

PE文件格式分析实验

实验报告

|  |  |
| --- | --- |
| 学生姓名 | maybeLocalhost |
| 学 号 |  |
| 专业班级 |  |
| 指导教师 |  |
| 学 院 | 计算机学院 |
| 完成时间 | 2020.11 |

目录

[一、实验简介 1](#_Toc62170324)

[1.1 实验目的 1](#_Toc62170325)

[1.2 实验内容 1](#_Toc62170326)

[1.3 实验环境 2](#_Toc62170327)

[二、实验原理 2](#_Toc62170328)

[2.1 PE文件基本介绍 2](#_Toc62170329)

[2.2 PE文件格式 2](#_Toc62170330)

[2.2.1 Dos部分 3](#_Toc62170331)

[2.2.2 PE头部分 3](#_Toc62170332)

[2.2.3 节表 4](#_Toc62170333)

[2.2.4 节数据 4](#_Toc62170334)

[三、实验过程 5](#_Toc62170335)

[3.1 使用UltraEdit观察Notepad.exe数据 5](#_Toc62170336)

[3.2 Ollydbg调试 9](#_Toc62170337)

[3.2.1 打开Ollydbg加载PE文件 9](#_Toc62170338)

[3.2.2动态调试程序 10](#_Toc62170339)

[3.3 熟悉各类PE文件格式查看和编辑工具 10](#_Toc62170340)

[3.3.1 PEView、Stud\_PE、PEditor查看文件 11](#_Toc62170341)

[3.4 调试并修改notepad.exe程序 15](#_Toc62170342)

[3.5 分析user32.dll文件 16](#_Toc62170343)

[3.6 用UltraEdit修改notepad.exe程序的引入表 17](#_Toc62170344)

[3.6.1 获取Kernel32.dll基地址 18](#_Toc62170345)

[3.6.2 加载User32.dll 21](#_Toc62170346)

[3.7 获取 MessageBoxA的函数地址 24](#_Toc62170347)

[3.8 汉化PEview.exe程序 25](#_Toc62170348)

[3.9 修改PEview.exe的图标 26](#_Toc62170349)

[四、实验总结与问题的解决 27](#_Toc62170350)

[4.1 实验总结 27](#_Toc62170351)

[4.2 遇到问题的解决方案 29](#_Toc62170352)

[五、参考文献 29](#_Toc62170353)

# 一、实验简介

## 1.1 实验目的

1. 熟悉各种PE编辑查看工具，详细了解PE文件格式；
2. 重点分析PE文件文件头、引入表、引出表，以及资源表；
3. 调研并总结PE文件格式与PE病毒之间的关系；
4. 自己制作一个尽可能小的PE文件。

## 1.2 实验内容

本实验涵盖以下内容：

1. 使用UltraEdit观察Notepad.exe数据，制作该PE文件基本结构图，格式如下：MZ头部+DOS stub+PE文件头+可选文件头（含数据目录）+节表+节（请注意地址对齐方式）；
2. Ollydbg对该程序进行初步调试，了解该程序功能结构，在内存中观察该程序的完整结构（请注意地址对齐方式）；
3. 熟悉各类PE文件格式查看和编辑工具(PEVIEW、Stub\_PE、PEditor)。结合notepad.exe熟悉PE文件头部、引入表、引出表的结构，按照课堂教学强调的关键结构或字段画图标注出来；熟悉函数导入、导出的基本原理；
4. 调试并修改notepad.exe程序（请注意不要修改功能代码，仅修改程序菜单项名称或者弹出窗口文本内容或窗口提示框标题）；
5. 找到系统SYSTEM32目录下的user32.dll文件，用UltraEdit打开并分析该文件导出表，找出 MessageBoxA的地址，并验证该地址是否正确；
6. 用UltraEdit修改notepad.exe程序的引入表，使得该程序仅可以从kernel32.dll中引入LoadLibrary和GetProcAddress函数，而不从user32.dll导入任何函数；
7. 在代码节中写入部分代码利用上述导入的两个函数获取 MessageBoxA的函数地址，使得notepad.exe程序的原有功能正常；
8. 自学探索类实验：熟悉资源表。利用PEview.exe熟悉资源表的结构，利用PEview.exe分析PEview.exe程序；用UltraEdit修改PEview.exe，使得该文件的图标变成PEview.ico；
9. 选做实验：熟悉eXeScope工具的实用，并利用该工具汉化PEview.exe程序；

## 1.3 实验环境

VMware 15.0、Windows 7

# 二、实验原理

## 2.1 PE文件基本介绍

PE文件的全称是Portable Executable，意为可移植的可执行的文件，是Windows操作系统下使用的一种可执行文件，由COFF（UNIX平台下的通用对象文件格式）格式文件发展而来。32位成为PE32，64位称为PE+或PE32+，常见的EXE、DLL、OCX、SYS、COM都是PE文件。

## 2.2 PE文件格式

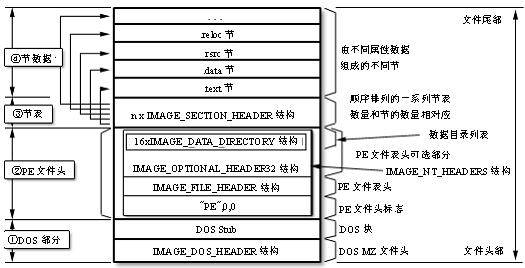


图 1 PE文件格式

一个完整的PE文件主要有4个部分组成：DOS头，PE头，节表以及节数据。

1. Dos部分主要用来对非FE格式文件的处理，DOS时代遗留的产物，是PE文件的一个遗传基因；
2. PE头部分用于宏观上记录文件的一些信息，运行平台，大小，创建日期，属性等；
3. 节表部分用于对各中类型的数据进行定义分段；
4. 节数据不言而喻就是文件的数据部分，实际上我们编写程序的过程中就是对该部分的数据进行编写。而其他的部分则是由编译器依照我们编写的部分进行相应的填写而得到的。

### 2.2.1 Dos部分

DOS部分由如下两部分构成：Dos头和Dos块。

* Dos头：长度40h；
* Dos块：长度不定，DOS插桩代码，是DOS下的16位程序代码，只是为了显示上面的提示数据。这段代码是编译器在程序编译过程中自动添加的。

Dos头对于非pe结构的文件将指引dos可执行程序部分，也就指引到dos块。而对于PE结构的文件将指引到PE结构部分。具体如何指引是通过dos头结构进行指定的。

### 2.2.2 PE头部分

PE头的三个构成部分：

* PE标志位，长度4H（4个字节）；
* 文件头，长度14H（20个字节）；
* 可选头，长度E0H（224个字节）；

PE header 是真正的Win32程序的格式头部，其中包括如下：

* PE标志位：共有4个字节的长度，用来判断文件是否为pe文件的标识。PE文件该字段的值固定为00004550，在字符串中看到的数据为“PE..”；
* 文件头：共占有14H个字节的长度，主要记录该文件的一些调试信息；
* 可选头：长度为E0H个字节。该部分虽然叫可选头，PE文件中不可或缺的一部分。文件在装入内存时，主要的参照的就是该结构。文件在从源代码进行编译的时候也主要是对该部分进行填写。并在生成的时候依照该结构的一些字段的设置。

### 2.2.3 节表

* 节表：PE代码和数据的结构数据，指示装载系统代码段在哪里，数据段在哪里等。

PE文件中所有节的属性都被定义在节表（段表）中，节表是一个由IMAGE\_SECTION\_HEADER结构组成的数组，每个结构用来描述一个节。我们来分析一下这个结构。

结构的排列顺序和它们描述的节在文件中的排列顺序是一致的。全部有效结构的最后以一个空的IMAGE\_SECTION\_HEADER结构作为结束，所以节表中总的IMAGE\_SECTION\_HEADER结构数量等于节的数量加一。节表总是被存放在紧接在PE文件头的地方，也就是从PE文件头开始的偏移为00f8h的地方。（注意：不是文件本身的头部，由于程序的dos块的大小并不是固定的，所以导致PE头相对于mz头的偏移变化的。 所以我们在计算偏移的时候，对于PE头后面的数据都习惯用相对于PE头的偏移，PE头大小是固定的的。）

在文件头中已经定义了文件的节的数目：0ah个，那么该文件的节表共有11个IMAGE\_SECTION\_HEADER结构。

### 2.2.4 节数据

常见的节数据：

* .text：代码段，是在编译或汇编结束时产生的一种块，它的内容全部是指令代码。也有的编译器将该段命名为.code；
* .data：初始化的数据块，是初始化的数据块，包含那些编译时被初始化的变量、字符串；
* .idata：输入表，包含其他外来dll的函数和数据信息，也就是输入表，也有人称之为导入表；
* .rsrc：资源数据块，包含模块的全部资源数据，如图标、菜单、位图等；
* .reloc：重定位表，用于保存基址的重定位表。即当装在程序不能按照连接器所指定的地址装载文件是，需要对指令或已经初始化的变量进行调整，该块中也包含了调整过程中所需要的一些数据，如果装载能够正常装在则忽略此段中的数据；
* .edata：导出表，是pe文件的输出表，以供其他模块使用，并不是每个pe文件都有此数据段，因为有的文件并不需要输出一些函数，该数据段常见于动态连接库文件中；
* .radata：存放调试目录、说明字符串，该数据块并不常见主要是用于存放一些调试信息。

