 装载器将DLL 装入，并将调用函数的指令和函数实际所处的地址联系起来(动态连接)
    - 这就需要导入表完成，其中IAT导入地址表就指示函数实际地址。

一. 壳的概念

- （5）对重定位项进行处理
  - 程序在加载执行时，系统会按照文件声明中填写的地址，将程序载入指定内存中，这个地址称为基址
  - 对于EXE文件，Window提供给程序的基址是400000h，在这种情况下，重定位功能是不需要的
  - 为了使程序更小巧，有些加壳程序删除了重定位区块。但对于DLL动态链接库这种需要动态加载的可执行文件，需要对重定位区块重建，以保证正常运行。

一. 壳的概念

- （6）跳转到程序的原入口点
  - OEP：(Original Entry Point)，程序的入口点，软件加壳就是隐藏了OEP（或者用了假的OEP），只要找到程序真正的OEP，就可以立刻脱壳
  - 当程序运行到OEP这个位置，程序控制权会交还给原程序，一般脱壳的步骤需要寻找这个OEP点

第八章 软件脱壳技术

- 一. 壳的概念
- 二. 脱壳的相关工具
- 三. 逆向分析

北邮网安学院 崔宝江

二. 脱壳的相关工具

- 为了分析某些程序的逻辑与函数行为，首先需要对程序壳进行脱壳处理，以暴露出其实际执行的功能代码。

二. 脱壳的相关工具

- 1 查壳工具
- 2 内存Dump工具
- 3 输入表重建工具

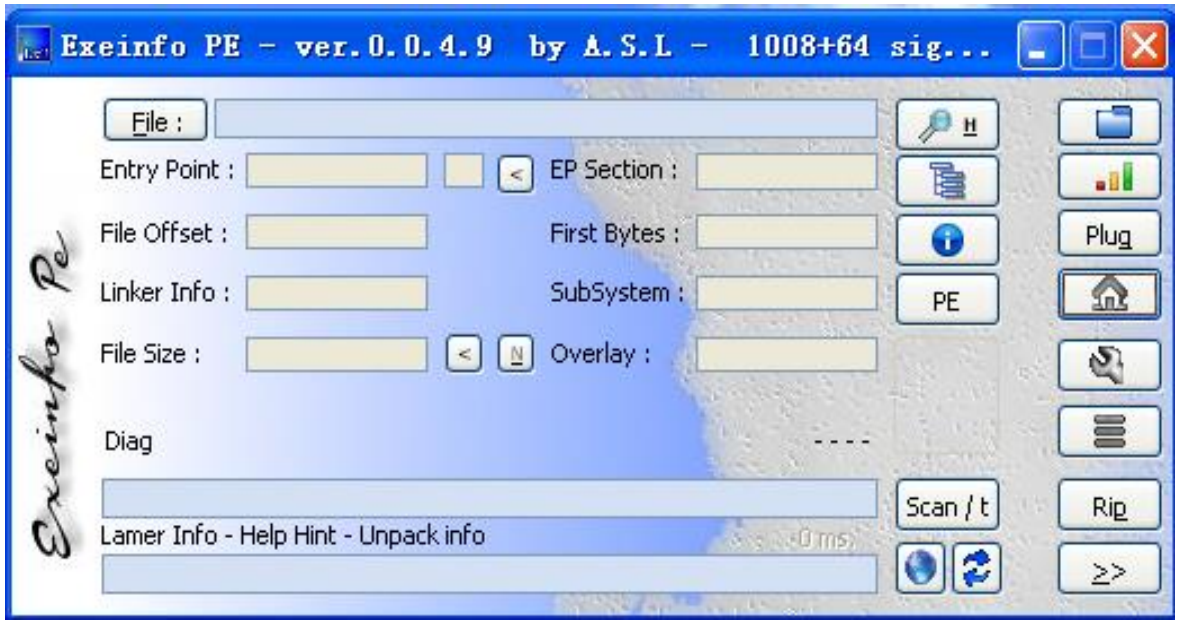
二. 脱壳的相关工具

- 1 查壳工具
  - Exeinfo PE
  - PEiD

二. 脱壳的相关工具

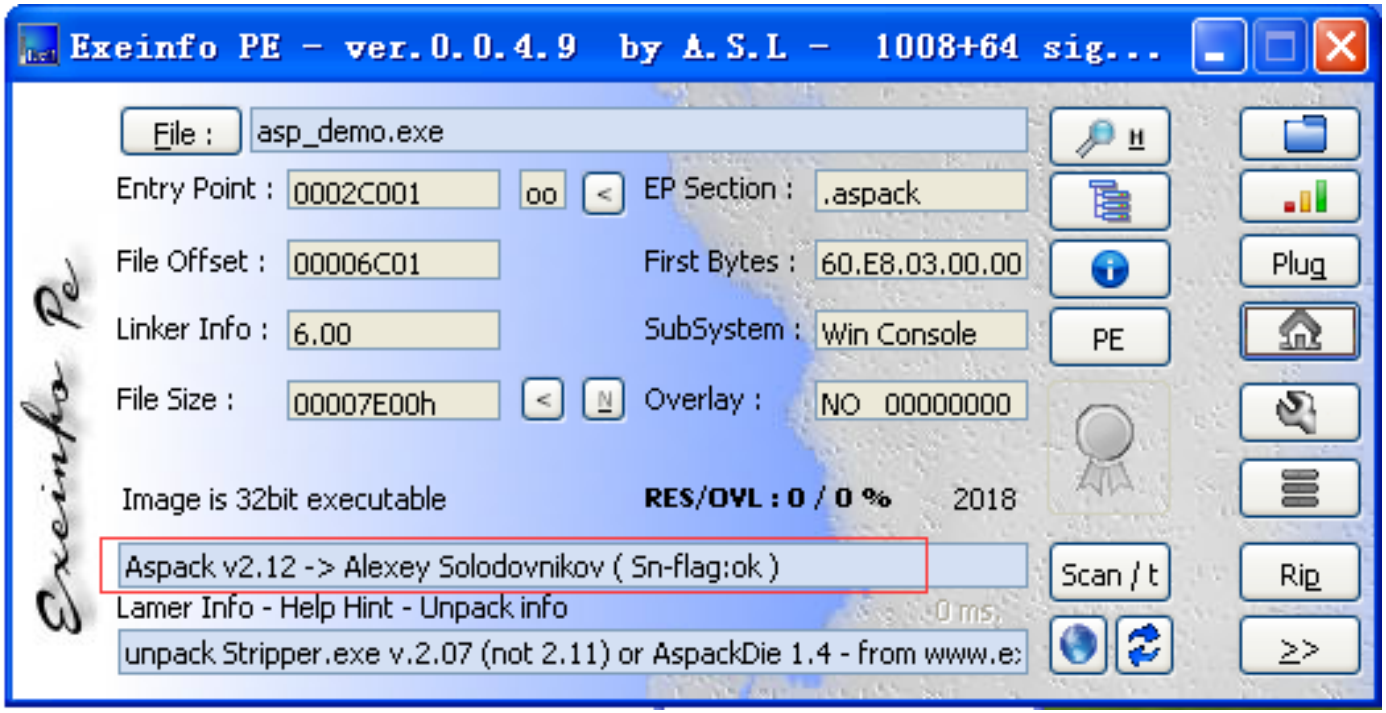
Exeinfo PE

- Exeinfo PE 是一款免费的Win32可执行程序检查器，可以检测到加壳程序相关信息，该工具也是一款图形化工具



二. 脱壳的相关工具

- 直接将要检测的程序拖入**ExeinfoPE**中即可，该工具会将程序的关键信息展示出来
  - 包括程序入口点、程序入口点所在段、文件偏移以及关注的加壳信息

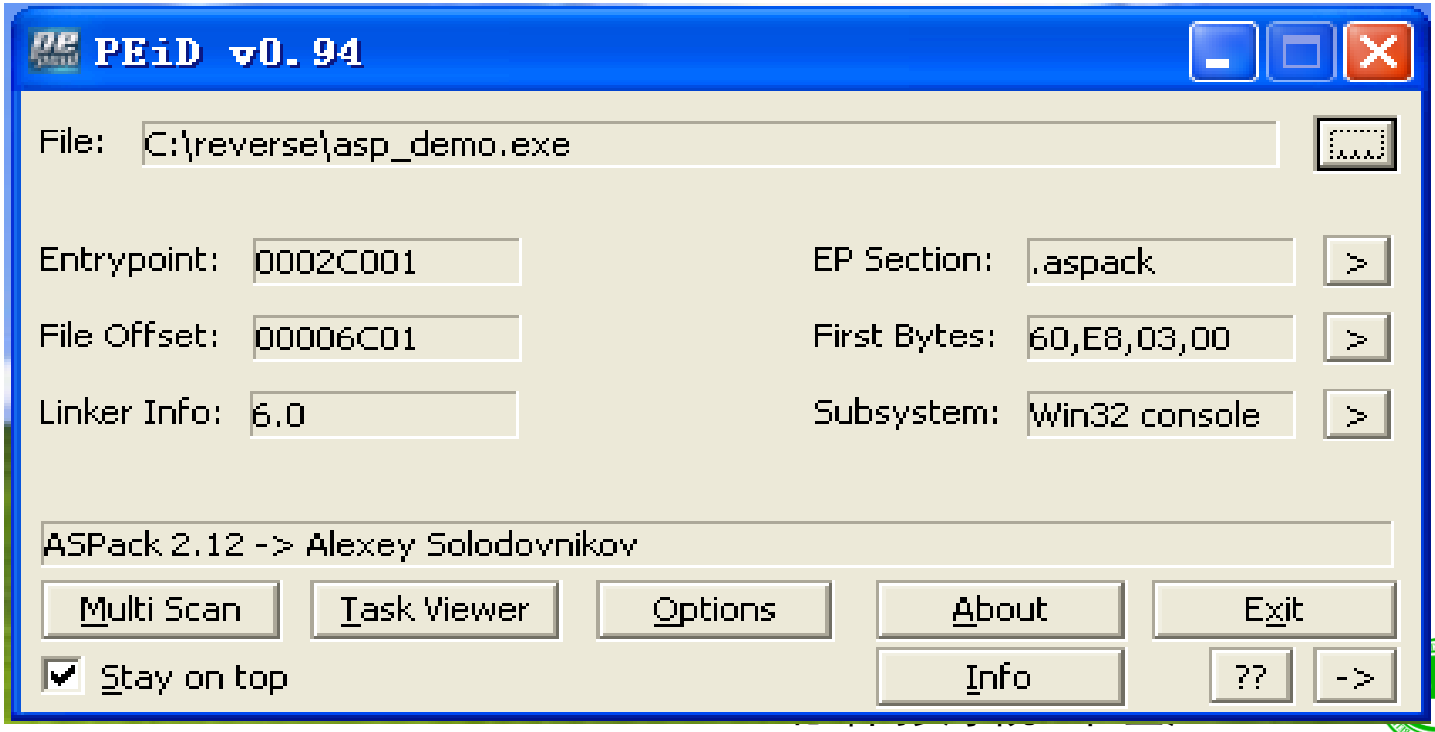


二. 脱壳的相关工具

- PEiD（PE Identifier）
  - PEiD是一款著名的查壳工具，能够查出来大多数的压缩壳、加密壳以及程序的编译器等信息
  - 能够检测出来超过470种不同PE文件的签名信息