# 三、实验过程

## 3.1 使用UltraEdit观察Notepad.exe数据

该部分实验使用UltraEdit观察Notepad.exe数据，制作该PE文件基本结构图，格式如下：MZ头部+DOS stub+PE文件头+可选文件头（含数据目录）+节表+节（请注意地址对齐方式）。

* **MZ头文件（000h-03fh）**

该部分固定大小为40H个字节。偏移3cH处字段Offset to New EXE Header，指示“NT映象头的偏移地址”，其中000000F8是NT映象头的文件偏移地址，定位PE文件头开始位置，用于PE文件合法性检验。DOS下运行该程序时，会提示用户“This Program cannot be run in DOS mode”。

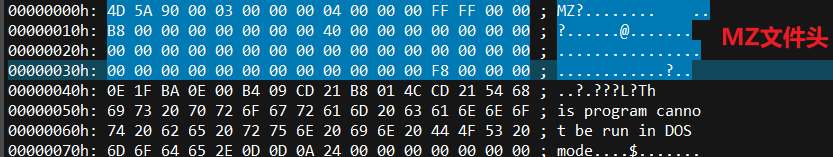


图 2 MZ文件头

* **DOS Stub（040h-0f7h）**

DOS Stub部分大小不固定，位于MZ文件头和NT映象头之间，可由MZ文件头中的Offset to New EXE Header字段确定。

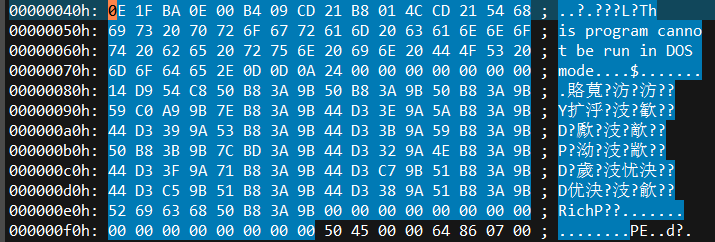


图 3 DOS Stub

* **PE文件头（0f8h-1ffh）**

该部分包括PE标识、映像文件头、可选文件头。



图 4 PE文件头

* Signature：字串“PE\0\0”，4个字节（0f8H~0fcH）
* 映象文件头File Header：14H个字节（0fdH~10cH）

偏移2H处，字段Number of Section 给出节的个数（2个字节）：0070 偏移10H处，字段Size of Optional Header 给出可选映象头的大小（2个字节）：00F0

* 可选映象头Optional Header：0100H~1a7H

PE文件可选文件头224字节，其对应的字段信息如下所示：

1. **typedef** **struct** \_IMAGE\_OPTIONAL\_HEADER {
3. **WORD**    Magic;　　　　　　　　　　　　　　　　　　/\*机器型号,判断是PE是32位还是64位\*/
4. **BYTE**    MajorLinkerVersion;　　　　　　　　　　/\*连接器版本号高版本\*/
5. **BYTE**    MinorLinkerVersion;　　　　　　　　　　/\*连接器版本号低版本,组合起来就是 5.12 其中5是高版本,C是低版本\*/
6. **DWORD**   SizeOfCode;　　　　　　　　　　　　　　　/\*代码节的总大小(512为一个磁盘扇区)\*/
7. **DWORD**   SizeOfInitializedData;　　　　　　　　/\*初始化数据的节的总大小,也就是.data\*/
8. **DWORD**   SizeOfUninitializedData;　　　　　　 /\*未初始化数据的节的大小,也就是 .data ? \*/
9. **DWORD**   AddressOfEntryPoint;　　　　　　　　  /\*程序执行入口(OEP) RVA(相对偏移)\*/
10. **DWORD**   BaseOfCode;　　　　　　　　　　　　　　 /\*代码的节的起始RVA(相对偏移)也就是代码区的偏移,偏移+模块首地址定位代码区\*/
11. **DWORD**   BaseOfData;　　　　　　　　　　　　　　 /\*数据结的起始偏移(RVA),同上\*/
12. **DWORD**   ImageBase;　　　　　　　　　　　　　　　/\*程序的建议模块基址(意思就是说作参考用的,模块地址在哪里)\*/
14. **DWORD**   SectionAlignment;　　　　　　　　　　　/\*内存中的节对齐\*/
15. **DWORD**   FileAlignment;　　　　　　　　　　　　　/\*文件中的节对齐\*/
16. **WORD**    MajorOperatingSystemVersion;　　　　/\*操作系统版本号高位\*/
17. **WORD**    MinorOperatingSystemVersion;　　　　/\*操作系统版本号低位\*/
18. **WORD**    MajorImageVersion;　　　　　　　　　　/\*PE版本号高位\*/
19. **WORD**    MinorImageVersion;　　　　　　　　　　/\*PE版本号低位\*/
20. **WORD**    MajorSubsystemVersion;　　　　　　　　/\*子系统版本号高位\*/
21. **WORD**    MinorSubsystemVersion;　　　　　　　　/\*子系统版本号低位\*/
22. **DWORD**   Win32VersionValue;　　　　　　　　　　/\*32位系统版本号值,注意只能修改为4 5 6表示操作系统支持nt4.0 以上,5的话依次类推\*/
23. **DWORD**   SizeOfImage;　　　　　　　　　　　　   /\*整个程序在内存中占用的空间(PE映尺寸)\*/
24. **DWORD**   SizeOfHeaders;　　　　　　　　　　　　/\*所有头(头的结构体大小)+节表的大小\*/
25. **DWORD**   CheckSum;　　　　　　　　　　　　　　 /\*校验和,对于驱动程序,可能会使用\*/
26. **WORD**    Subsystem;　　　　　　　　　　　　　　/\*文件的子系统 :重要\*/
27. **WORD**    DllCharacteristics;　　　　　　　　 /\*DLL文件属性,也可以成为特性,可能DLL文件可以当做驱动程序使用\*/
28. **DWORD**   SizeOfStackReserve;　　　　　　　　 /\*预留的栈的大小\*/
29. **DWORD**   SizeOfStackCommit;　　　　　　　 　 /\*立即申请的栈的大小(分页为单位)\*/
30. **DWORD**   SizeOfHeapReserve;　　　　　　　  　/\*预留的堆空间大小\*/
31. **DWORD**   SizeOfHeapCommit;　　　　　　　 　  /\*立即申请的堆的空间的大小\*/
32. **DWORD**   LoaderFlags;　　　　　　　　　　　　 /\*与调试有关\*/
33. **DWORD**   NumberOfRvaAndSizes;　　　　　　　  /\*下面的成员,数据目录结构的项目数量\*/
34. IMAGE\_DATA\_DIRECTORY DataDirectory[16];  /\*数据目录,默认16个,16是宏,这里方便直接写成16\*/
35. } IMAGE\_OPTIONAL\_HEADER32, \*PIMAGE\_OPTIONAL\_HEADER32;

* **节表（200h-277h）**

总结来说节表中重要成员有三个：

1. 内存中起始位置
2. 节数据对齐后大小
3. 文件中起始位置.

* 根据第二个成员和第三个成员可以得出节数据从哪里结束：

节起始位置 + 节数据对齐后大小 = 节结束位置

* + 每个表项的起始位置起（8个字节），字段Name给出对应节的名称。
  + 每个表项的偏移14H处（4个字节），字段Offset to Raw Data给出对应节的起始文件偏移。

该结构包括3个节，对应上图的3个struct IMAGE\_SECTION\_HEADER，即“.text”、“.rdata”、“.data”节：

* .text（200h-227h）



图 5 代码节

* .rdata（228h-24fh）



图 6 引出函数节

* .data（250h-277h）



图 7 已初始化的数据节

## 3.2 Ollydbg调试

该部分将使用Ollydbg对该程序进行初步调试，了解该程序功能结构，在内存中观察该程序的完整结构。注意，内存对齐单位和文件对齐单位的不同，内容和文件中IAT表内容的不同。

### 3.2.1 打开Ollydbg加载PE文件

Ollydbg是一款PE文件动态调试器，此时程序断点自动停止在程序入口点00971860H位置。

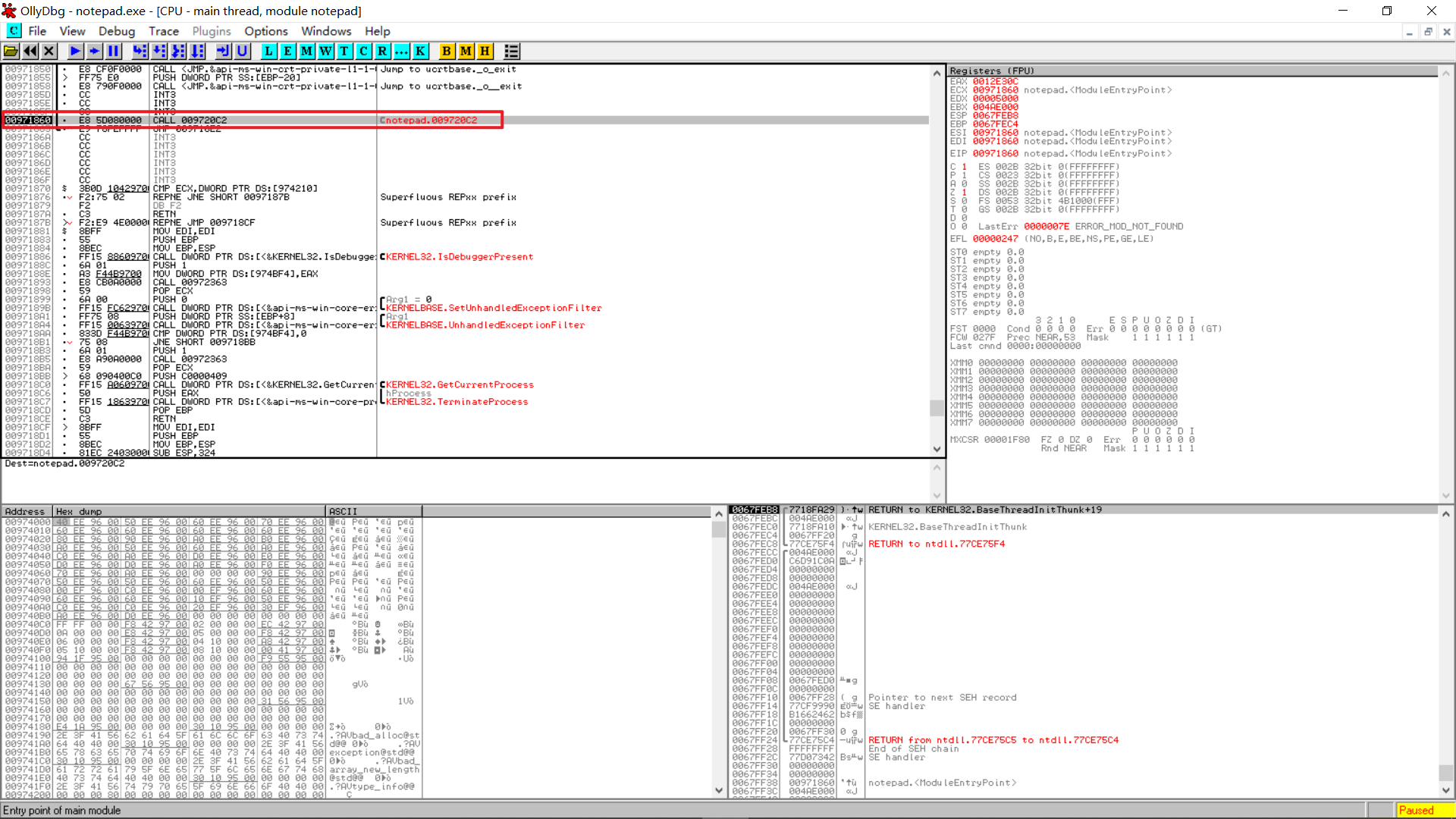


图 8 使用Ollydbg打开notepad.exe



图 9 程序入口点

### 3.2.2动态调试程序

F7单步步入，F8单步步过，这里我们使用F8单步步过动态调试程序。



图 10 F8单步步过动态调试程序

## 3.3 熟悉各类PE文件格式查看和编辑工具

熟悉各类PE文件格式查看和编辑工具(PEVIEW、Stub\_PE、PEditor)。结合notepad.exe熟悉PE文件头部、引入表、引出表的结构，按照课堂教学强调的关键结构或字段画图标注出来；熟悉函数导入、导出的基本原理。

在文件中的引入表结构：IMAGE\_IMPORT\_DESCRIPTOR：

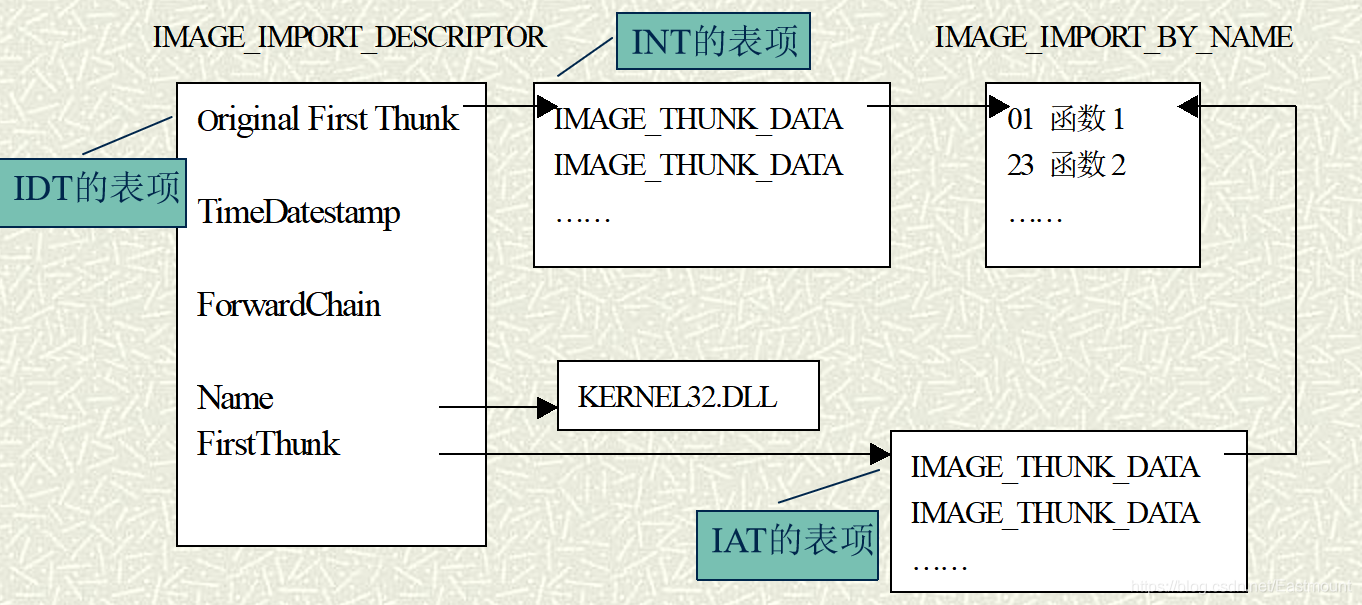


图 11 文件中的引入表结构

在内存中的引入表结构：IMAGE\_IMPORT\_DESCRIPTOR：

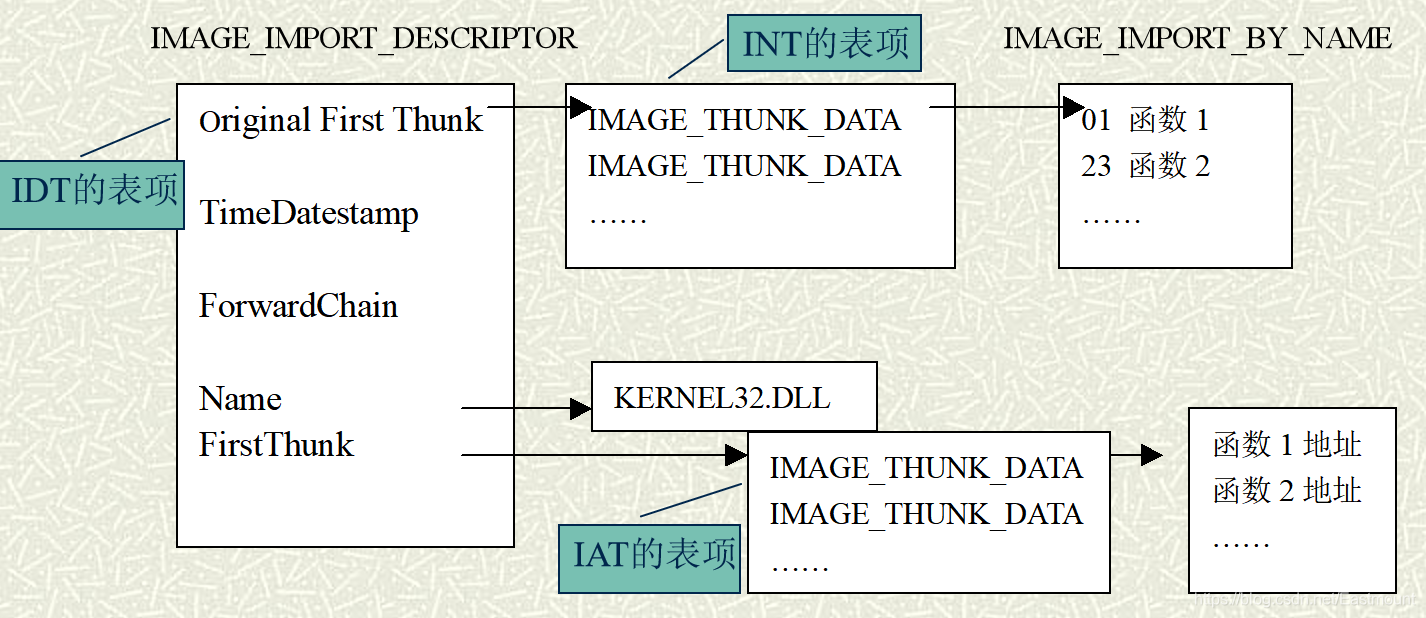


图 12 内存中的引入表结构

## 3.3.1 PEView、Stud\_PE、PEditor查看文件

1. **使用PEView查看文件属性**

PEview.exe工具是一款可以进行PE文件解析的强大PE文件解析工具，通过PEview软件可以分析PE文件的详细格式。如下图所示，左边是相应结构，右边是对应的数据。