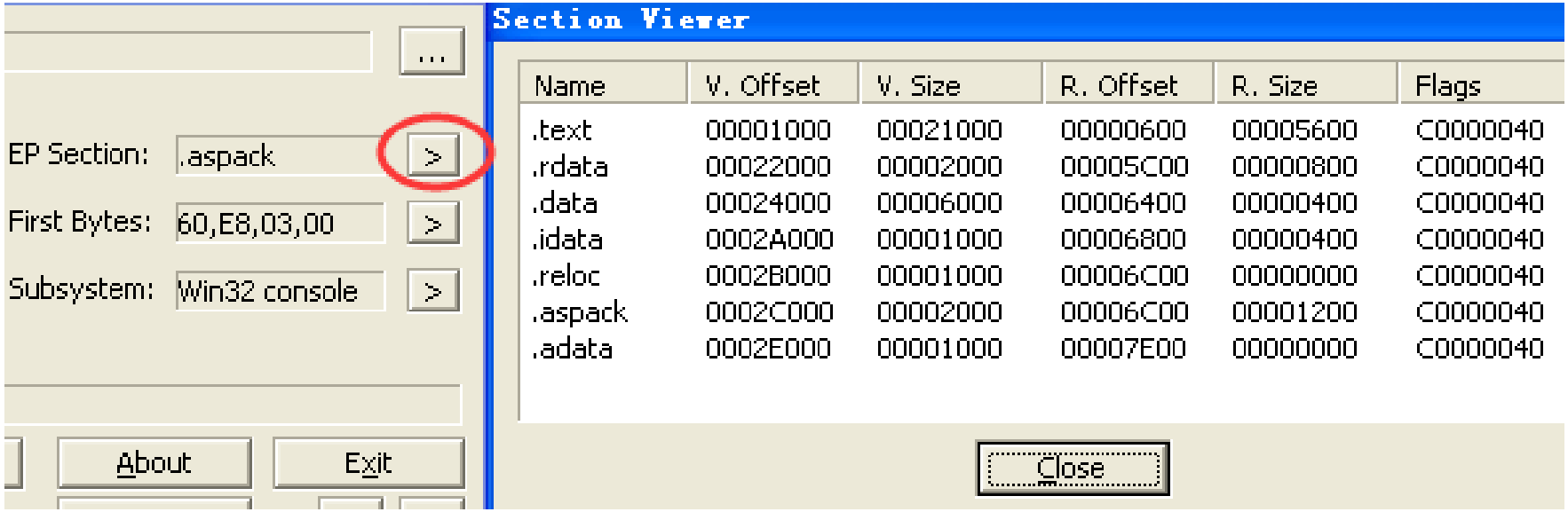
二. 脱壳的相关工具

- PEiD主界面同样会展示出来程序的入口点、入口点所在段、文件偏移以及加壳信息等
- 可以看到下面程序的加壳信息为**ASPack 2.12**



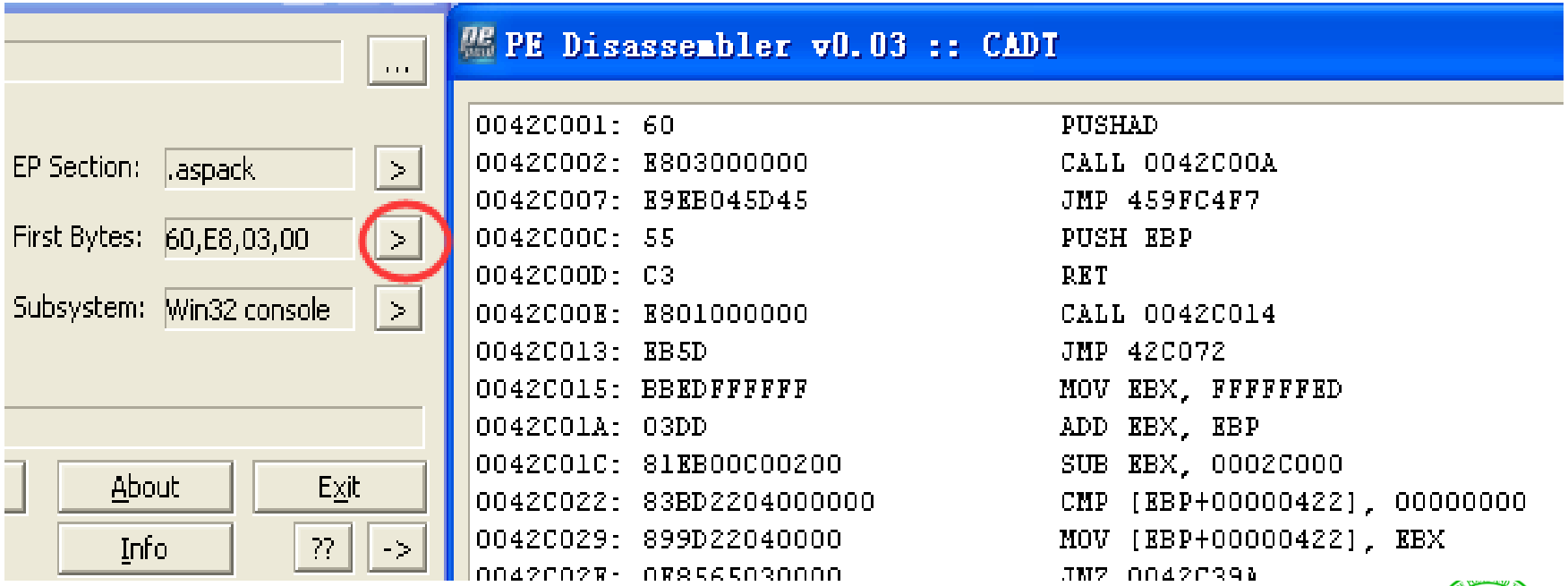
二. 脱壳的相关工具

- 如果需要查看程序的段信息，只需点击EP Section右边的”>”即可



二. 脱壳的相关工具

- PEiD还支持简单的反汇编功能，点击下图所示的按钮即可查看反汇编结果



二. 脱壳的相关工具

- 2 内存Dump工具
  - 内存Dump原理
  - LordPE
  - OllyDump

二. 脱壳的相关工具

2 内存Dump工具

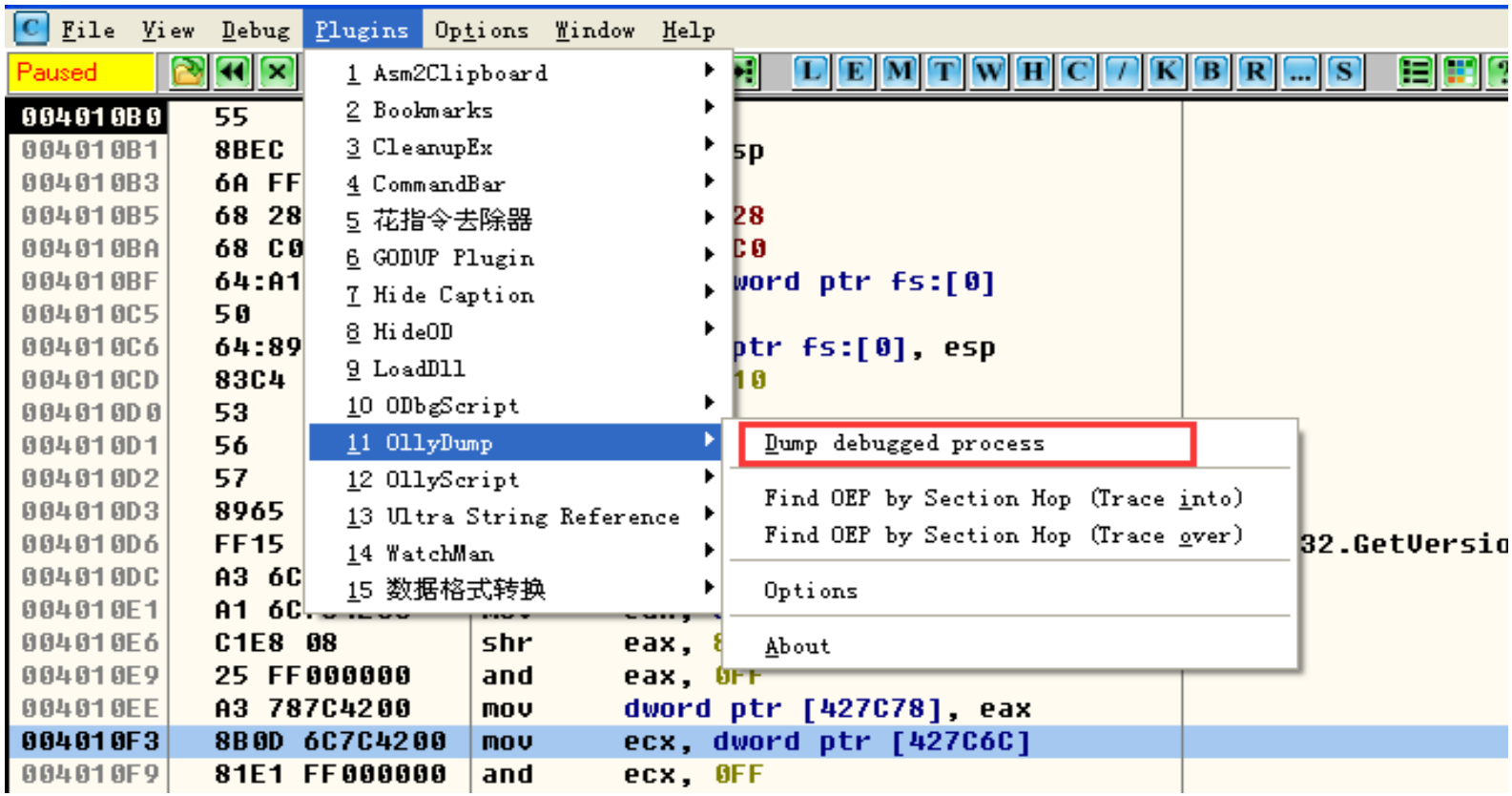
内存Dump原理

- 在执行到原程序入口点后，外壳程序已经将原程序的各个段以及导入表等数据都恢复完成
- 为了能够调试分析该程序， 要把程序在内存空间的数据都导出来，这就是Dump，生成的文件称为dump 文件
- 要得到正在运行进程的内存数据，需要获取到进程的相关信息，然后在从该进程中读取内存数据并保存到文件中
- 获取进程的信息可以采用Module32Next函数，该函数会返回一个指向MODULEENTRY32结构体的指针，利用该结构体中的modBaseAddr、modBaseSize、hModule等字段，再结合ReadProcessMemory函数可以从该进程中直接读取内存数据，实现内存Dump的功能

二. 脱壳的相关工具

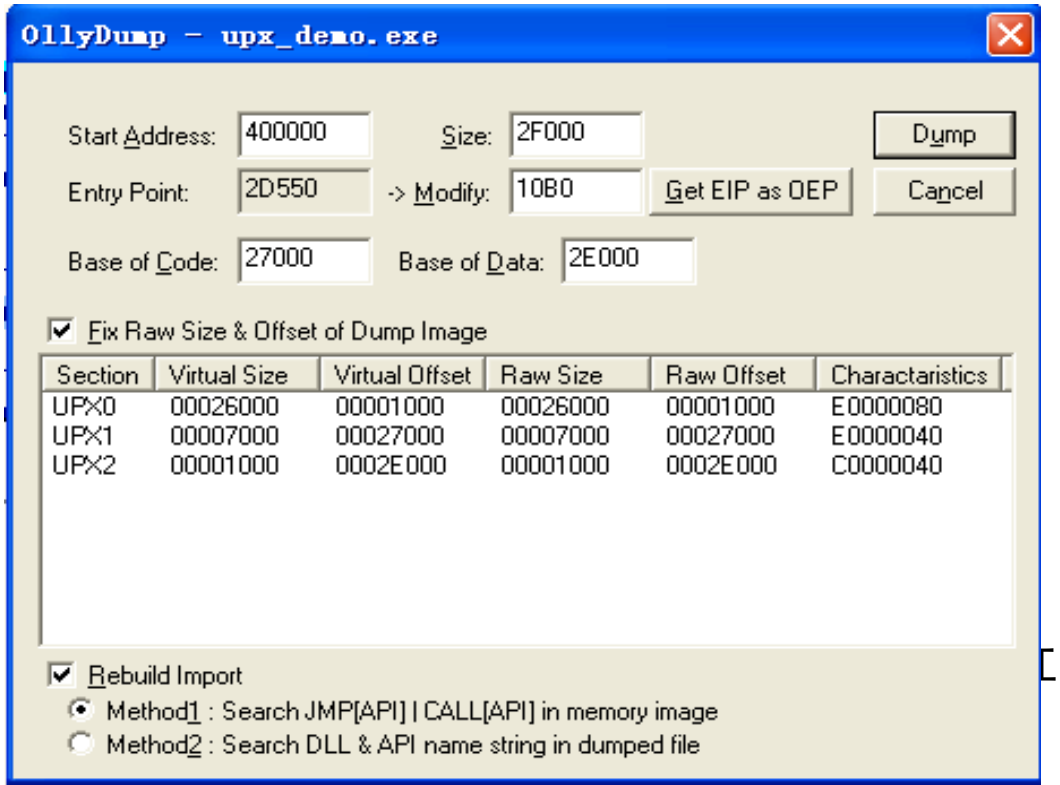
- **OllyDump**
  - **OllyDump为调试工具OllyDbg的一个插件，利用该插件可以完成内存Dump的功能。**
  - **依次点击OllyDbg中的Plugins→OllyDump→Dump debugged process，便可启动该插件**

二. 脱壳的相关工具


File View Debug Plugins Options Window Help
Paused
1 Asm2Clipboard
2 Bookmarks
3 CleanupEx
4 CommandBar
5 花指令去除器
6 GODUP Plugin
7 Hide Caption
8 HideOD
9 LoadDll
10 ODbgScript
11 OllyDump
12 OllyScript
13 Ultra String Reference
14 WatchMan
15 数据格式转换
Dump debugged process
Find OEP by Section Hop (Trace into)
Find OEP by Section Hop (Trace over)
Options
About
004010B0 55
004010B1 8BEC
004010B3 6A FF
004010B5 68 28
004010BA 68 C0
004010BF 64:A1
004010C5 50
004010C6 64:89
004010CD 83C4
004010D0 53
004010D1 56
004010D2 57
004010D3 8965
004010D6 FF15
004010DC A3 6C
004010E1 A1 6C
004010E6 C1E8 08 shr eax,
004010E9 25 FF000000 and eax, 0FF
004010EE A3 787C4200 mov dword ptr [427C78], eax
004010F3 8B0D 6C7C4200 mov ecx, dword ptr [427C6C]
004010F9 81E1 FF000000 and ecx, 0FF

二. 脱壳的相关工具

- 启动OllyDump后，可以手动输入Dump的起始地址、映像大小、程序入口点等信息，同时可以选择是否修复导入表等，点击Dump按钮即可完成内存Dump



二. 脱壳的相关工具

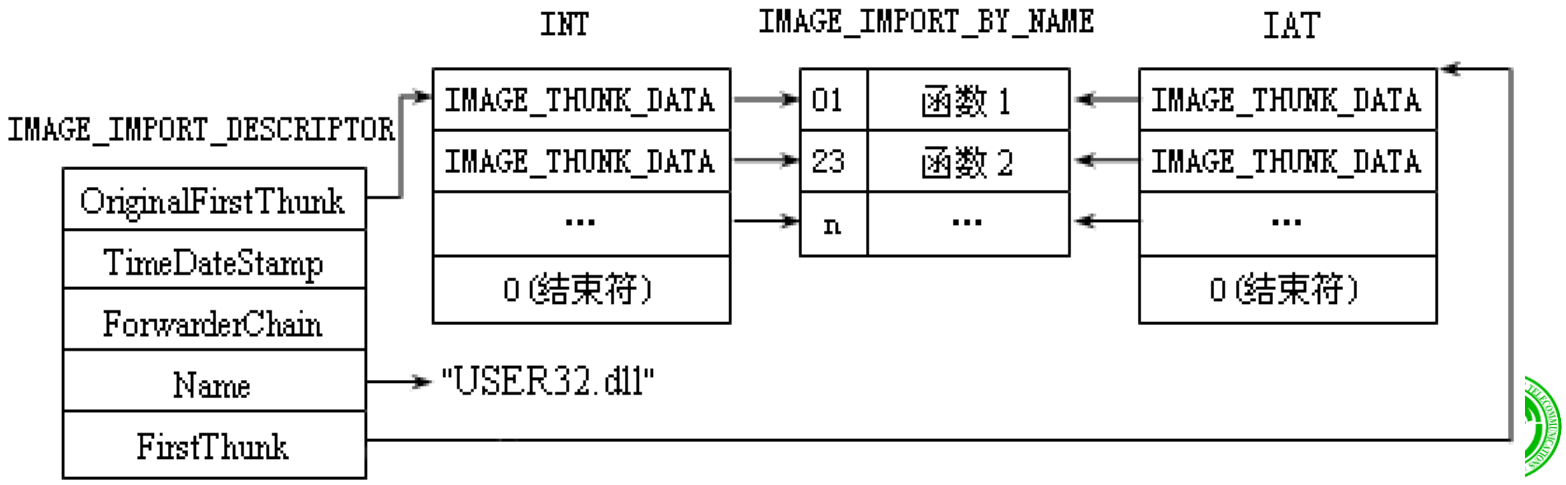
- 3 输入表重建工具
  - 通过内存Dump的方式将程序的内存映像保存下来之后，一般情况下是无法直接运行这个Dump下来的程序的，一般的加密壳都会破坏掉源程序的输入表，为了让程序正常运行，需要对输入表进行重建

二. 脱壳的相关工具

- 输入表修复原理
  - PE文件在运行时一般都会用到从外部DLL导入的函数，在编译程序时是无法事先获得这些外部函数的真实地址的。
  - 为了获得这些外部函数在内存中的地址，需要在程序装载时将这些函数的地址查询出来并保存起来
  - PE文件中的输入表就是负责保存程序用到了哪些DLL文件中的哪些函数，以及这些函数的真实地址

二. 脱壳的相关工具

- 输入表的结构
  - 其中，保存函数真实地址的数据结构就是**Import Address Table（IAT），外壳程序在处理的过程中会模拟Windows装载器来获取函数的真实地址并将其填充到IAT中，也就是说外壳程序处理完毕后，整个程序的内存中存在着一个完整的IAT**