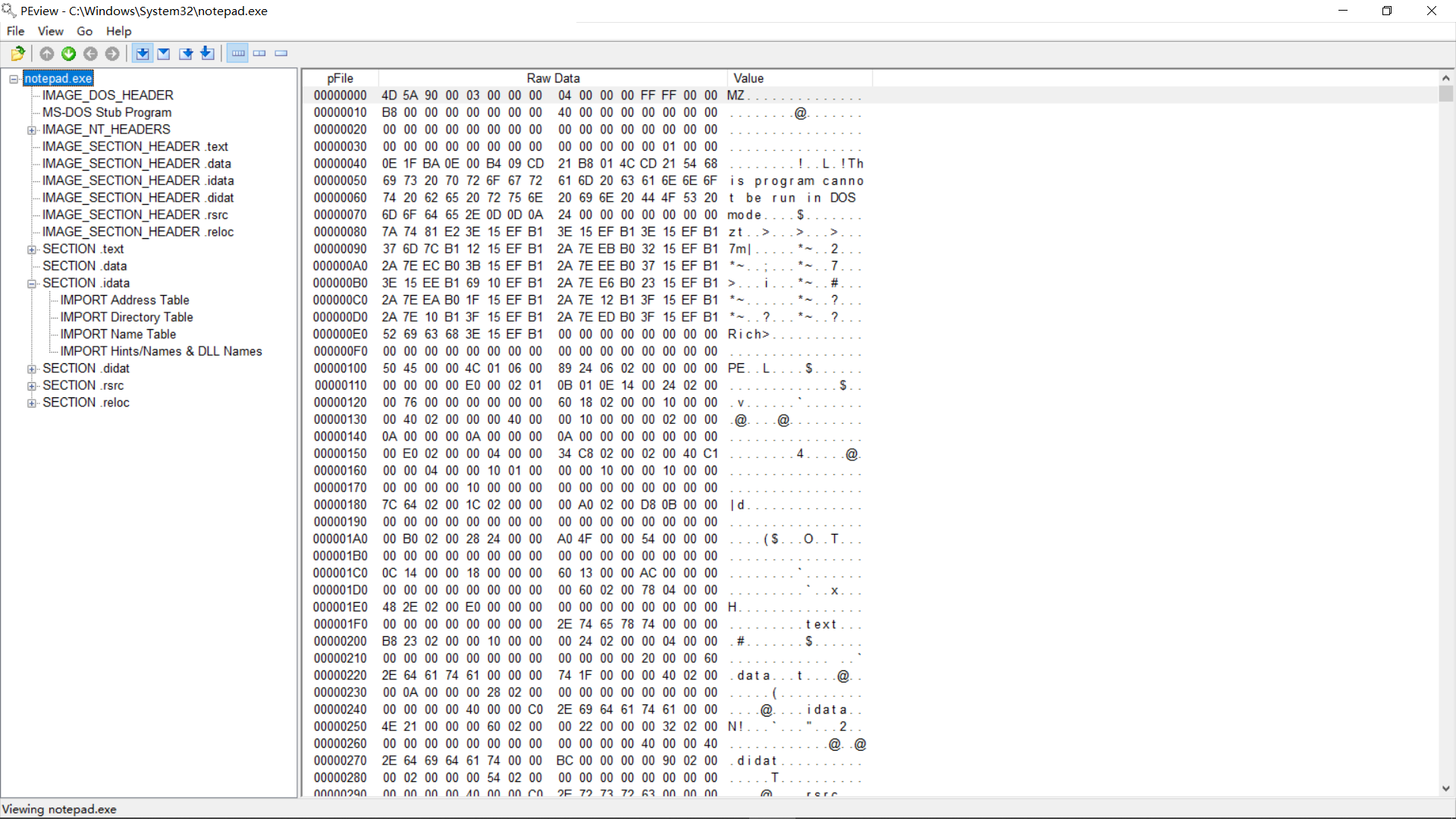


图 13 PEView查看notepad.exe

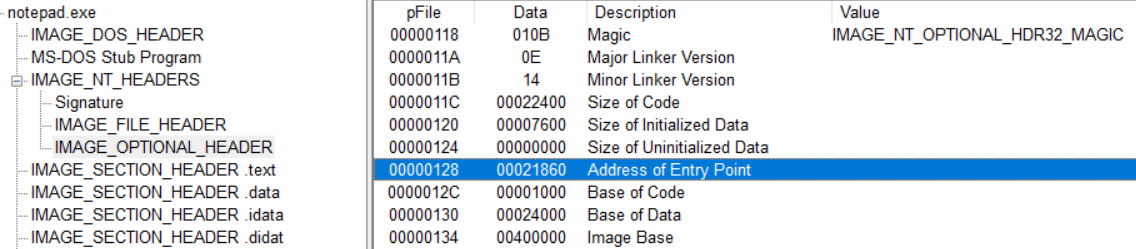


图 14 AddressOfEntryPoint

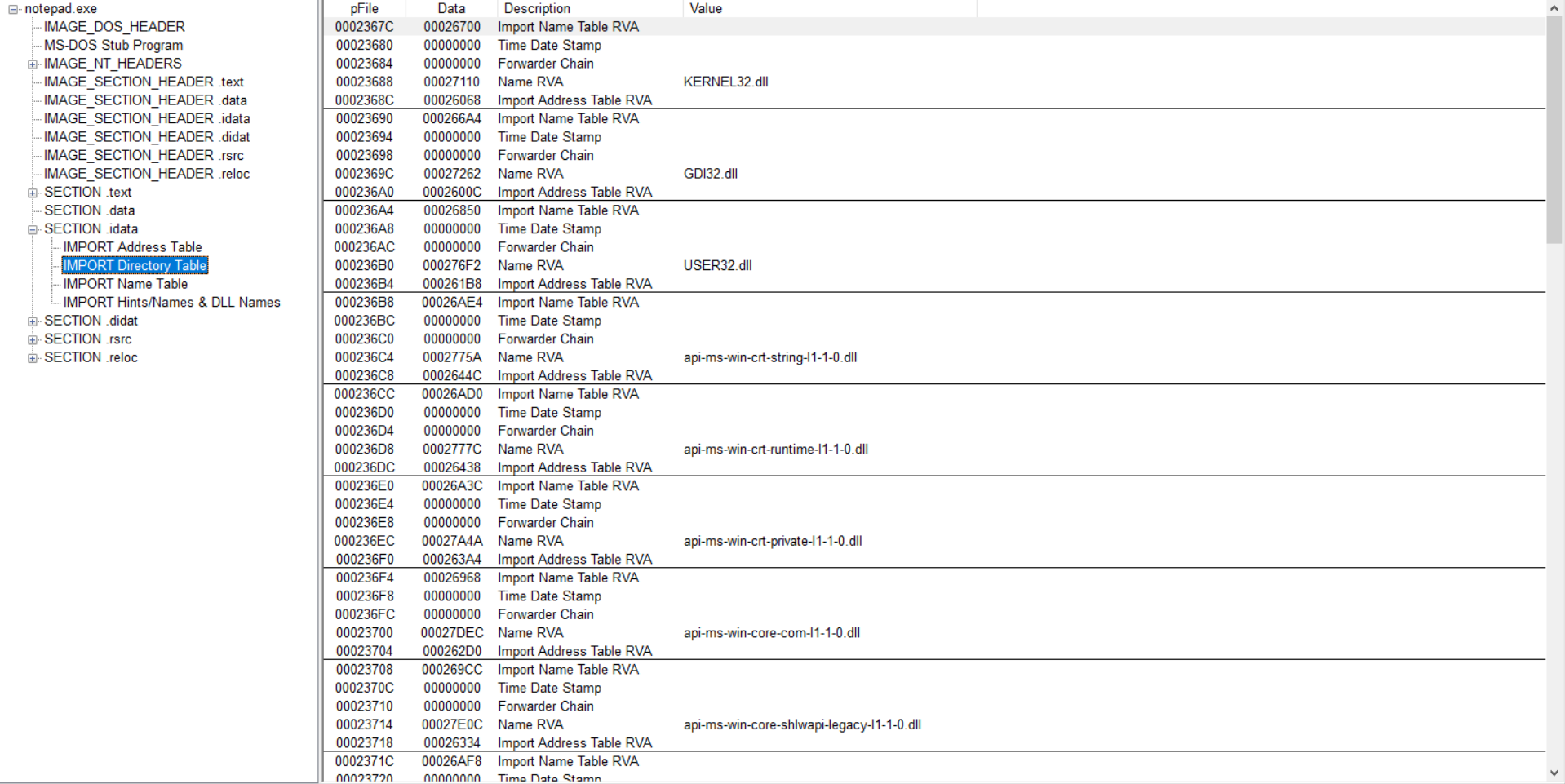


图 15 导入表信息

1. **使用stud\_PE查看文件属性**

该软件用于显示头部、DOS、区段、函数等信息，包括导入表、导出表等。

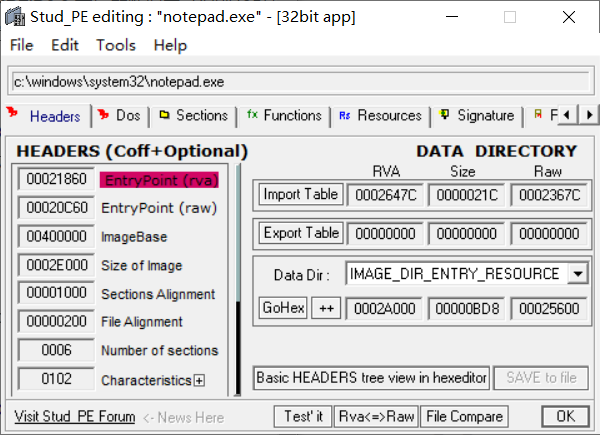


图 16 stud\_PE查看notepad.exe的Headers

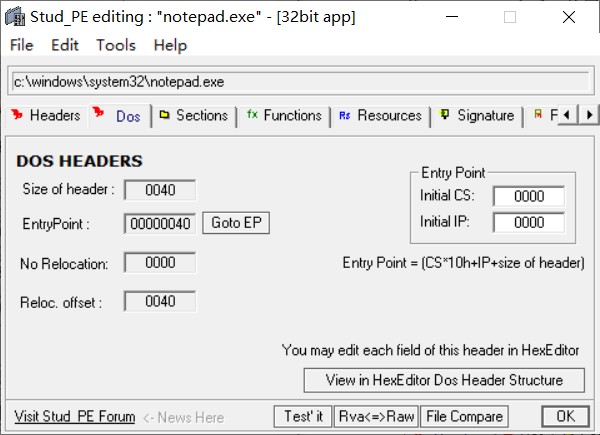


图 17 stud\_PE查看notepad.exe的Dos

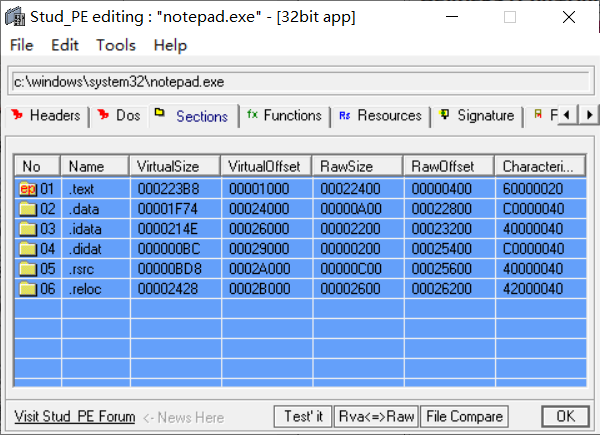


图 18 stud\_PE查看notepad.exe的Sections

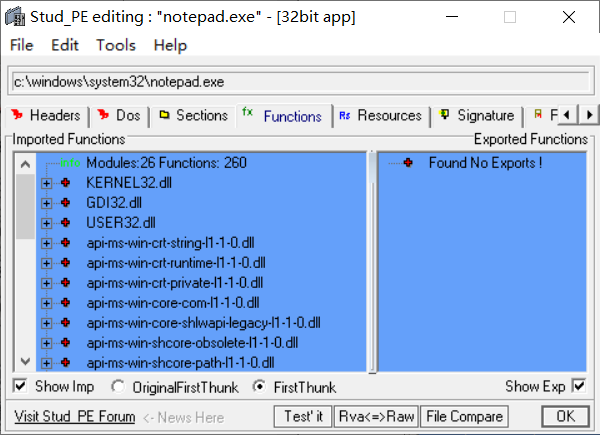


图 19 stud\_PE查看notepad.exe的Functions

1. **使用PEditor查看文件属性**

由于PEditor的版本问题且notepad.exe存在写保护，PEditor无法成功打开notepad.exe。



图 20 PEditor打开notepad.exe失败

## 3.4 调试并修改notepad.exe程序

调试并修改notepad.exe程序，请注意不要修改功能代码，仅修改程序菜单项名称或者弹出窗口文本内容或窗口提示框标题。由于notepad.exe存在写保护，无法进行修改，因此这里使用PEview.exe进行代替，原理仍相同。



图 21 修改前的PEview界面

使用UltraEdit打开PEview.exe文件，在其中使用“查找ASCII(S)”查找运行界面上的“pFile”并将其替换为“bFile”。



图 22 将“pFile”替换为“bFile”

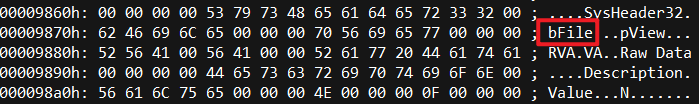


图 23 替换后的“bFile”

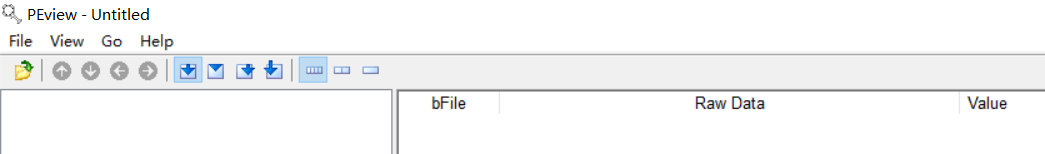


图 24 替换成功

## 3.5 分析user32.dll文件

找到系统SYSTEM32目录下的user32.dll文件，用UltraEdit打开并分析该文件导出表，找出 MessageBoxA的地址，并验证该地址是否正确。

首先我们使用PEView打开user32.dll，查找MessageBoxA的RVA。找到系统System32目录下的user32.dll文件并打开，在AddressOfNames数组（Name Pointer Table）中找到MessageBoxA字符串，其数组序号为0865。

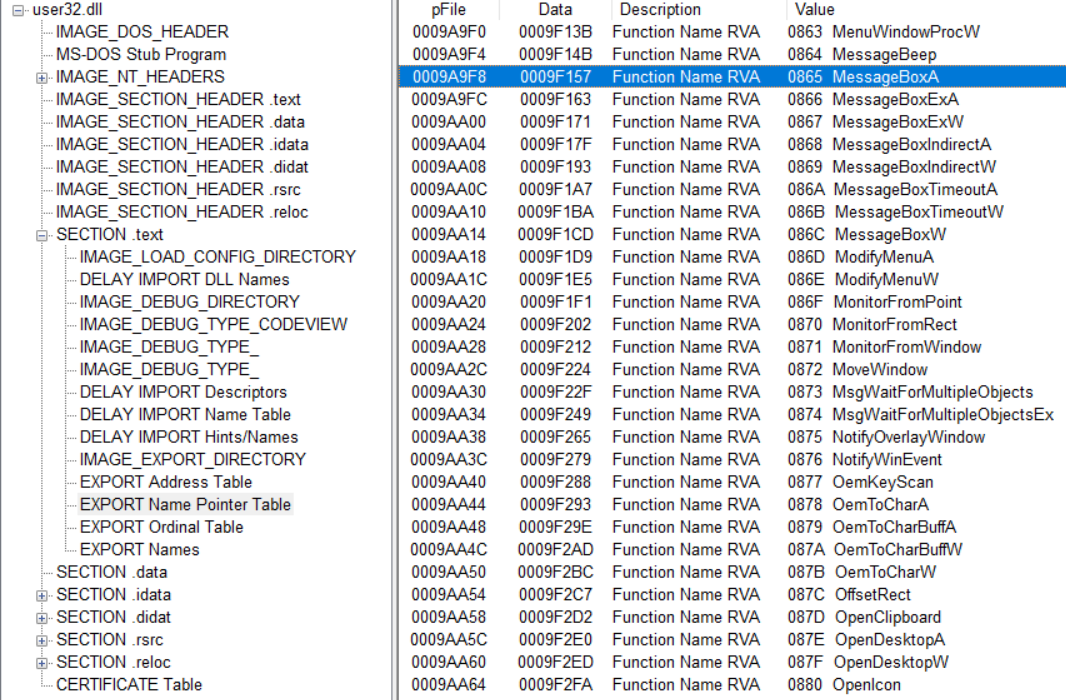


图 25 Name Pointer Table中的MessageBoxA

在Ordinal Table找到序号（Value）0865项，Data为0287；而由于Ordinal Table中Data从3开始，故在Address Table中找到第0287-2项，即第0285项。