二. 脱壳的相关工具

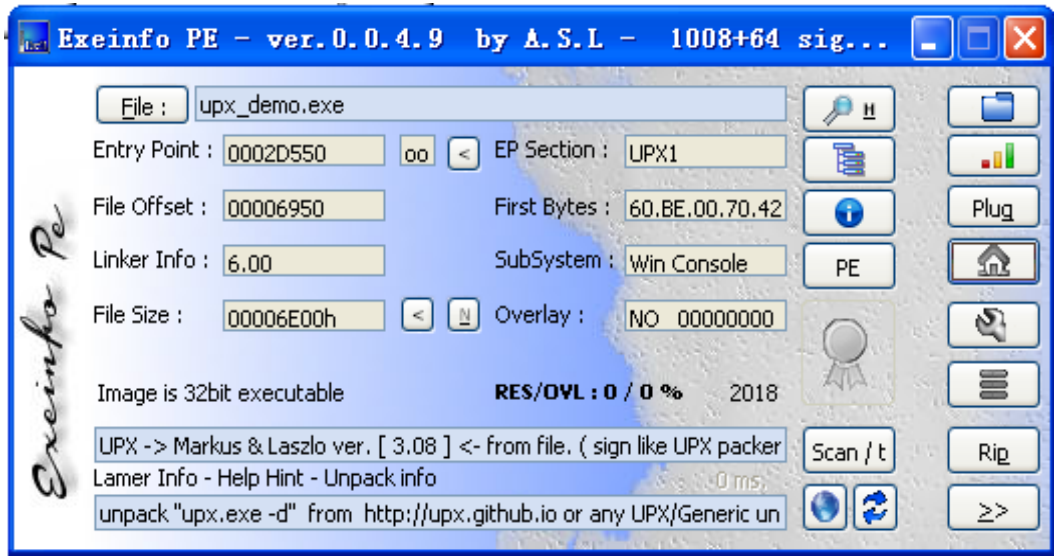
- 输入表修复的原理
  - 输入表修复的原理就是找到内存中存在的IAT，根据该IAT来重新构建一个完整的输入表结构
  - 构建完成后，程序再次运行就能通过该结构正常填充IAT中的数据，使得程序正常运行。

二. 脱壳的相关工具

- ImportREC重建工具
  - ImportREC是一款专业的输入表重建工具，可以根据内存中的IAT重新构建一个输入表。
  - 根据重建输入表的原理，需要读取目标进程的内存并找到IAT，根据该IAT重新构建输入表中的其他数据结构。
  - 使用ImportREC时，目标进程需要处于运行中

二. 脱壳的相关工具

- 修复输入表的例子
  - 示例程序为upx_demo.exe，通过查壳工具可以知道是采用UPX加壳



二. 脱壳的相关工具

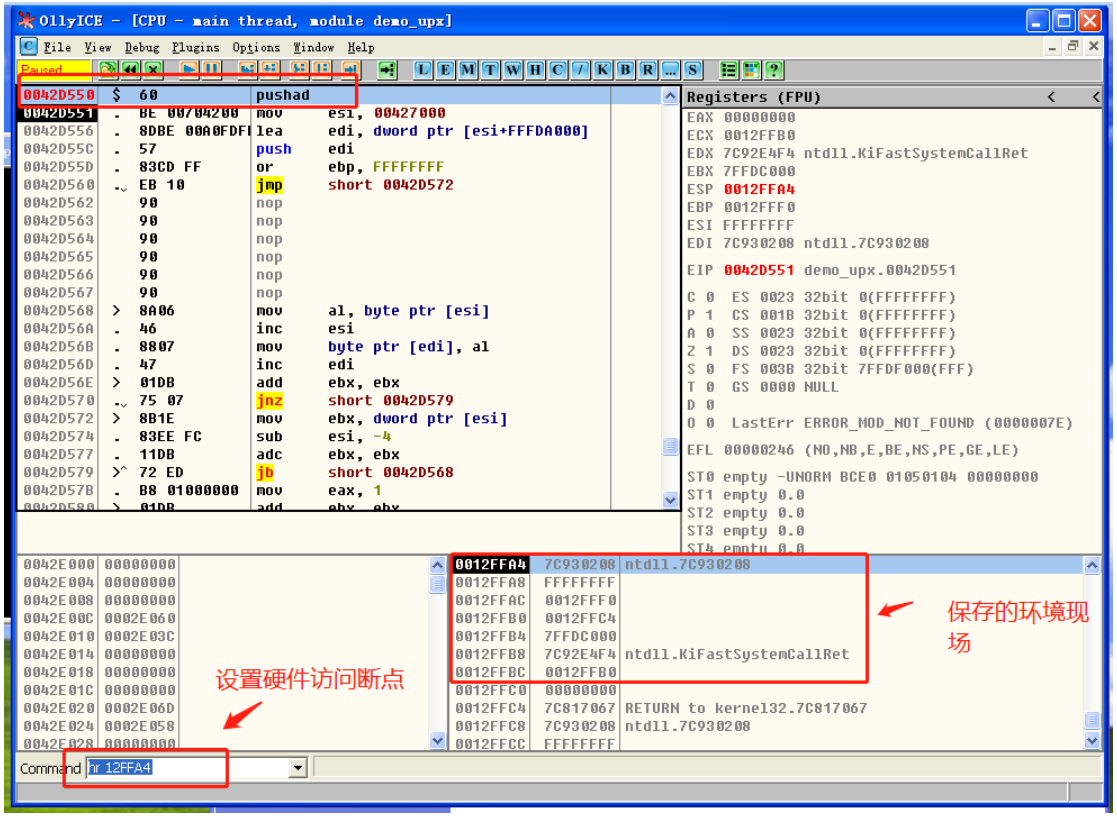
- 首先需要找到程序的原始入口点（OEP）
  - 利用栈平衡原理（ESP守恒定律）
    - 加壳软件，必须保证外壳初始化的现场环境（寄存器）与原程序的现场环境相同。
    - 加壳程序初始化时保存各寄存器的值，外壳执行完毕，再恢复各寄存器内容，最后再跳到原程序执行。
    - 程序通常使用pushad/popad、pushfd/popfd指令来保存与恢复现场环境。

二. 脱壳的相关工具

- 根据堆栈平衡原理，先执行完**pushad**指令后，在栈地址**0x12FFA4**处设置硬件访问断点（命令：**hr 12FFA4**）
  - 地址**0x12ffa4**就是在**push ad**指令执行后的栈顶地址，也就是说这个地址处存放的是某一个寄存器的指令
  - 设置断点**hr 12ffa4**以使程序在读取这个地址时会触发断点
  - 当程序读取这个值的时候，就说明程序已经执行完了壳程序的部分，所以触发断点的位置就在源程序入口处附近，这样就能快速的跟踪到程序入口

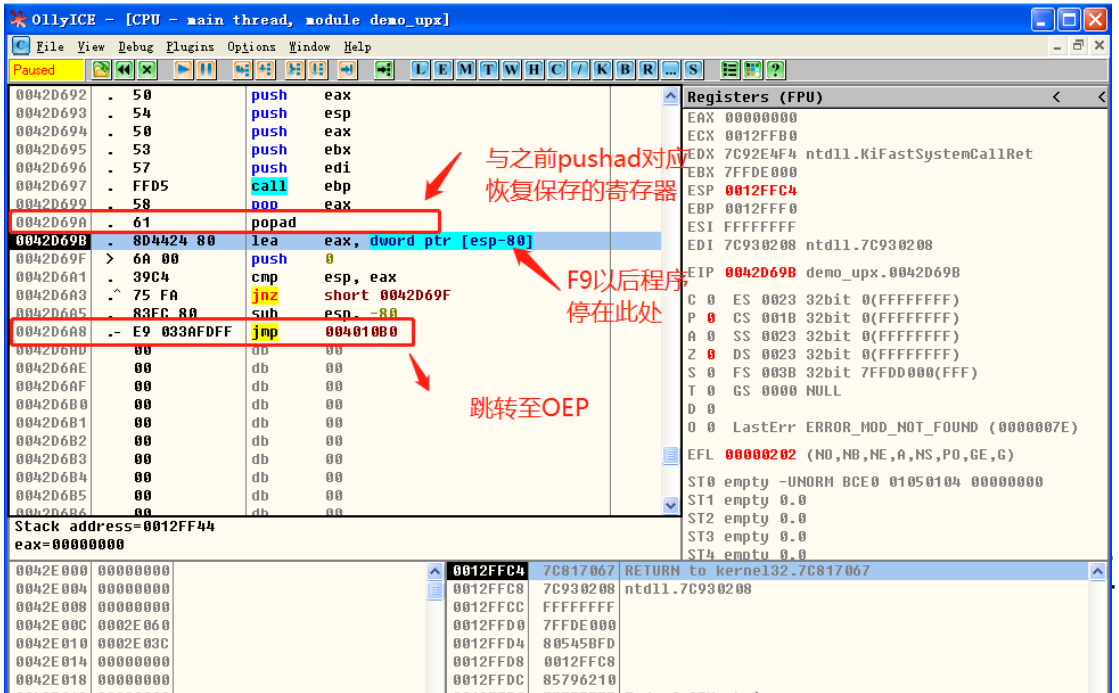
二. 脱壳的相关工具

❒利用堆栈平衡找OEP



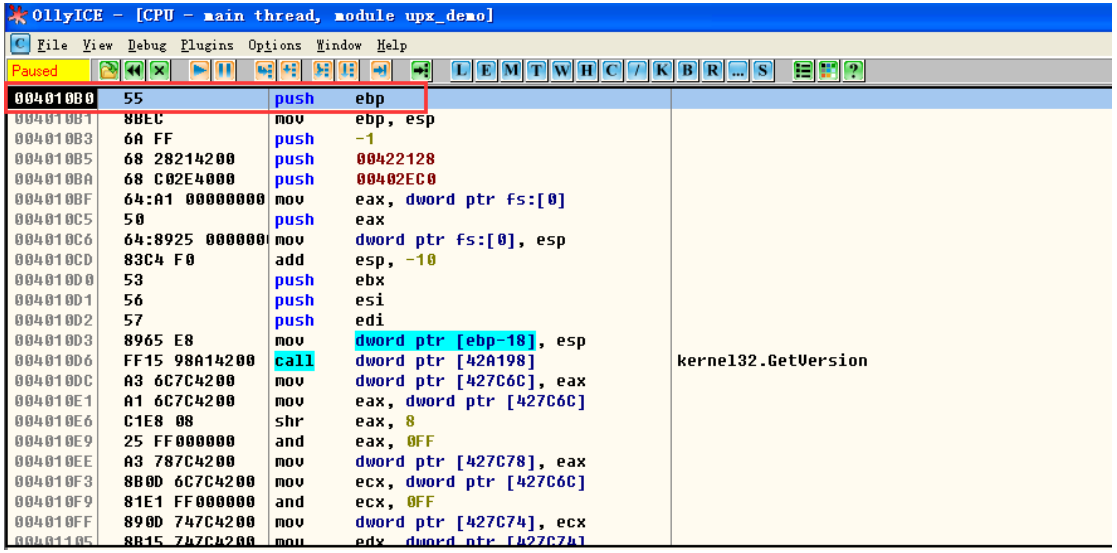
二. 脱壳的相关工具

- 按F9让程序运行，程序断在恢复保存的现场环境以后（即popad指令后），此操作结束后，程序会跳转至OEP
- 跳转到jmp

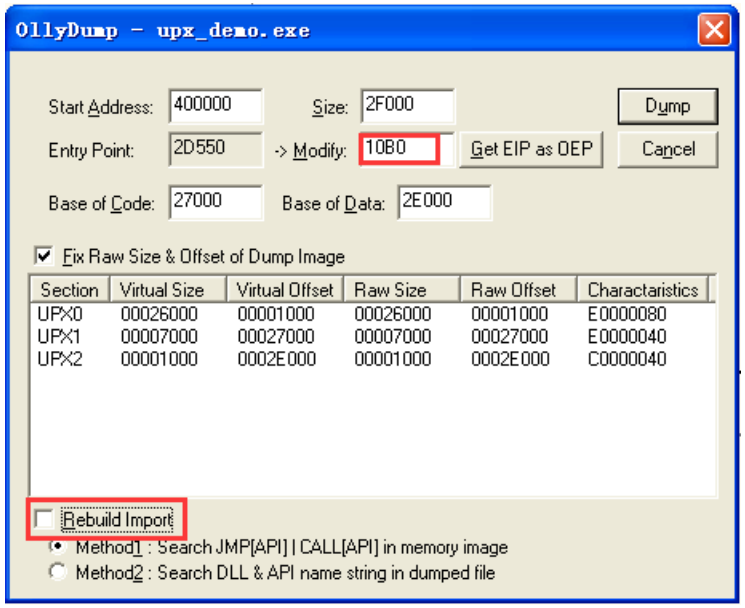


二. 脱壳的相关工具

- 找到程序入口点为**0x4010B0**，并使用**Dump**内存工具**OllyDump**将内存映像**Dump**到文件中（取消勾选重建输入表选项）

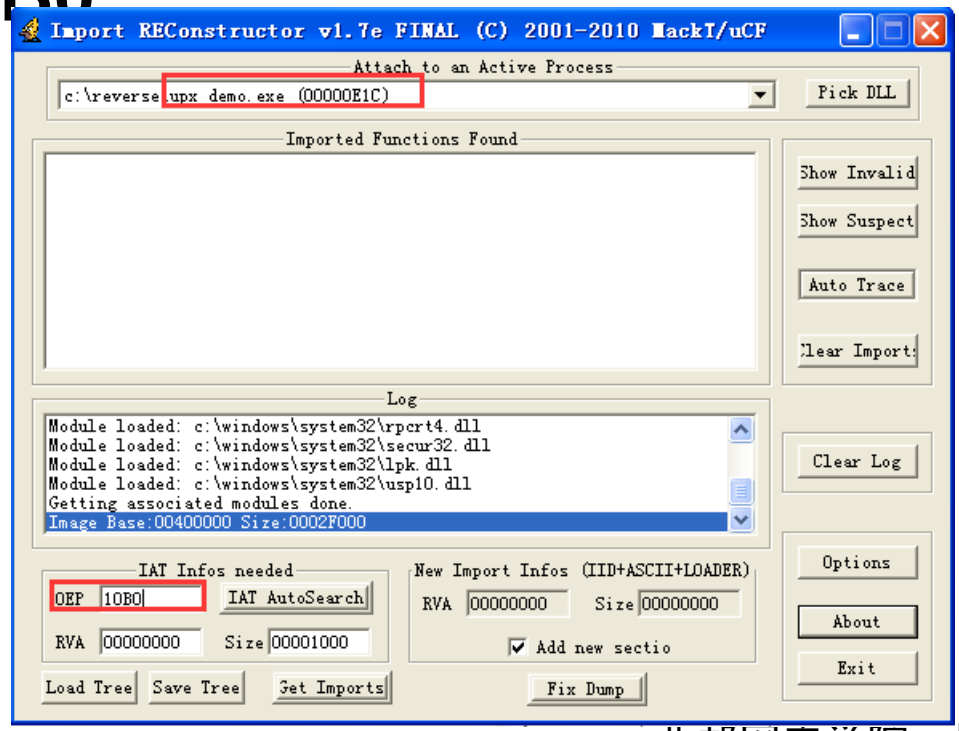


二. 脱壳的相关工具


OllyDump - upx_demo.exe
Start Address: 400000
Size: 2F000
Dump
Entry Point: 2D550
-> Modify: 10B0
Get EIP as OEP
Cancel
Base of Code: 27000
Base of Data: 2E000
Fix Raw Size & Offset of Dump Image
Section Virtual Size Virtual Offset Raw Size Raw Offset Charactaristics
UPX0 00026000 00001000 00026000 00001000 E0000080
UPX1 00007000 00027000 00007000 00027000 E0000040
UPX2 00001000 0002E000 00001000 0002E000 C0000040
Rebuild Import
Method1 : Search JMP[API] | CALL[API] in memory image
Method2 : Search DLL & API name string in dumped file