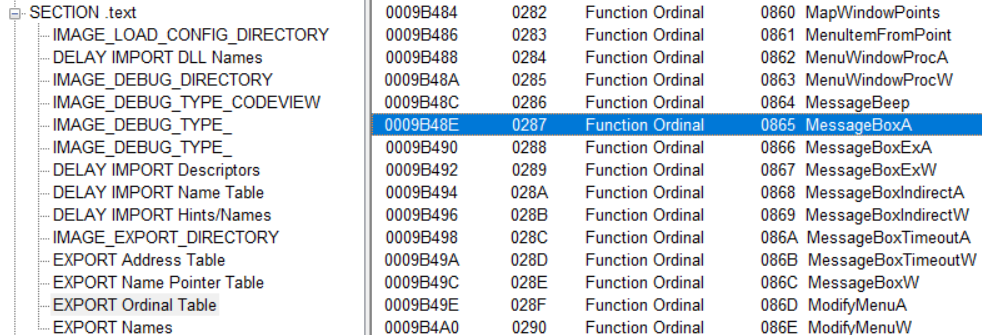


图 26 Ordinal Table中的MessageBoxA

Address Table中第0285项即0865项的Data为0007EE90，也即MessageBoxA的RVA（相对虚拟地址）。

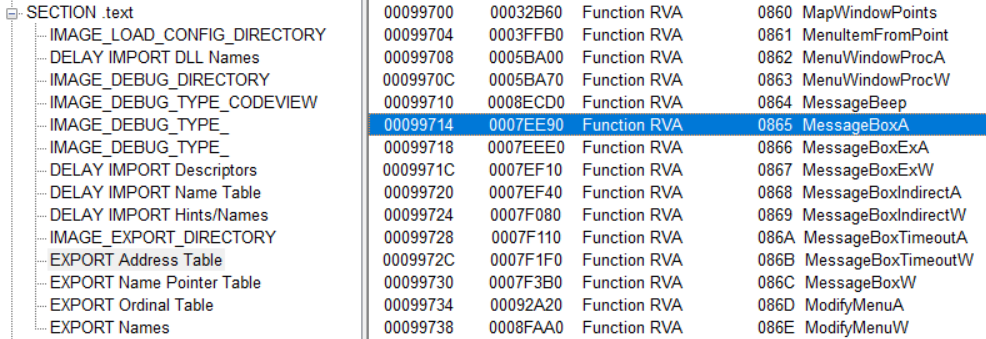


图 27 Address Table中的MessageBoxA

由于Windows XP之后的操作系统对真实地址做了相关保护，所以两地址不相同。

## 3.6 用UltraEdit修改notepad.exe程序的引入表

用UltraEdit修改notepad.exe程序的引入表，使得该程序仅可以从kernel32.dll中引入LoadLibrary和GetProcAddress函数，而不从user32.dll导入任何函数。

### 3.6.1 获取Kernel32.dll基地址

所有的进程都维护一个PEB（进程环境块结构）；这个结构体可以在搜索引擎查找，也可以使用windbg工具获取；首先用windbg加载一个进程。我们使用“!peb”指令，解析当前进程的PEB结构：

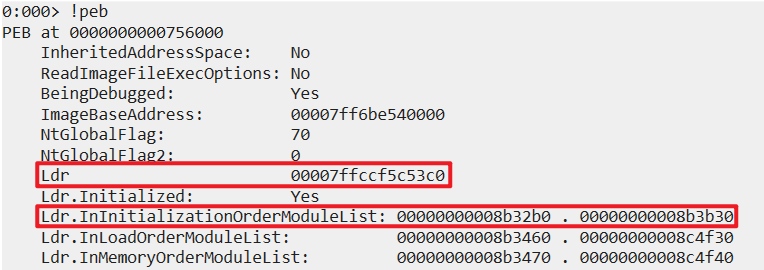


图 28 解析当前进程PEB结构

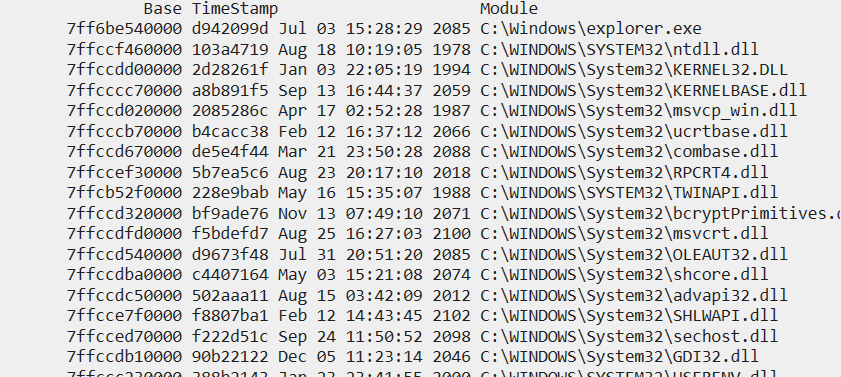


图 29 DLL基地址

可以看到PEB的基地址0x00756000、LDR的基地址0x7ffccf5c53c0和LDR中InInitializationOrderModuleList的地址范围。在在PEB结构中，维护着\_PEB\_LDR\_DATA（偏移0xC）结构保存着进程加载的所有的Module信息（包括Kernel32.dll，基地址）下面我们通过“dt nt!\_peb 00756000 -r”的指令来获取LDR：

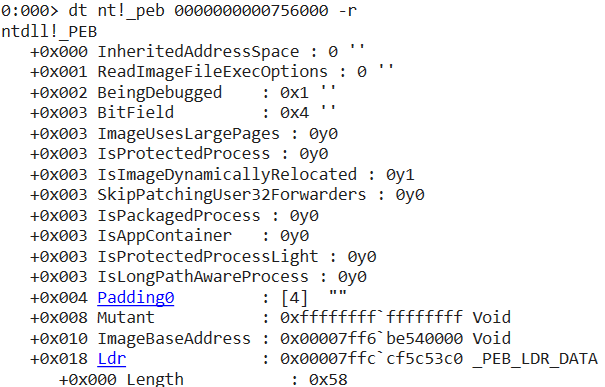


图 30 详细列出PEB结构

根据获取到的LDR去获取双向链表结构：

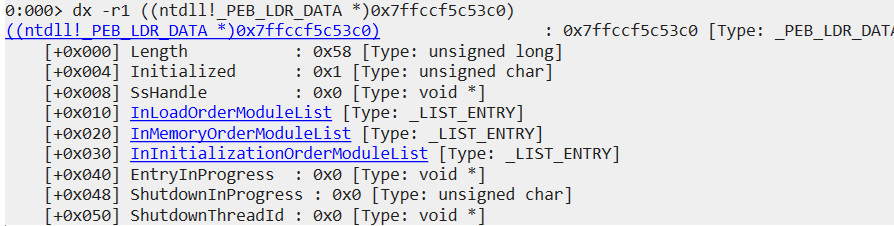


图 31 获取LDR中的双向链表结构

接下来我们需要尝试从InInitializationOrderModuleList链表中获取Kernel32.dll基地址（通过FLINK指针遍历链表）。

第一个节点，对应的Module的基地址是0x8b32b0，对照一下可以发现是ntdll.dll：

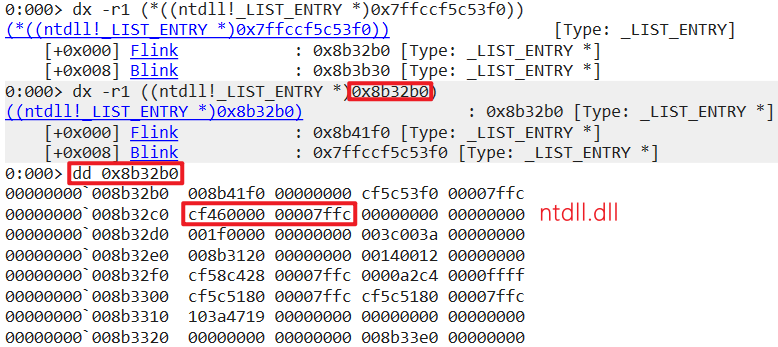


图 32 找到ntdll.dll

NEXT指向的下一个节点，对应的Module基地址是0x8b41f0，对照图5-6发现是KERNELBASE.dll：

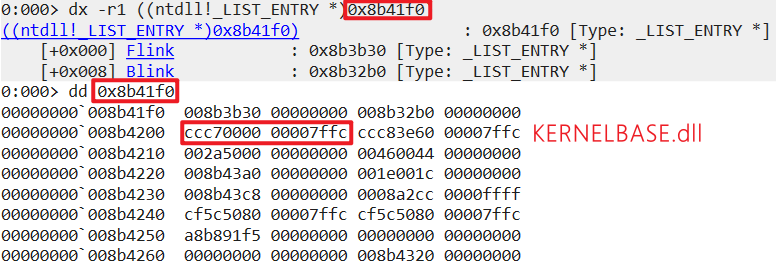


图 33 找到KERNALBASE.dll

继续向下找，发现第三个节点对应的Module基地址正是Kernel32.dll的基地址0x8b3b30：

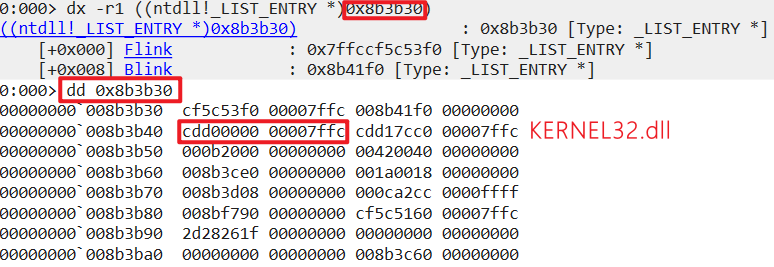


图 34 找到KERNEL32.dll

通过上述过程，我们完成了从PEB中获取某Module基地址的过程，kernel32.dll在不同Windows系统上InInitializationOrderModuleList的位置有些区别，上述过程在Windows 10 32bits完成，发现keernel32.dll在InInitializationOrderModuleList的第三个节点的位置。大家需要在自己测试系统上根据上述步骤寻找kernel32.dll的位置。（基地址开机随机，节点位置固定）。