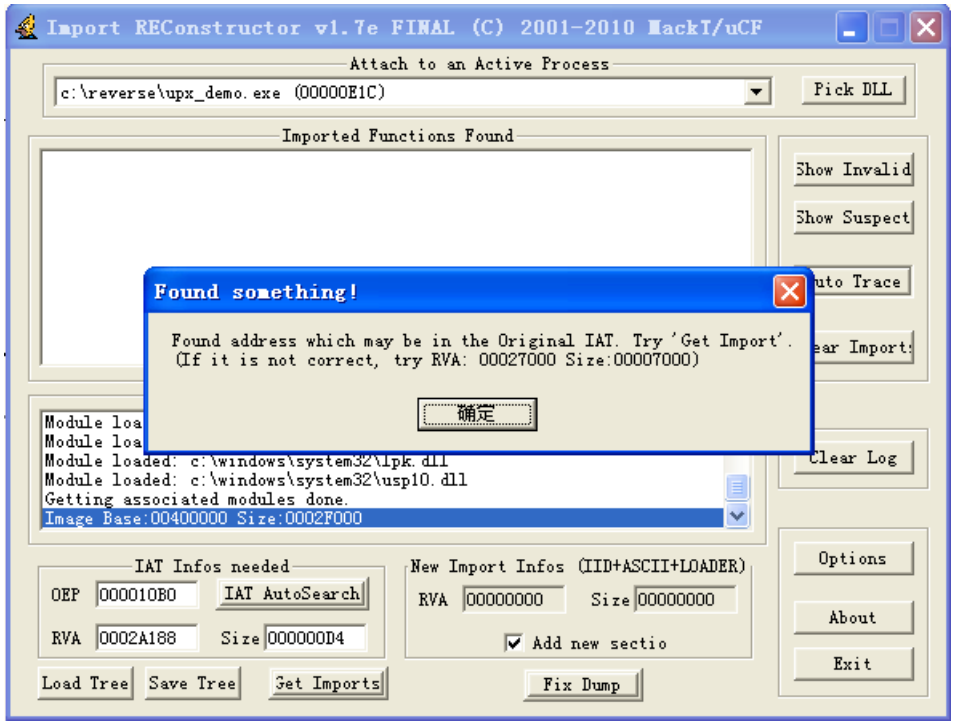
二. 脱壳的相关工具

- 打开ImportREC工具，选中目标进程upx_demo.exe，并将OEP处填为我们得到的0x10B0



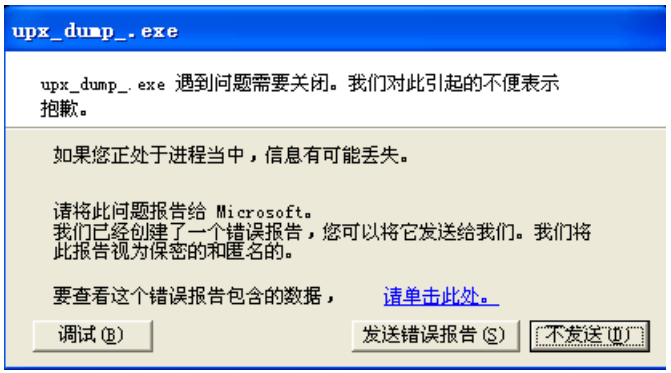
二. 脱壳的相关工具

- 点击按钮**IATAutoSearch**后工具就会查找内存中的**IAT**数据，如果出现“**Found address which may be the Original IAT.Try‘Get Import’**”，就说明输入的**OEP**起作用了



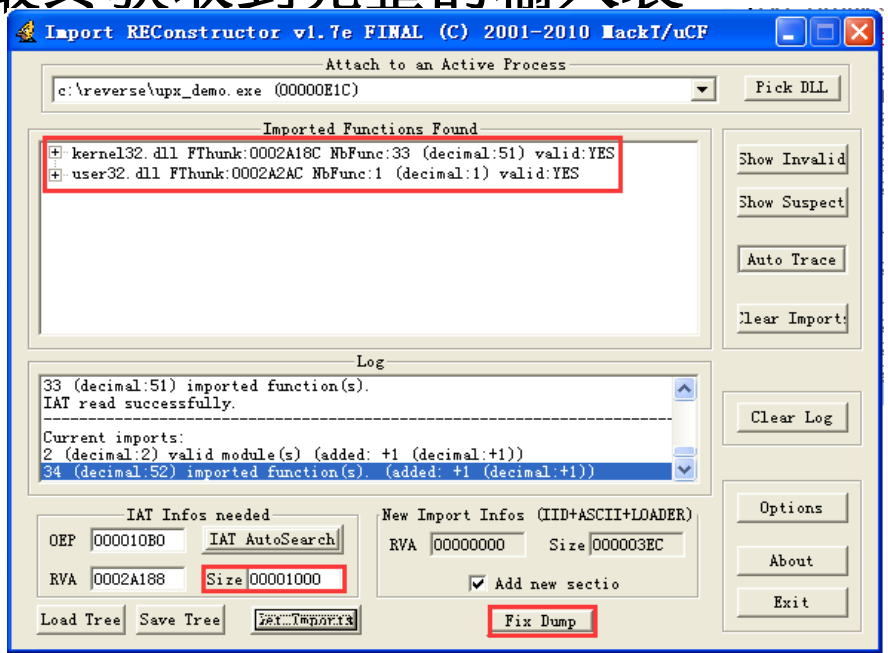
二. 脱壳的相关工具

- 输入表中每一个DLL都有与之对应的IAT，一般情况下IAT之间的间隔为一个DWORD的0，但是有些情况下他们之间的间隔会发生变化，这样ImportREC不能获取完整的IAT表，只能获取到其中的第一份IAT，如果仅以此IAT进行重建，程序仍然不能正常运行，依次点击Get Import→Fix Dump，并选择最开始Dump下来的文件，会发现仍然不能正常运行



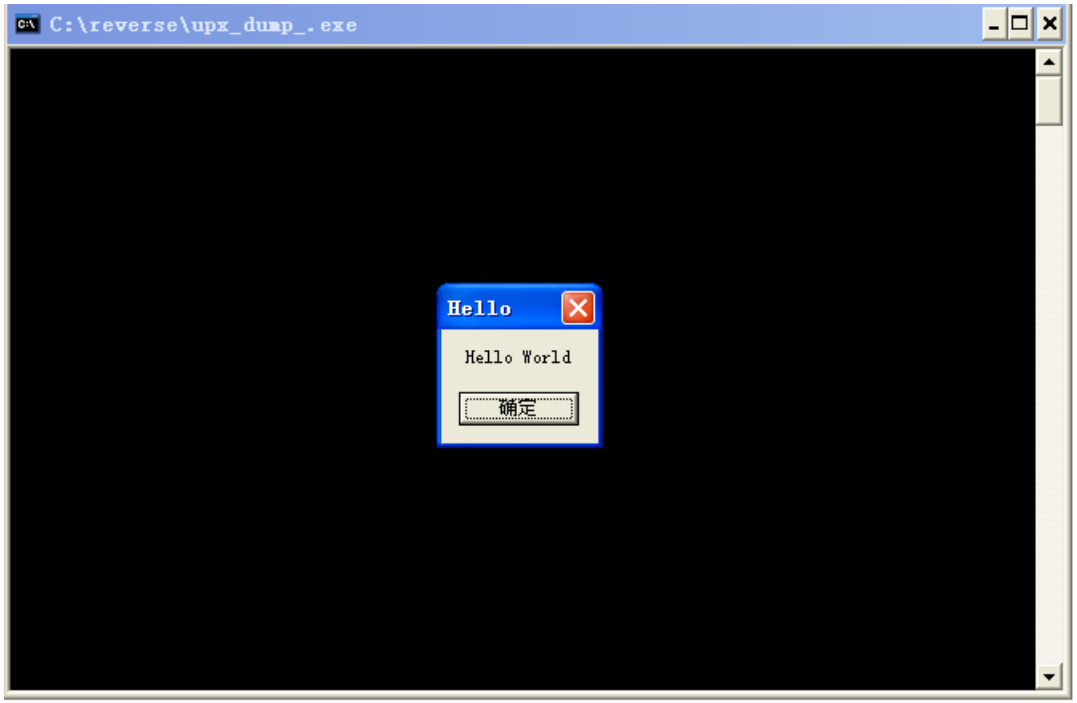
二. 脱壳的相关工具

- 这时有两种方法来解决这个问题：依次填写剩余IAT的地址和大小到RVA（手工找），Size字段中，并点击“GetImport”，重复该过程直到所有的IAT均搜索完毕即可；也可以修改Size字段，使其值足够覆盖所有的IAT。这里通过将Size修改为0x1000（保证包含了所有IAT表，2000也行），最终获取到完整的输入表



二. 脱壳的相关工具

- 最终修复后的PE文件正常运行



第八章 软件脱壳技术

- 一. 壳的概念
- 二. 脱壳的相关工具
- 三. 逆向分析

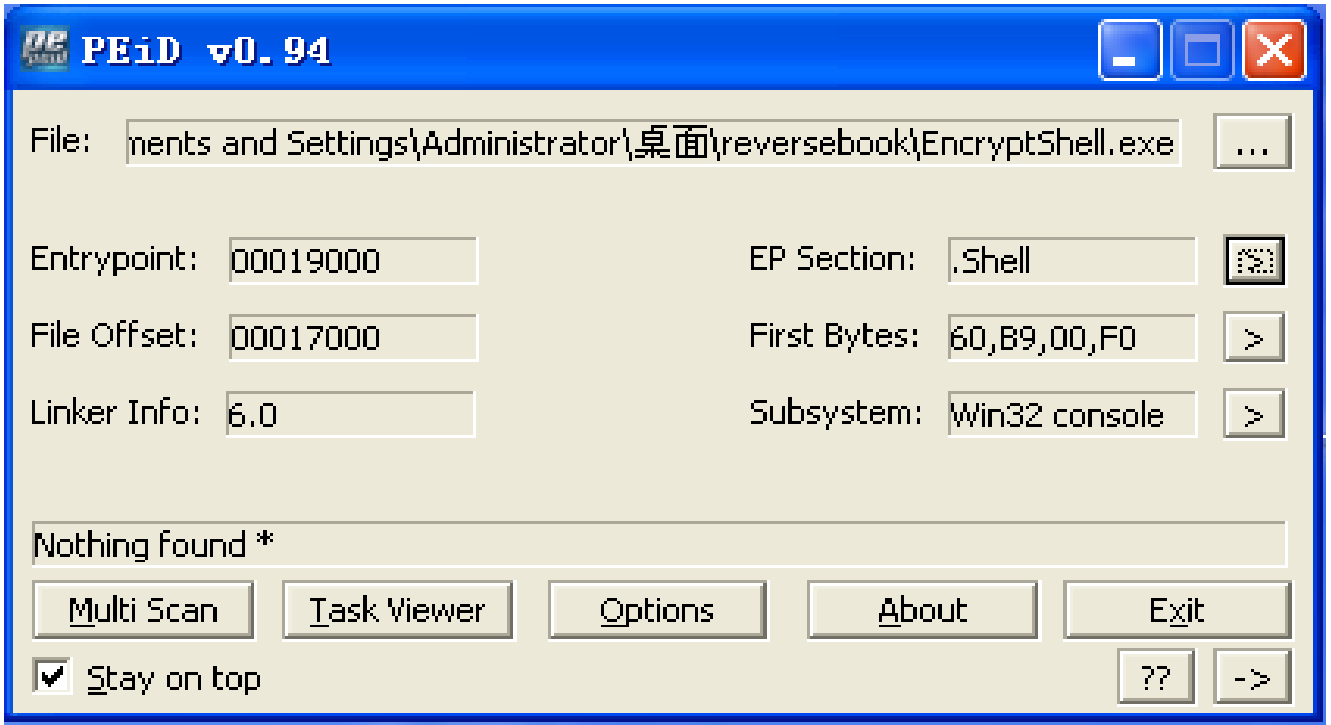
北邮网安学院 崔宝江

三. 逆向分析

- 1例题讲解
- 2简单壳的手动实现

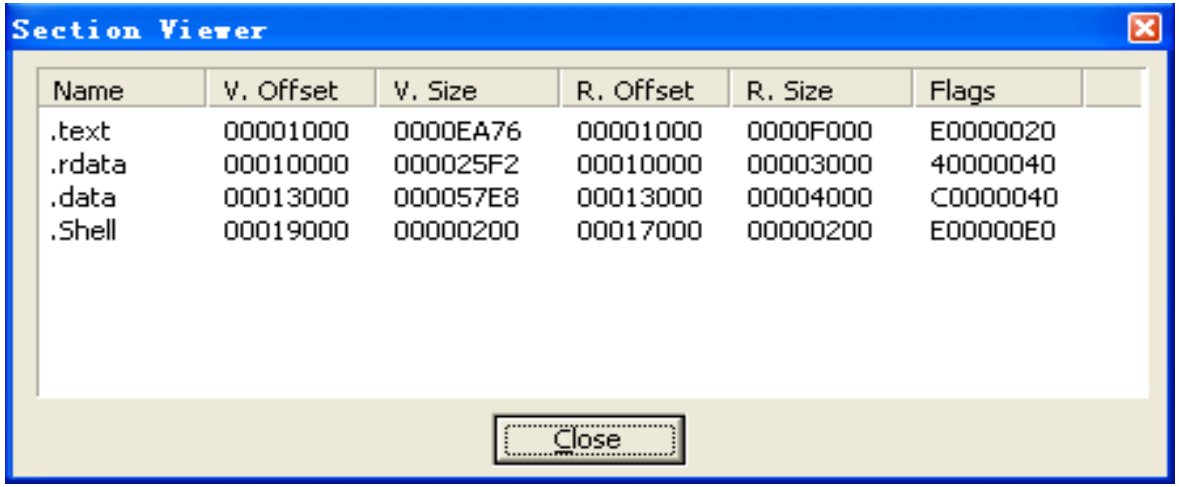
1 例题讲解

- 以加了一层加密壳的例题来练习
  - 可以用工具**PEiD**来查看该程序被加了什么壳
  - 也可以用**Exeinfo PE**来查壳，结合两个工具来比较分析



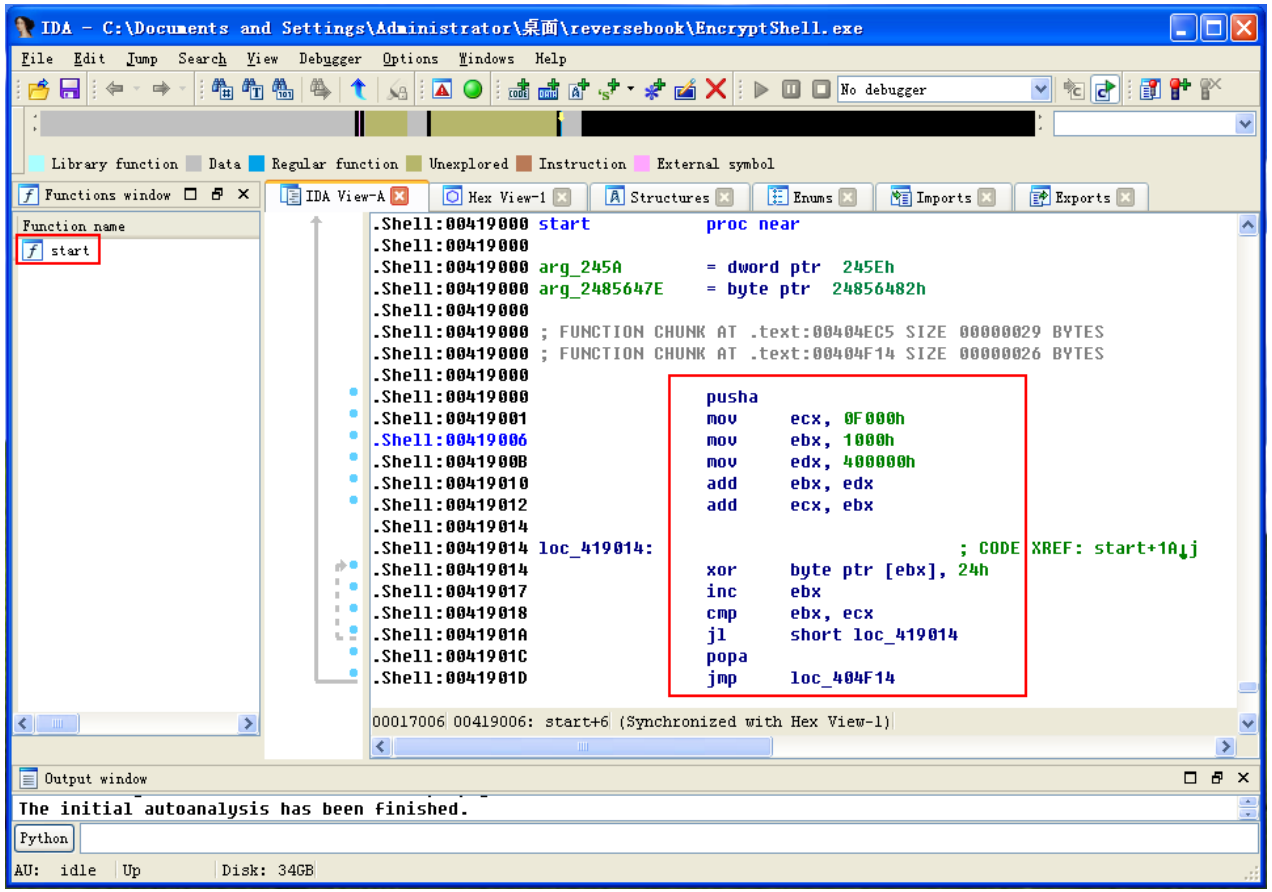
1 例题讲解

- 没有查出壳，但查看程序的区段，发现除了**.text**段、**.rdata**段、**.data**段之外，还多了一个奇怪的**.Shell**段



1 例题讲解

- 用IDA加载程序，可以看到IDA只识别出了一个start函数

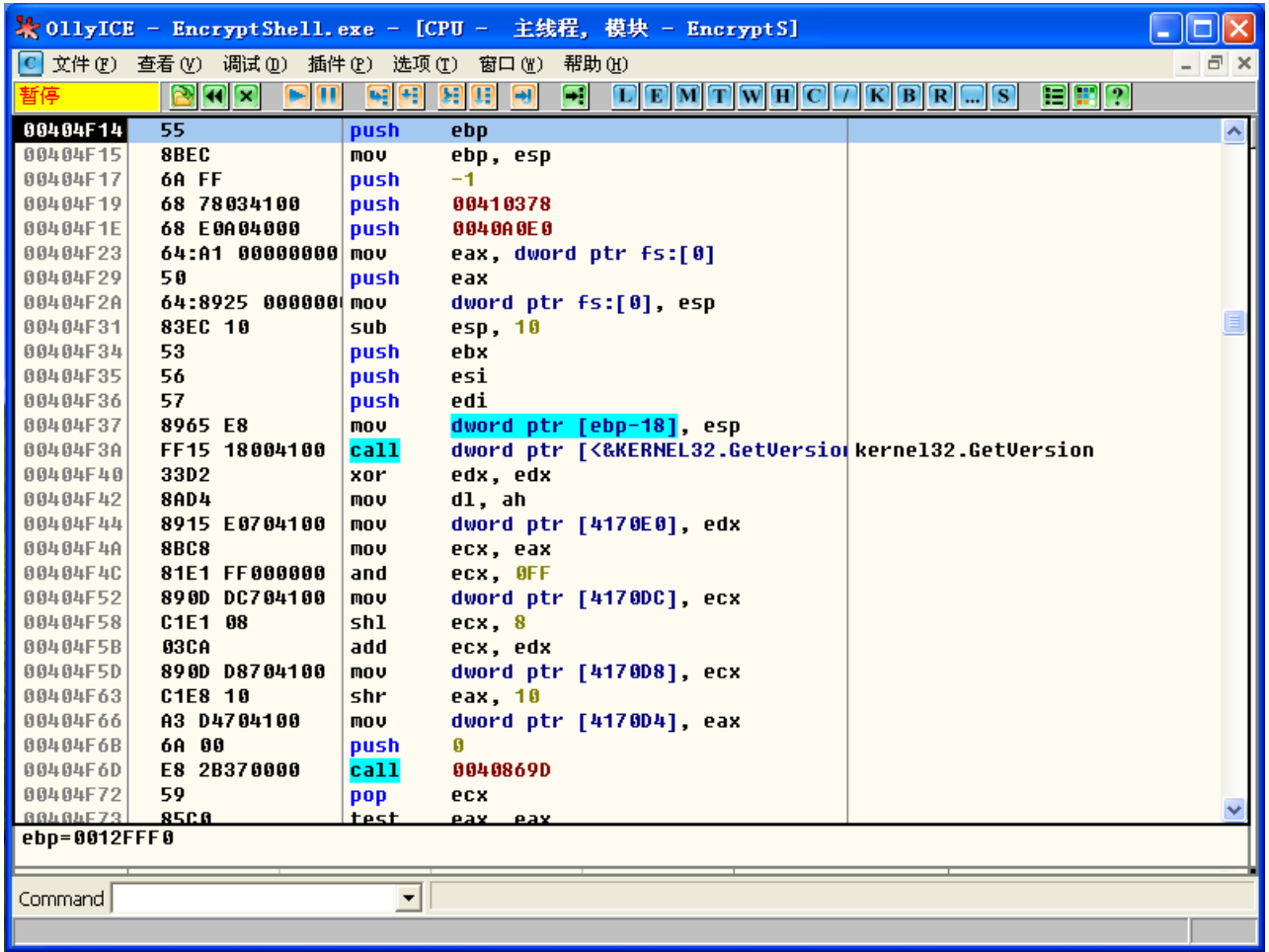


1 例题讲解

- 根据栈平衡原理，可以清楚的知道这里的pushad/popad指令是用来保存和恢复寄存器的值
- 这两条指令中间的代码片段，其实是在完成循环解密text段的内容
- 解密完后，通过jmp指令跳转到程序的真正入口处。

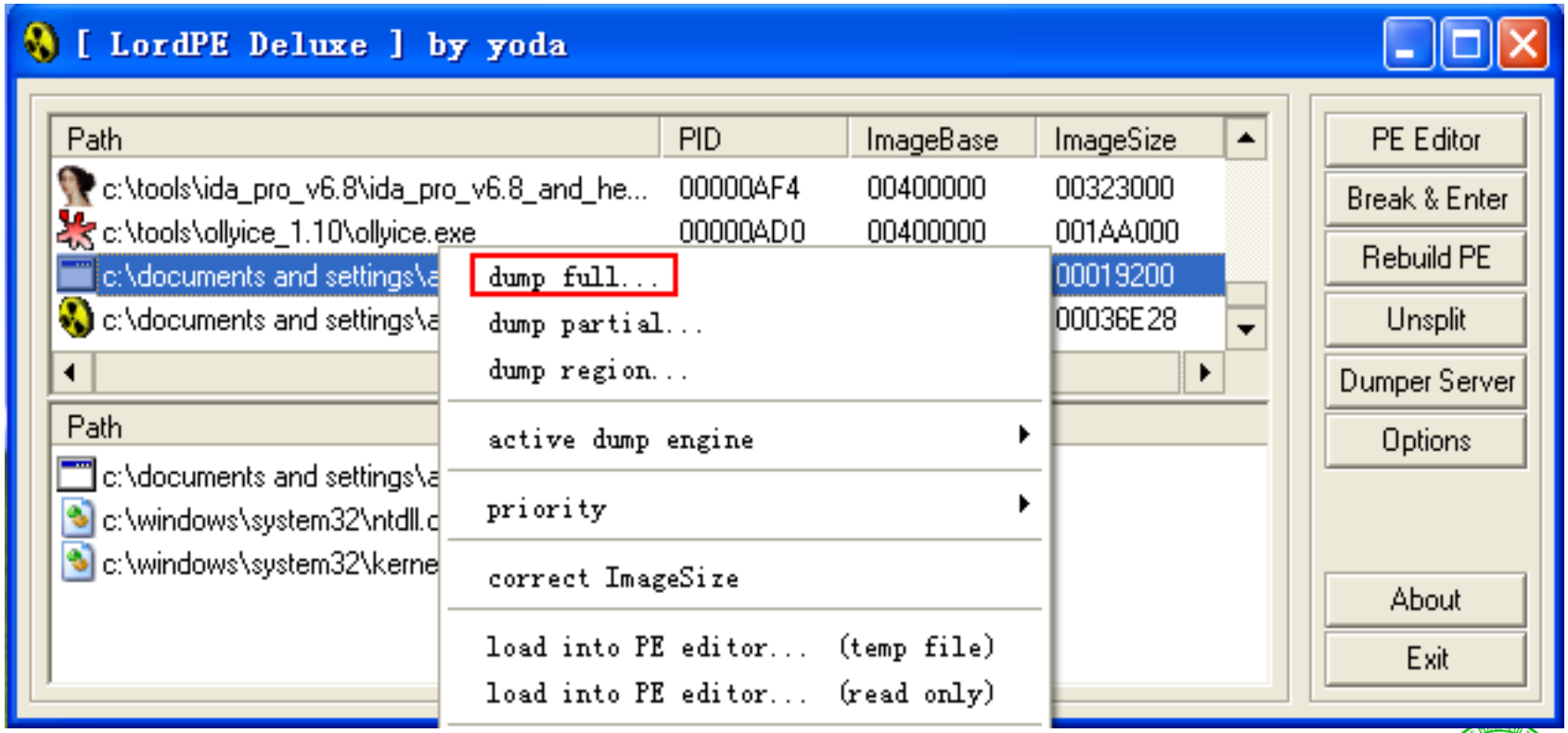
1 例题讲解

- 用OD调试到程序的OEP处



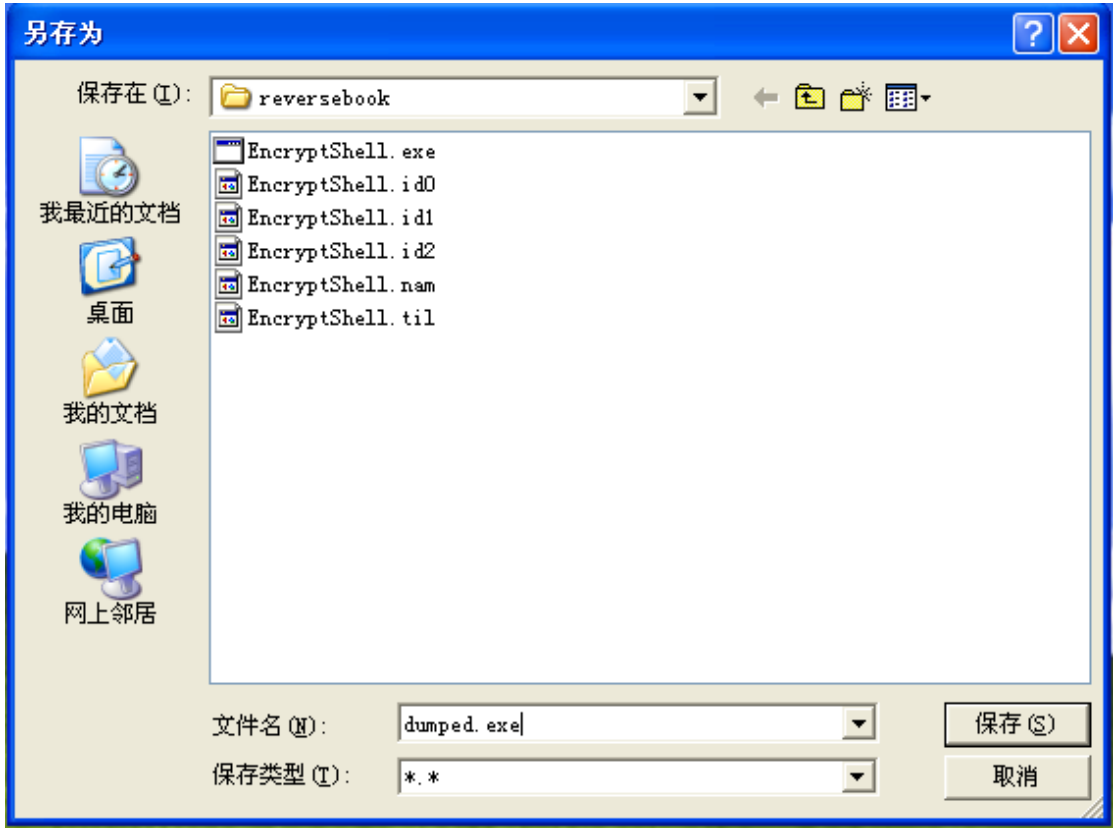
1 例题讲解

- 然后使用LordPE工具dump程序内存



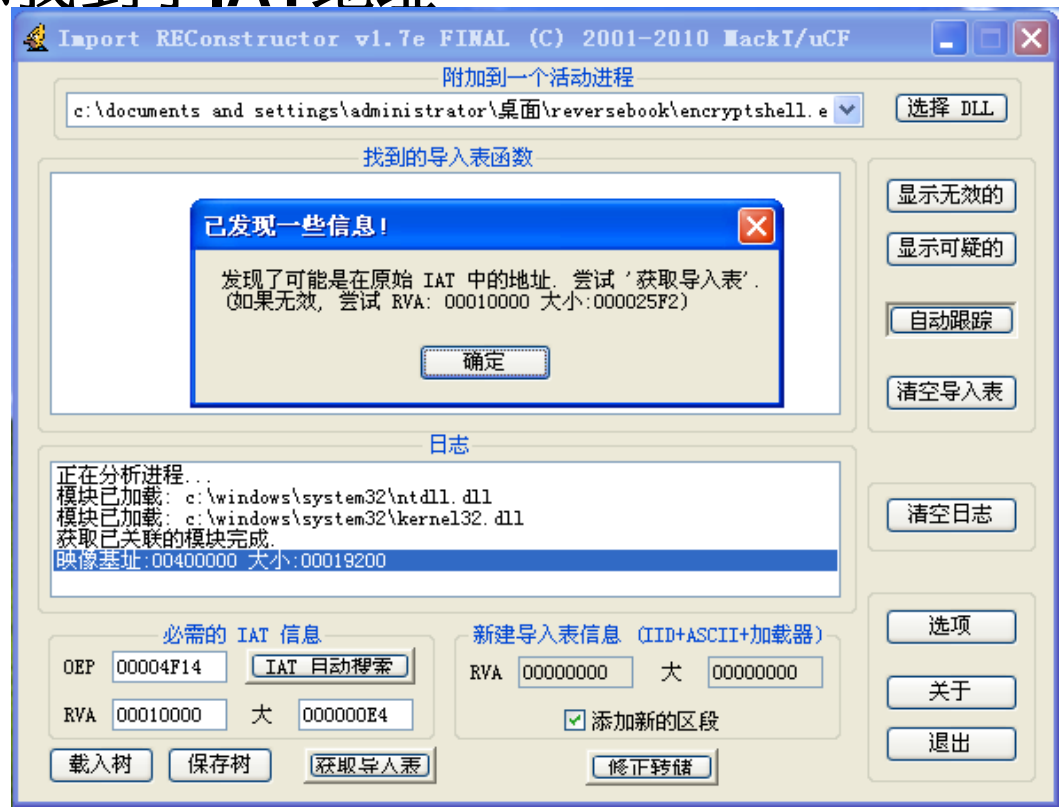
1 例题讲解

- 保存为dump.exe



1 例题讲解

- 接下来，需要用到ImportREC修复程序的IAT，将OEP填为刚刚找到的入口处相对地址（之前自动找的是错误的），即0x4F14，并点击IAT自动搜索，提示找到了IAT地址