### 3.6.2 加载User32.dll

User32.dll可能程序中并没有加载，此时需要LoadLibrary加载；而LoadLibrary函数位于kernel32.dll中；只需要从Kernel32.dll中找到LoadLibrary函数即可。

虽然在某一DLL中函数的偏移是固定的，但是出于通用性的考虑，我们并不直接硬编码偏移，而是Shellcode实现在DLL中搜索特定的函数。

**(1)** Shellcode编码——API hash

在shellcode中，为了尽可能地压缩Shellcode大小，可以将用到的API做Hash处理，在DLL中搜索API时直接比较Hash值即可，这样可以重复利用Shellcode代码，也避免了将目的API暴露在Shellcode中。

常见的API Hash方式是ror13。其Hash算法的汇编代码如下：

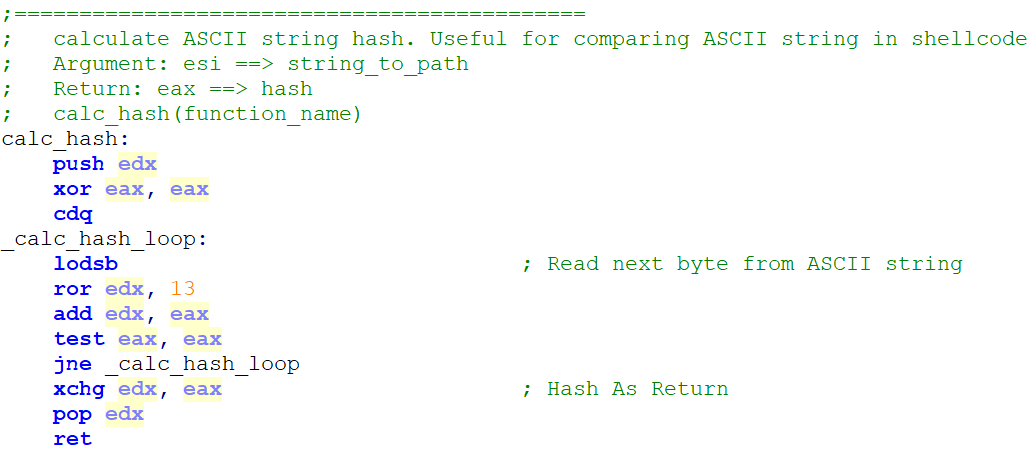


图 35 ror13汇编代码

**(2) Shellcode实现**

由于FS:30始终指向PEB结构，首先我们找到PEB的基地址，然后根据其中LDR在PEB中固定的相对偏移找到LDR的相对虚拟内存地址，同理根据InInitializationOrderModuleList在LDR中的相对偏移找到其相对虚拟内存地址，根据上述windbg解析过程，实现如下汇编代码：

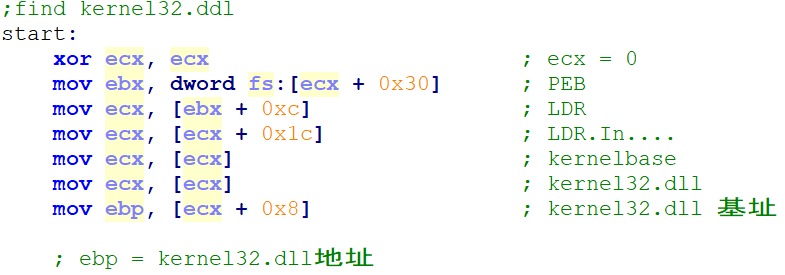


图 36 获取kernel32.dll基地址

获取Kernel32.dll基地址后，我们需要从中找到LoadLibraryA函数地址；这里使用一个通用函数，用于在指定DLL中根据DLL导出表和名称表寻找特定Hash值的函数。在导出表中NumberOfNames是DLL导出函数名的数目，AddressOfNames就是上导出函数名地址列表；通过遍历该列表，计算每个name的hash值与目标hash值对比来寻找函数。AddressOfFunctions是函数入口地址；AddressOfName/Ordinals保存着函数Name和入口的下标值列表。从Kernel32.dll获取LoadLibraryA的函数地址，Export Table的工作流程基本如下：

1. 首先定位到Kernel32.dll的导出表
2. 在到AddressOfNames地址，遍历搜索`LoadLibraryA`
3. 假如在AddressOfNames数组第index个位置找到，读取AddressOfNameOrdinals数组的第index个位置为ordinal。
4. 读取AddressOfFunctions数组的ordinal位置就是函数LoadLibraryA的RVA，加上Kernel32.dll的基地址就是该函数的地址。

该过程实现汇编代码如下：



图 37 遍历名称表，找到LoadLibraryA函数

得到LoadLibraryA函数后，我们就执行LoadLibraryA(“user32”)获取user32.dll模块基地址。

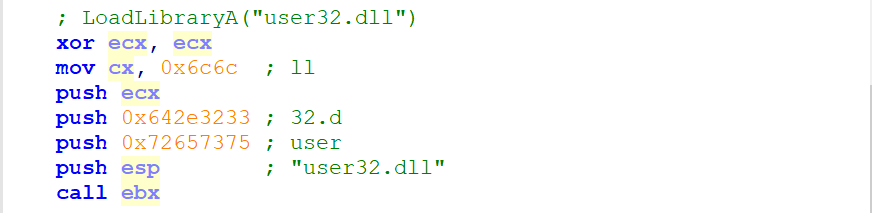


图 38 执行LoadLibraryA("user32.dll")

## 3.7 获取 MessageBoxA的函数地址

在代码节中写入部分代码利用上述导入的两个函数获取 MessageBoxA的函数地址，使得notepad.exe程序的原有功能正常。

刚刚已经得到了user32.dll基地址，根据同样的道理可以获得MessageBoxA函数地址：



图 39 遍历名称表，找到MessageBoxA函数

接下来需要在栈上布置MessageBoxA函数参数，最终调用MessageBoxA(NULL, “aaa”, “bbb”, 1)；注意保持堆栈平衡，以便执行完shellcode后可以顺利恢复原执行流程。

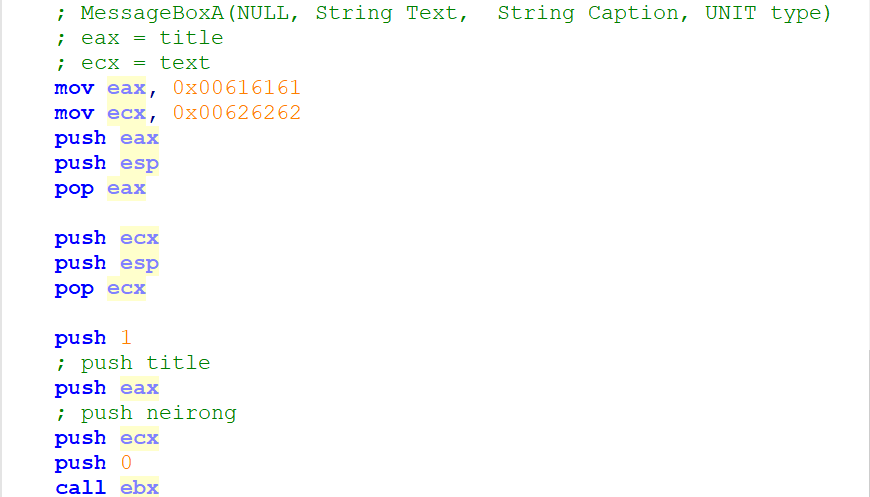


图 40 执行MessageBoxA函数

## 3.8 汉化PEview.exe程序

熟悉eXeScope工具的实用，并利用该工具汉化PEview.exe程序



图 41 PEview.exe汉化前

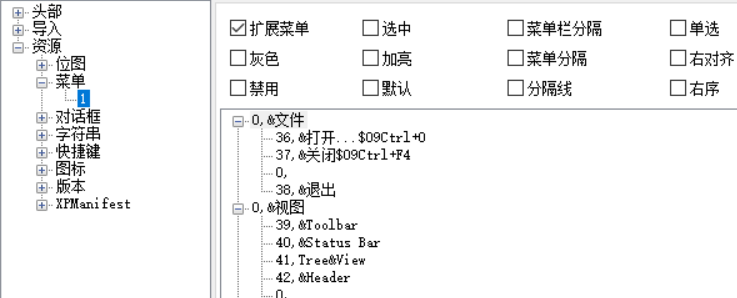


图 42 使用eXeScope进行汉化



图 43 PEview汉化后

## 3.9 修改PEview.exe的图标

熟悉资源表。利用PEview.exe熟悉资源表的结构，利用PEview.exe分析PEview.exe程序；用eXeScope修改PEview.exe，使得该文件的图标变成其他ico。