1 例题讲解

- 点击“获取导入表”，然后修正转储即可

1 例题讲解

- 脱完壳的程序能够正常运行说明我们脱壳成功，将程序拖入IDA静态分析，可以在main函数看到程序的关键逻辑

```
119  sub_4048BA(aPleaseInput);
120  sub_4011D0(v63, 30, 10);
121  v3 = strlen(v63);
122  if ( v3 < 27 )
123    goto LABEL_14;
124  for ( i = 0; i < v3; ++i )
125    v62[i] = (v63[i] >> 2) + ((v63[i] & 3) << 6);
126  for ( j = 0; j < v3; ++j )
127    v62[j] ^= *(&v8 + j);
128  for ( k = 0; k < v3; ++k )
129  {
130    if ( v62[k] != *(&v35 + k) )
131      break;
132  }
133  if ( k != v3 )
134  {
135LABEL_14:
136    sub_4048BA(aSorryYouAreWro);
137    result = 0;
138  }
139  else
140  {
141    sub_4048BA(aCorrectTheFlag);
142    result = 0;
143  }
```

宝江

1 例题讲解

- 程序一开始读取用户输入，并判断字符长度，如果小于27就输出错误。程序先将用户输入的字符串每个字符的高6位和低2位调换，然后与某个字符相异或，最后将加密后的输入数组与预先设定的check数组判断是否相等

1 例题讲解

❍编写的解题脚本

```
#include<stdio.h>
#include<string.h>
intmain(){
        unsignedcharcheck[]={137,40,169,72,145,100,197,104,50,20,80,16,97,194,110,152,226,160,233,168,175,146,55,76,16,176,43};
        unsignedcharxor[]={25,125,189,93,79,52,18,188,126,79,76,11,56,21,63,3,58,60,183,180,178,70,45,21,11,171,116};
        unsignedcharflag[30]={0};
        intlen,i;
        len=27;
        for(i=0;i<len;i++){
                flag[i]= check[i]^xor[i];
                flag[i]=((flag[i]&0x3f)<<2)+(flag[i]>>6);
        }
        printf("The flag is: %s\n",(char*)flag);
        return0;
}
```

北邮网安学院　崔宝江

三. 逆向分析

- 1例题讲解
- 2简单壳的手动实现

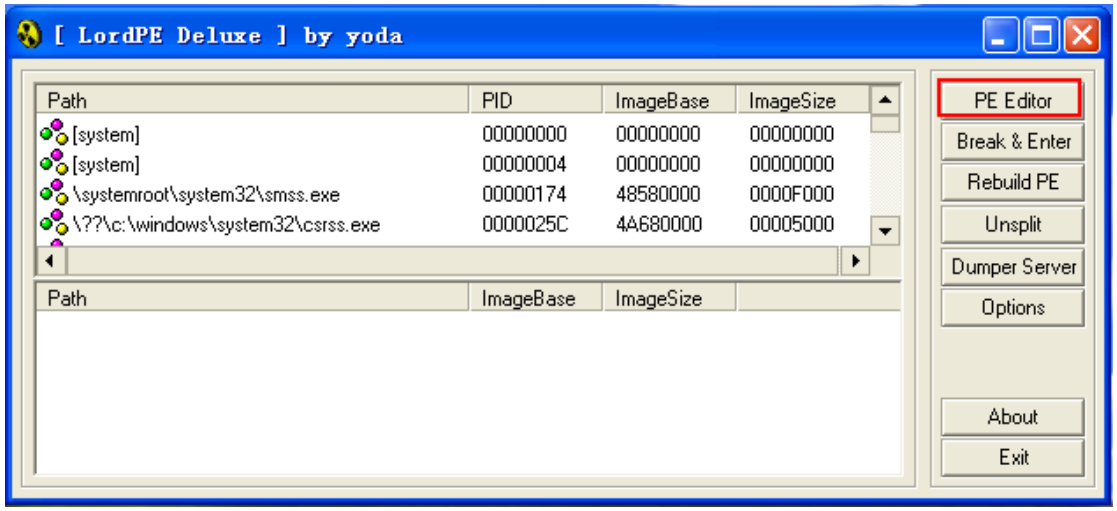
三. 逆向分析

- 这道题的加密壳是怎么实现的？
- 我们将一步一步介绍如何实现一个简单的加密壳
- 下面是源代码

```
#include <iostream>
#include <string.h>
usingnamespacestd;
intmain(){
        charinput[30];
        unsignedcharcheck[]={137,40,169,72,145,100,197,104,50,20,80,16,97,194,110,152,226,160,233,16
8,175,146,55,76,16,176,43};
        unsignedcharxor[]={25,125,189,93,79,52,18,188,126,79,76,11,56,21,63,3,58,60,183,180,178,70,4
5,21,11,171,116};
        unsignedcharresult[30];
        intlen,i;
        printf("Please input: ");
        cin.getline(input,30,'\n');
        len=strlen(input);
        if(len<27){
                printf("Sorry, you are wrong!\n");
                return0;
        }
        else{
                for(i=0;i<len;i++){
                        result[i]=(input[i]>>2)+((input[i]&0x3)<<6);
                }
                for(i=0;i<len;i++){
                        result[i]= result[i]^xor[i];
                }
                for(i=0;i<len;i++){
                        if(result[i]!=check[i])
                                break;
                }
                if(i==len)
                        printf("Correct! The flag is:%s\n",input);
                else
                        printf("Sorry, you are wrong!\n");
        }
        return0;
}
```

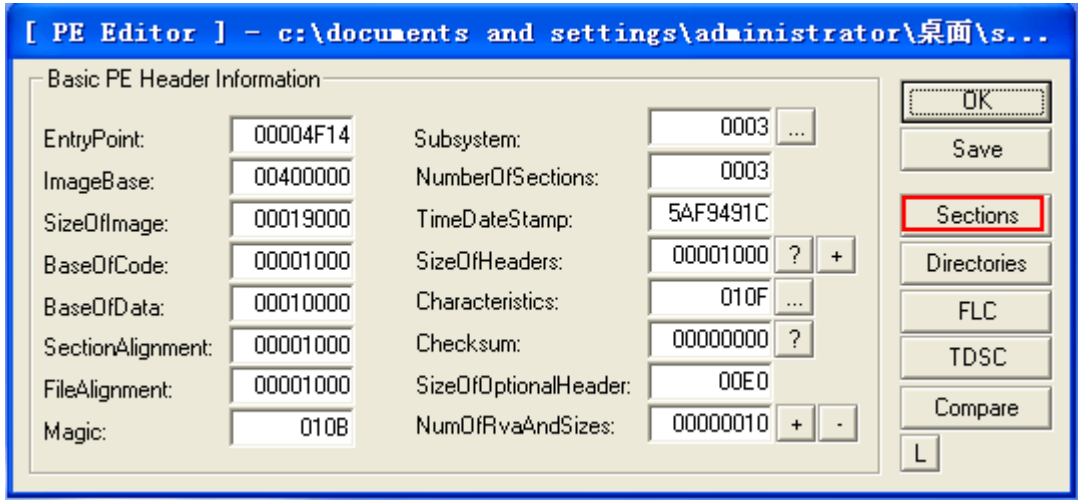
三. 逆向分析

- 用VC++6.0编译成release版本，点击LoadPE中PE Editor按钮，加载目标程序



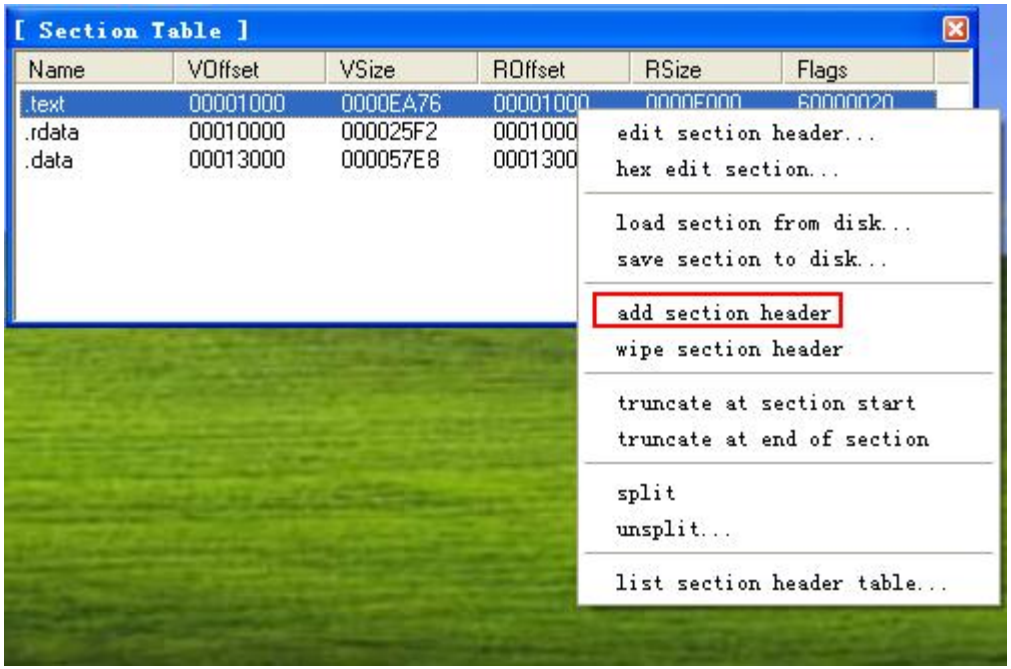
三. 逆向分析

- 可以看到程序加载的基址为**0x400000**，根据**EnrtyPoint**可以计算出程序的入口地址为**0x404F14**。点击**Sections**按钮



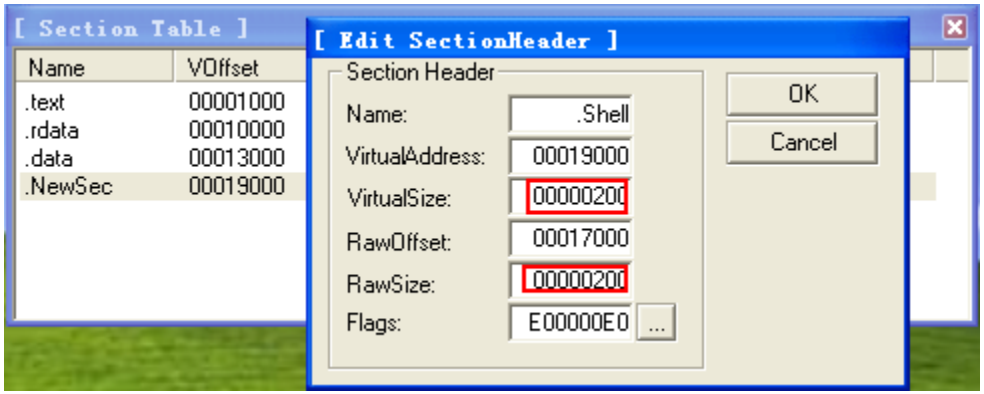
三. 逆向分析

- 在弹出的界面中，右击选择增加区段

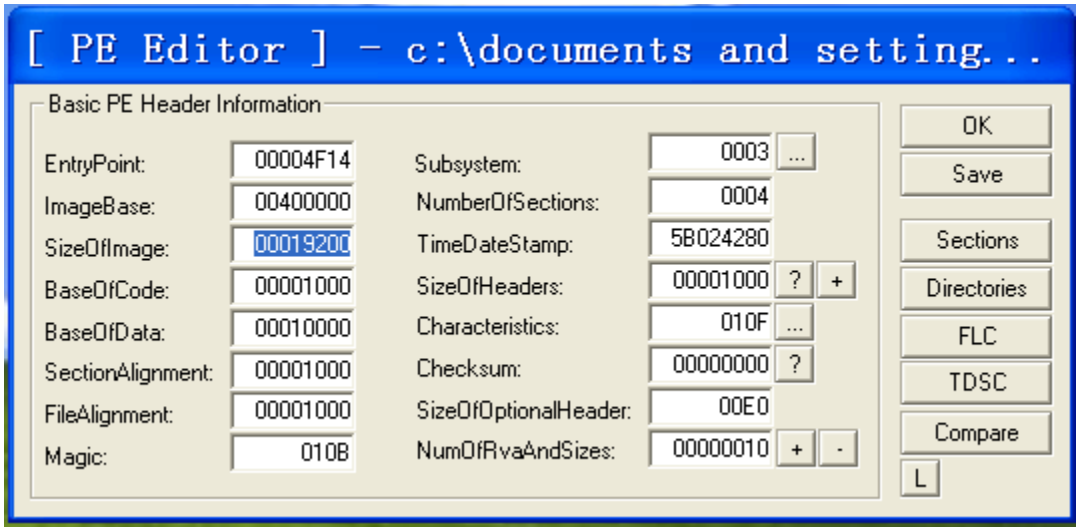


三. 逆向分析

- 编辑增加的区段
  - 修改名称为“.Shell”，并且将虚拟大小和物理大小都设置成0x200（根据想添加的代码量确定），然后保存

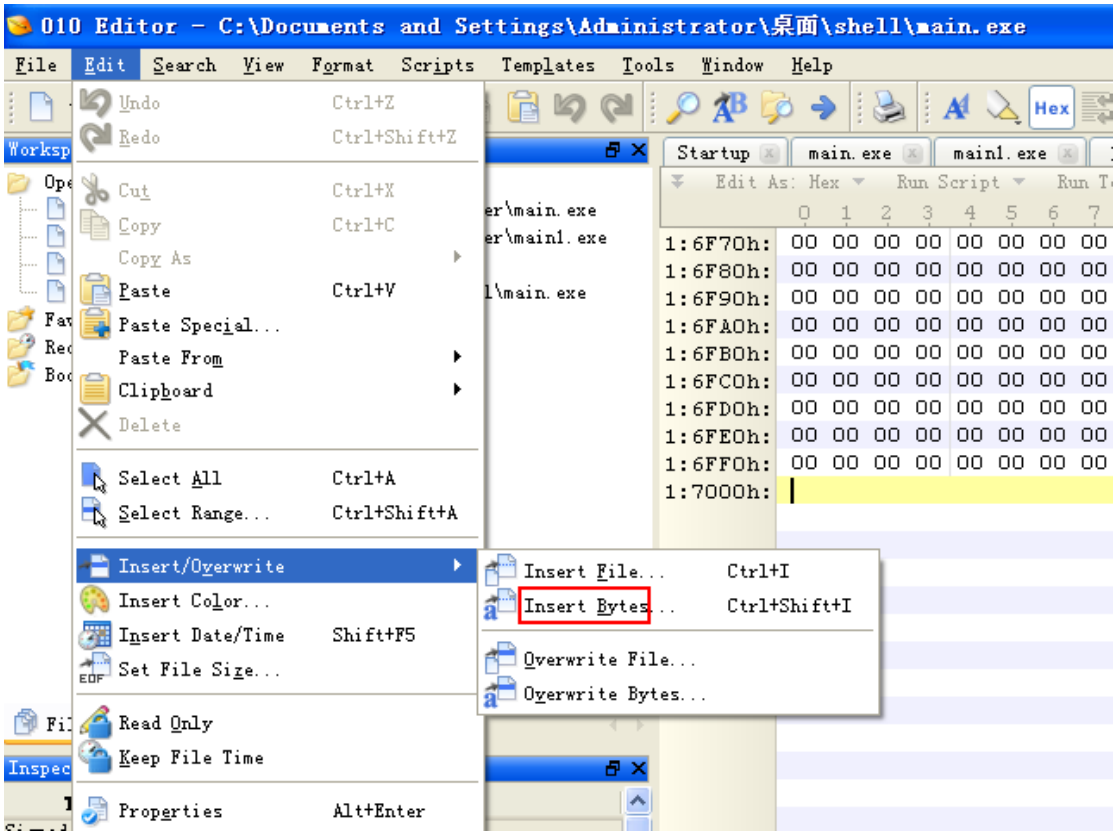


- 由于添加了一个区段，所以整个映像文件大小更改了，需要再将**sizeofimage**从**19000**，修改为**19200**，否则程序无法运行



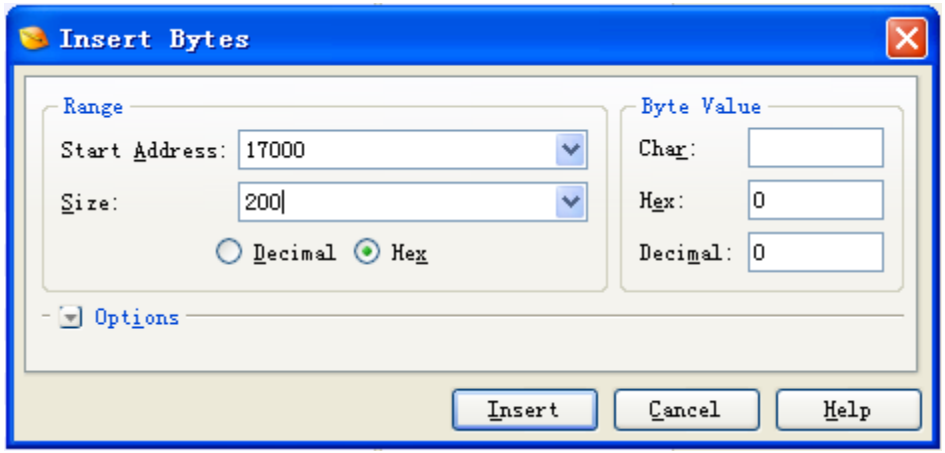
三. 逆向分析

- 用010 Editor编辑程序，给新加的区段添加内容，依次选择“Edit”->“Insert/Overwrite”->“Insert Byte”



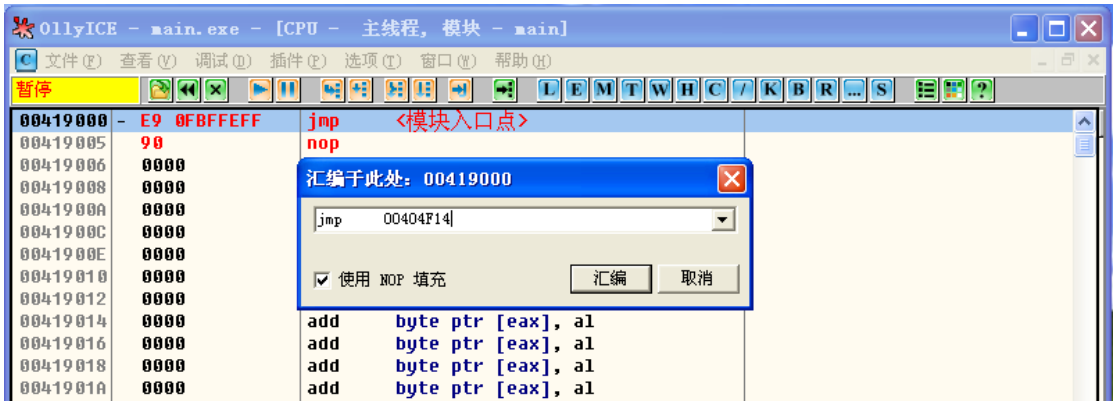
三. 逆向分析

- 在弹出的界面里填写如下内容，然后保存即可



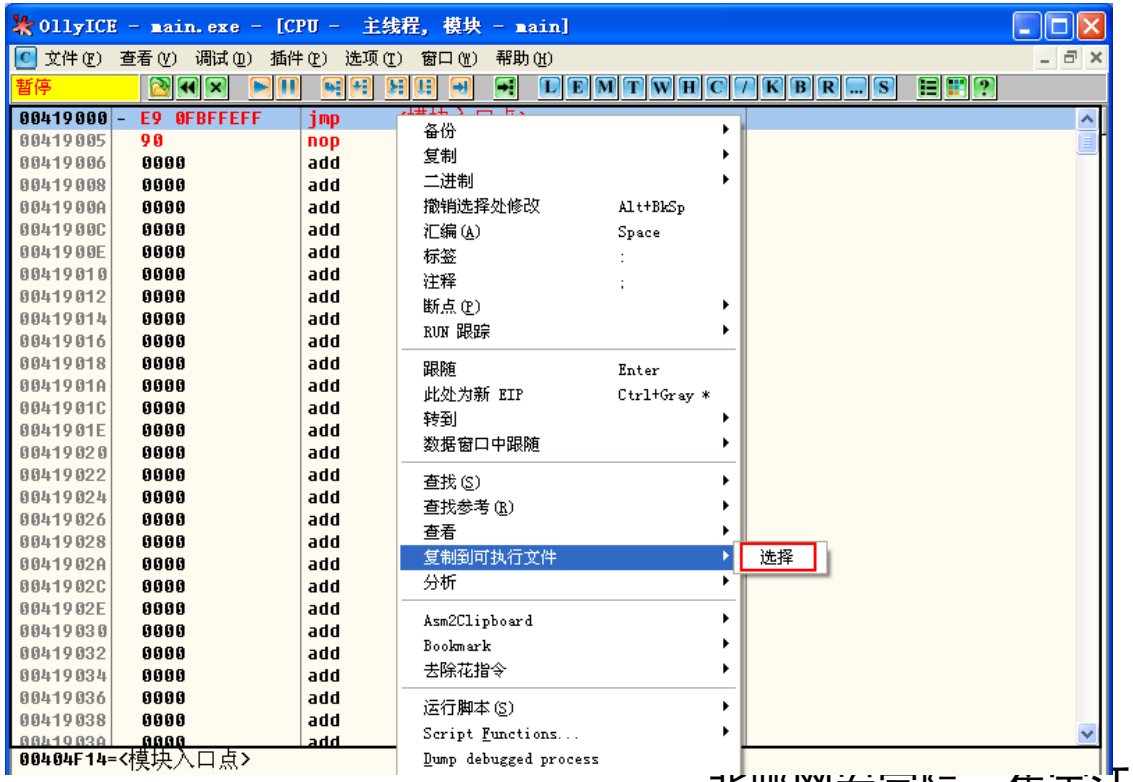
三. 逆向分析

- 接下来，我们将程序拖入OD，使用快捷键“ctrl+g”跳转到地址0x419000处，双击该地址处的汇编或按空格按钮，编辑汇编指令，使其跳转到入口地址0x404F14



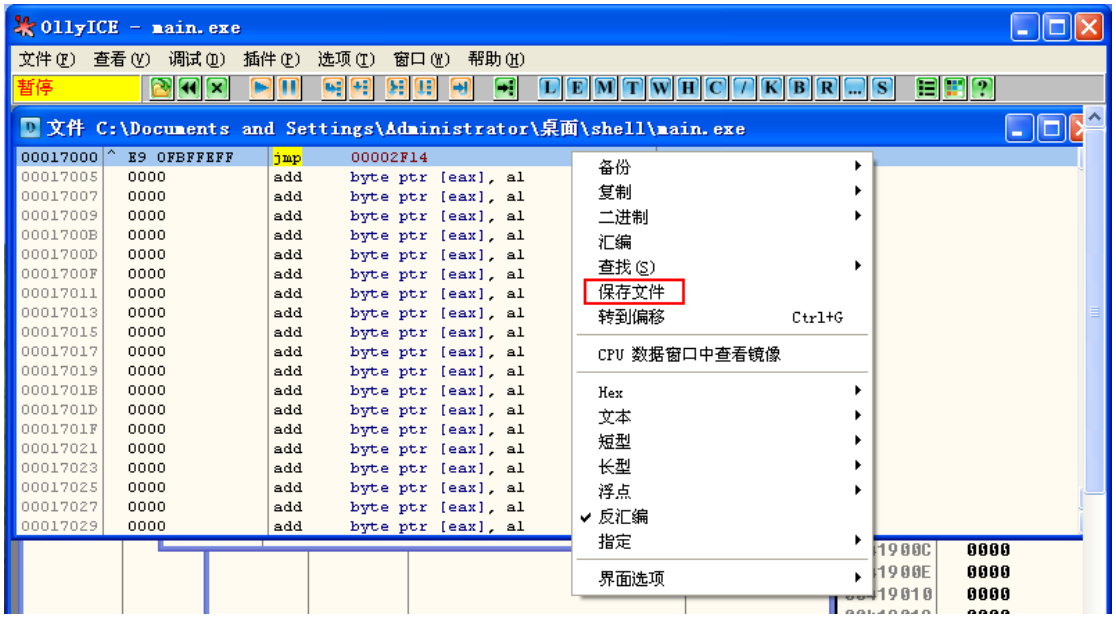
三. 逆向分析

- 选中修改的区域，右击选择“复制到可执行文件”->“选择”



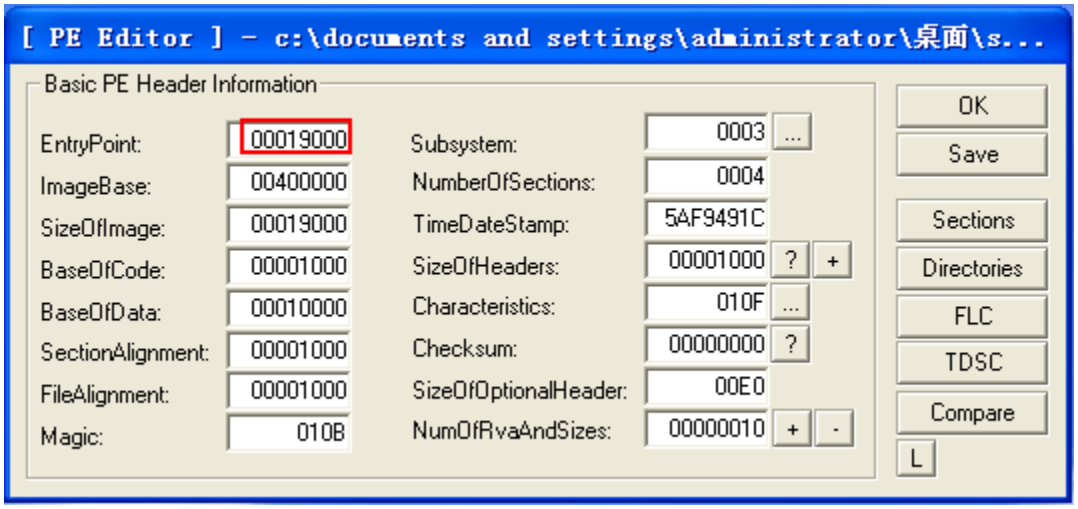
三. 逆向分析

- 然后继续右击，选择“保存文件”