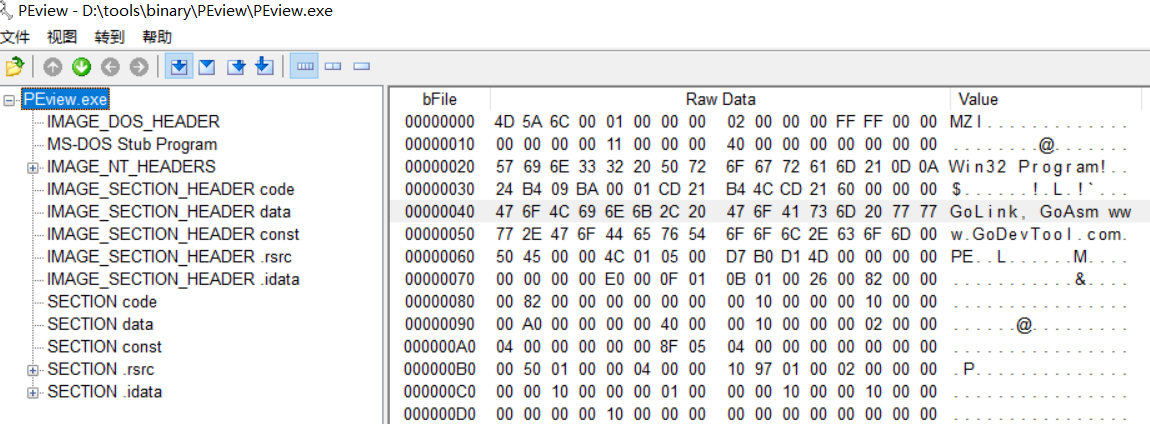


图 44 使用PEview查看PEview.exe

使用eXeScope将PEview.exe的图标变为eXeScope的图标。

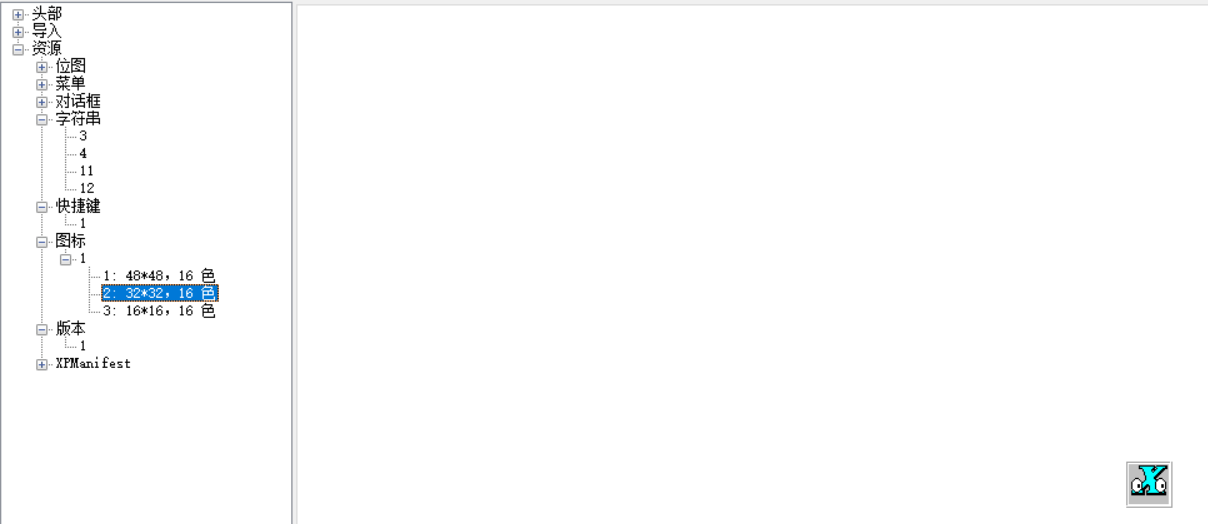


图 45 将32\*32大小的图标变为eXeScope的图标



图 46 修改成功



图 47 将16\*16大小的图标变为Ollydbg的图标



图 48 修改成功

# 四、实验总结与问题的解决

## 4.1 实验总结

1. **如何遍历DLL导出函数寻找目标函数。**

首先根据该DLL的基地址和PE头的偏移找到PE头的真实地址，然后根据找到的PE头的基地址和导出表的偏移，获得导出表的真实地址，再根据导出表的基地址和名称表的偏移，得到名称表的真实地址。最后以目标函数名称按小端序排列的十六进制作为目标遍历名称表，获得该目标函数的序号，然后去AddressOfOrdinals 结构里面查找该序号，获得该目标函数的真实地址。

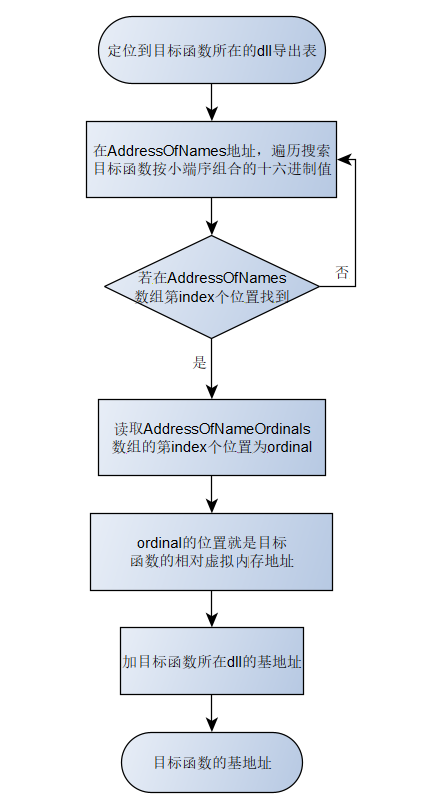


图 49 遍历DLL查找目标函数流程

1. **RVA和基地址，虚拟内存地址有何联系。**

虚拟内存地址（VA）= 进程的基地址 + 相对虚拟内存地址（RVA）。

在32 位Windows中用户的 PE 文件被操作系统加载进内存后，PE 对应的进程支配了自己独立的 4GB 虚拟空间。在这个空间中定位的地址称为虚拟内存地址（Virtual Address，VA），所以虚拟内存地址的范围是 00000000h~0ffffffffh。相对虚拟内存地址（Reverse Virtual Address, Rva）是相对于基地址的偏移，即 RVA 是虚拟内存中用来定位某个特定位置的地址，该地址的值是这个特定位置距离某个模块基地址的偏移量，所以说 RVA 是针对某个模块而存在的。在 PE 中进程本身的虚拟内存地址被解释为：进程的基地址+相对虚拟内存地址。

## 4.2 遇到问题的解决方案

根据目标函数名在名称表中遍历查找时，无法找到，后经查找资料发现此处转化为十六进制的函数名在查找代码中应使用小端序进行查找操作。

# 五、参考文献

1. Eastmount. [网络安全自学篇] 六十二.PE文件逆向之PE文件解析、PE编辑工具使用和PE结构修改（三）. https://blog.csdn.net/eastmount/article/details/105080804
2. 呱呱生. 【汉化教程】--第3课：UltraEdit软件的十六进制汉化. https://www.52pojie.cn/thread-613855-1-1.html
3. dldw777. 谈谈eXeScope这款exe修改器. https://blog.csdn.net/dldw777/article/details/83273565
4. 擒贼先擒王. PE文件结构详解 --（完整版）. https://blog.csdn.net/freeking101/article/details/102752048?utm\_medium=distribute.pc\_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.control&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.control
5. 苏紫方璇. [原创] eXeScope破解记录. https://www.52pojie.cn/forum.php?mod=viewthread&tid=358993&highlight=exescope
6. 小木曾雪菜. Galgame汉化中的逆向 (一)：文本加密（压缩）与解密. httht://www.52pojie.cn/forum.php?mod=viewthread&tid=1194555&highlighh=galgame%2B%BA%BA%BB%AF
7. 呱呱生. [原创] 【汉化教程】--第2课：后缀exe程序汉化菜单界面. https://www.52pojie.cn/thread-613071-1-1.html