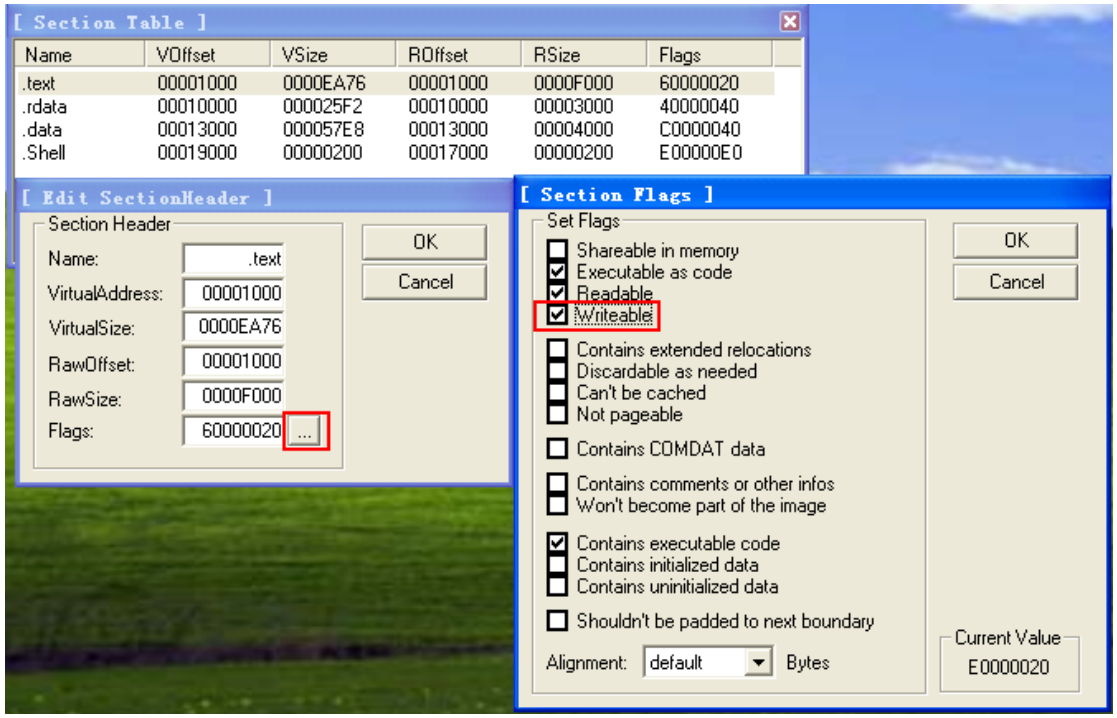
三. 逆向分析

- 用**LoadPE**修改程序的入口地址，**将其修改为.Shell段的虚拟地址，即0x19000**



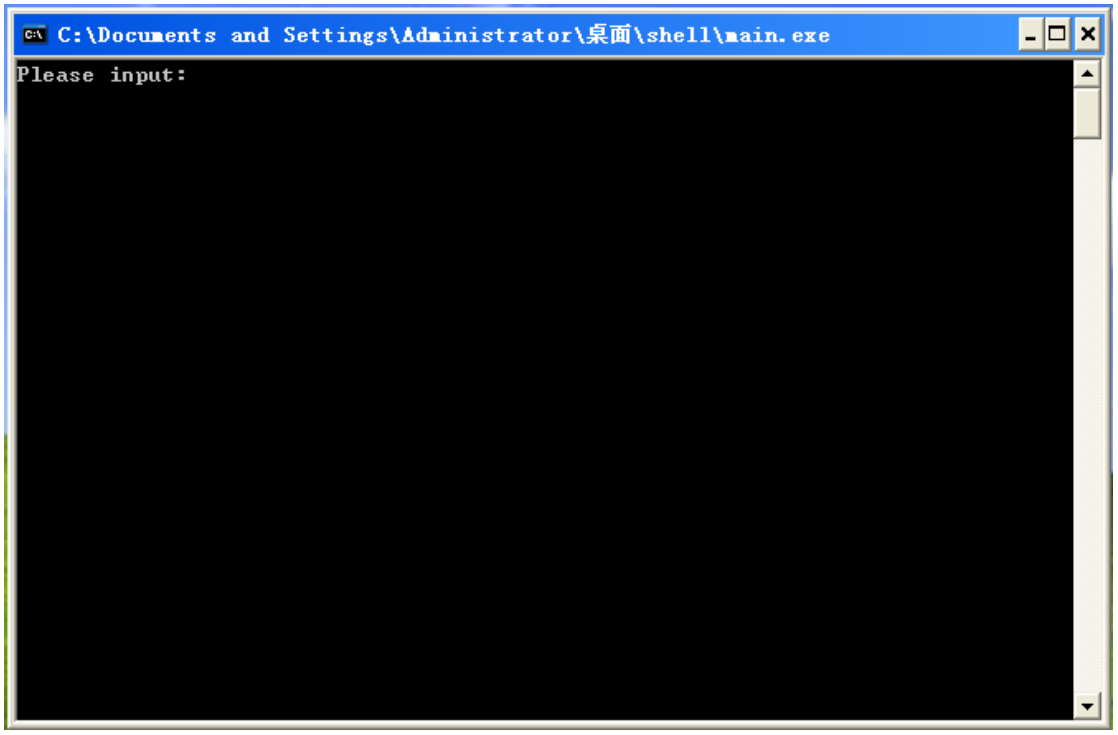
三. 逆向分析

- 选择section，并且，选中text段，右击选择编辑，修改text段的属性，将“Writeable”一项打勾



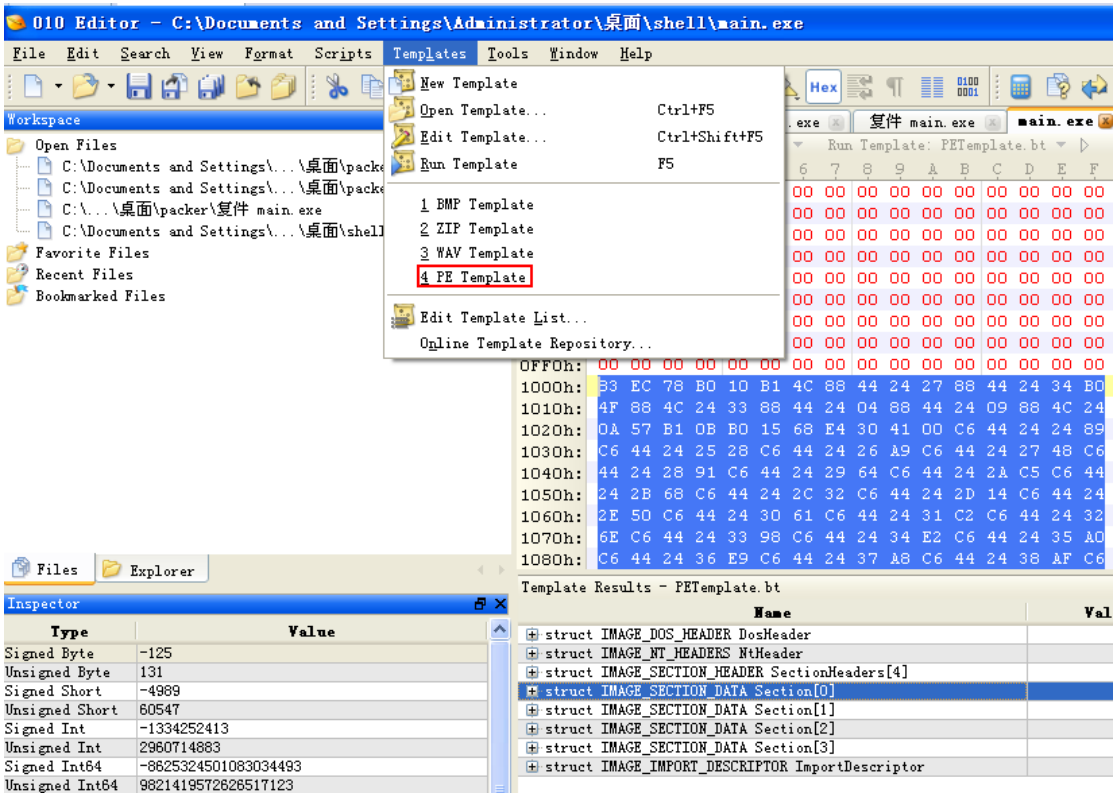
三. 逆向分析

- 这是可以试着运行一下程序，看看能不能正常运行，如果失败说明哪一下步骤错了，请重新来



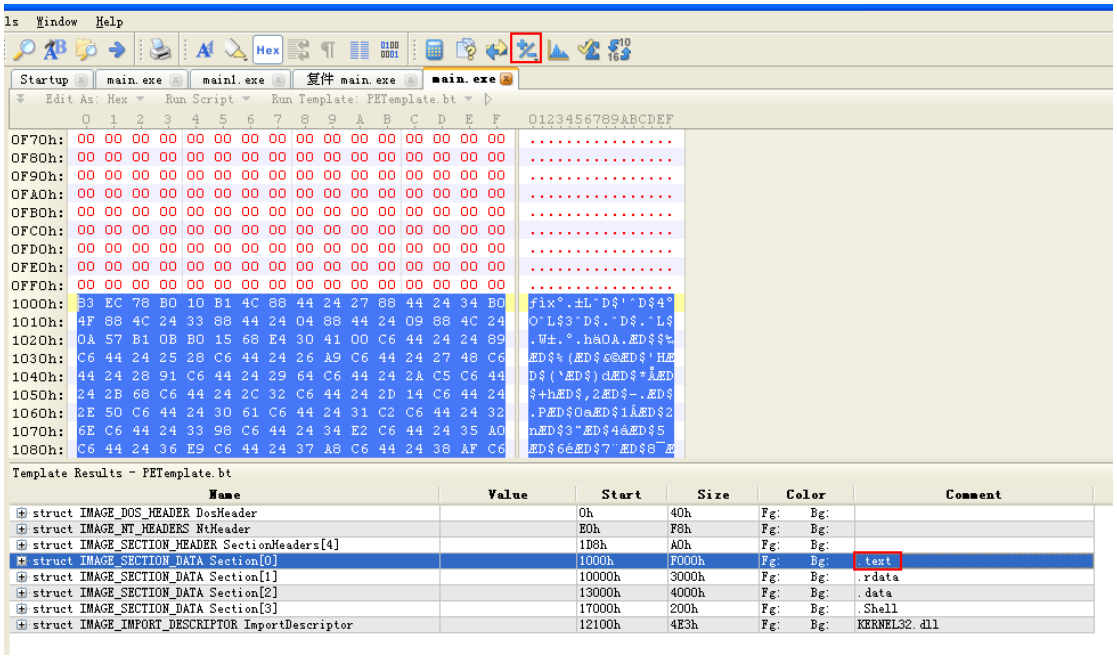
三. 逆向分析

- 继续用010 Editor编辑程序，并在“Templates”中选择“PE Template”，没有的话可以点击“Online Template Repository”去网上下载



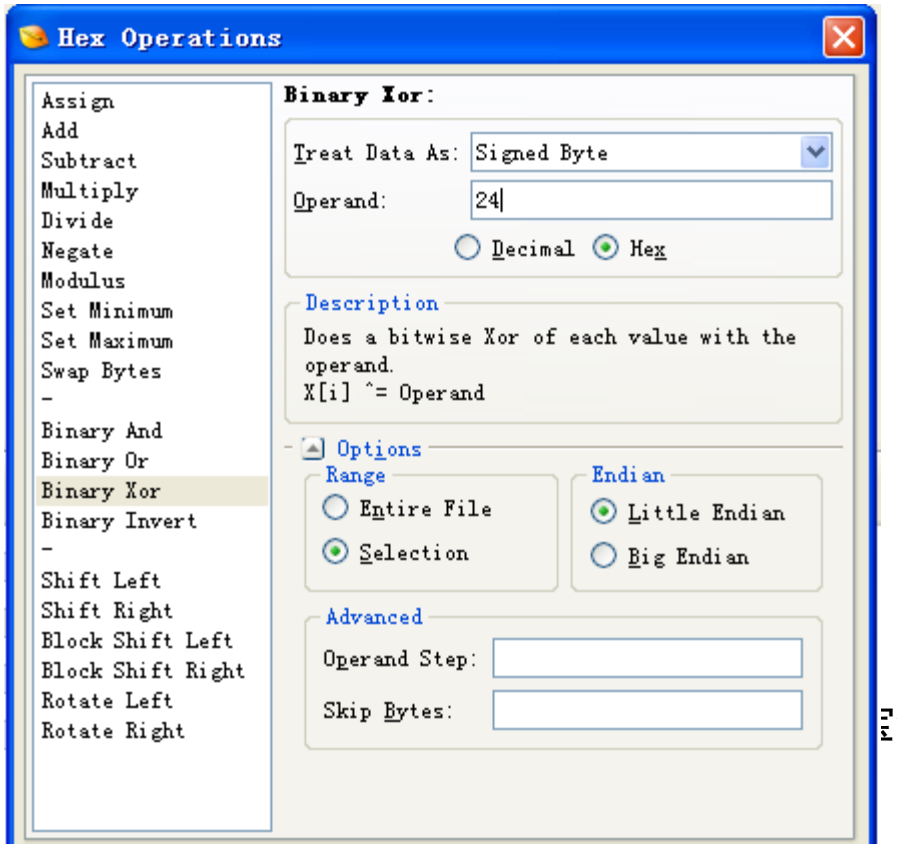
三. 逆向分析

- 选中**text**段，并点击“**Hex Operations**”按钮，即如图所示的加减号图标，选择“**BinaryXor**”：



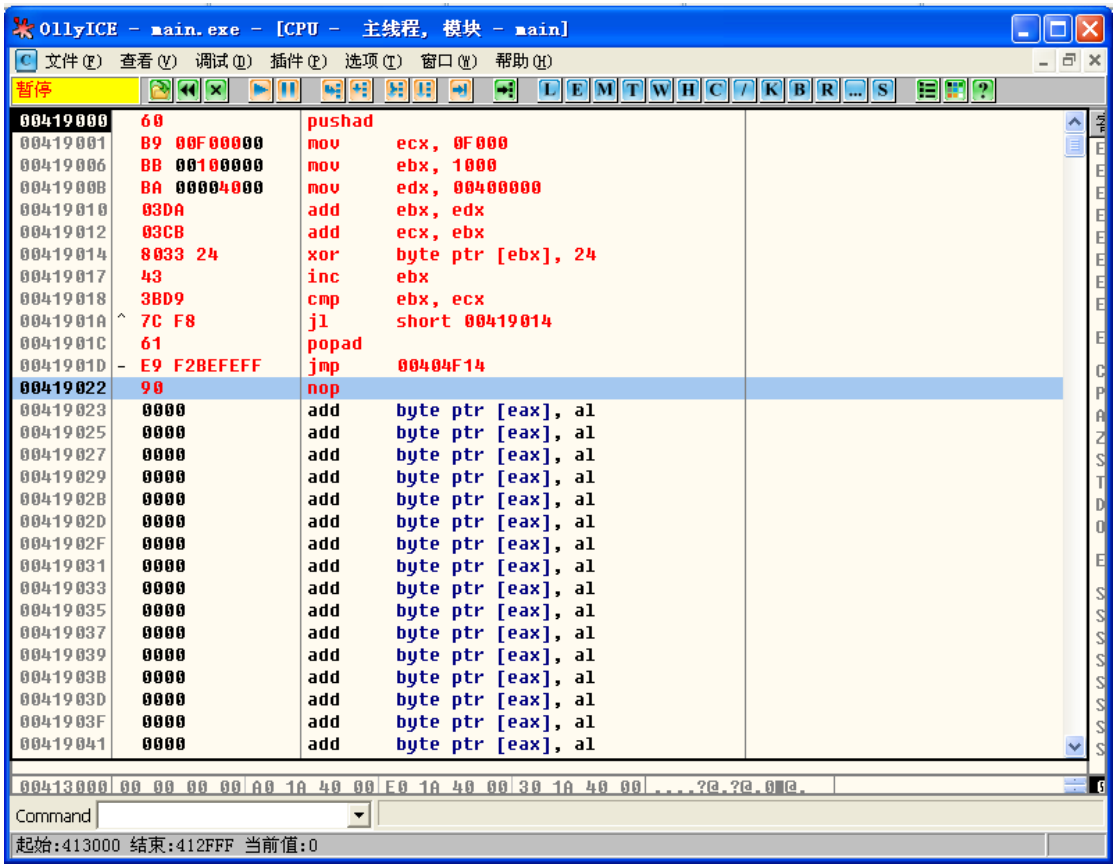
三. 逆向分析

- 在operand操作数填入0x24（即自己定的准备XOR的数值），即将text段的每个字节与0x24异或，点击OK后保存



三. 逆向分析

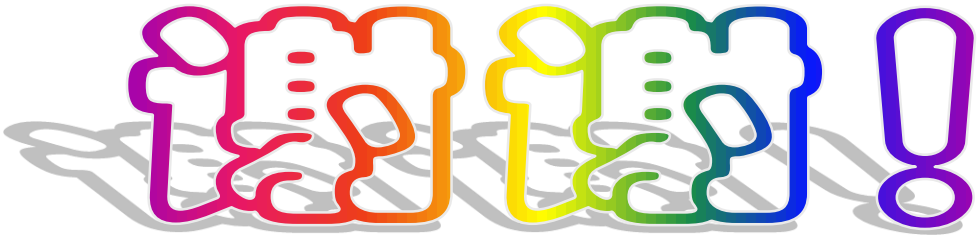
- 最后用OD载入，程序停在地址0x419000处，添加如下的汇编代码



三. 逆向分析

- 按照刚刚的方式保存即可，将修改后的程序拖入IDA，可以看到只有一个start函数，并且程序能够正常运行，说明已经成功的实现了一个简单的加密壳。

Q & A



北邮网安学院　崔宝江