编号：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验 | 一 | 二 | 三 | 四 | 五 | 六 | 七 | 八 | 总评 | 教师签名 |
| 成绩 |  |  |  |  |  |  |  |  |  |  |

武汉大学国家网络安全学院

课程实验(设计)报告

题 目： 实验二：PE文件格式分析

专业(班)： 2021级信安6班

学 号： 2021302181156

姓 名： 赵伯俣

课程名称： 软件安全实验

任课教师： 赵磊

2030年 10 月 18日

**目 录**

[实验2 PE文件格式分析 1](#_Toc13759)

[1.1实验名称 1](#_Toc1597)

[1.2实验目的 1](#_Toc14055)

[1.3实验步骤及内容 1](#_Toc9388)

[1.4实验关键过程、数据及其分析 3](#_Toc27007)

[1.4.1 分析hello-2.5.exe的16进制数据 3](#_Toc2127)

[1.4.2 对hello-2.5.exe调试 7](#_Toc23797)

[1.4.3 修改程序使其只能弹出第二个对话框。 8](#_Toc8440)

[1.4.4熟悉各类PE文件格式查看工具 9](#_Toc20349)

[1.4.5分析函数导入基本原理 10](#_Toc21231)

[1.4.6修改文件弹出第三个对话框 10](#_Toc24457)

[1.4.7寻找user32.dll文件，分析其引出表 12](#_Toc26907)

[1.4.8找出函数MessageBoxA的地址，并验证该地址是否正确 13](#_Toc8624)

[1.4.9修改程序引入表不使用user32.dll 15](#_Toc3524)

[1.4.10使修改后的程序恢复正常。 17](#_Toc1791)

[1.4.11分析PEview.exe程序了解资源节结构。 19](#_Toc9753)

[1.4.12分析ICO文件的格式信息。 21](#_Toc7899)

[1.4.13图标替换 22](#_Toc3861)

[1.4.14软件手工汉化 23](#_Toc22740)

[1.4.15打造最小PE文件 24](#_Toc28294)

[1.4.16如何编码实现PE程序中对应资源的提取与替换？涉及到哪些关键API函数。 32](#_Toc19986)

[1.4.17当目标程序的图标资源为多个时，每个图标资源分别对应着哪里？此时图标替换策略应该如何调整？ 33](#_Toc8505)

[1.4.18资源节与恶意代码有何关联？ 34](#_Toc5968)

[1.4.19什么是HOOK？其与本章学习有何关系？ 34](#_Toc22426)

[1.5实验体会和拓展思考 35](#_Toc7034)

## 实验2 PE文件格式分析

### 1.1实验名称

PE文件格式分析

### 1.2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. 自己打造一个尽可能小的PE文件

### 1.3实验步骤及内容

**第一阶段：PE文件格式初步分析与调试**

* 分析例子程序hello-2.5.exe
  + 使用二进制编辑工具观察PE文件例子程序hello-2.5.exe的16进制数据
  + 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
  + 使用PE编辑工具修改该程序，使得该程序仅弹出第二个对话框

**第二阶段：函数的引入与引出机制**

* 熟悉各类PE文件格式查看和编辑工具：
  + 结合hello-2.5.exe熟悉PE文件头部、引入表的结构
* 熟悉函数导入的基本原理
* 手工修改hello-2.5.exe程序，使得其可以弹出第三个对话框（提示框标题为“武汉大学信安病毒实验”，内容为：你的姓名+学号）
* 找到系统System32目录下的user32.dll文件，用二进制编辑器打开并分析该文件引出表，找出函数MessageBoxA的地址，并验证该地址是否正确。【视频中采用的方法是从文件中定位查找，在实验报告中请直接从实际进程（如test.exe）内存空间进行定位查找】

**第三阶段：在目标程序中新增功能代码**

* 用二进制编辑工具修改hello-2.5.exe程序的引入表，使该程序仅可以从kernel32.dll中引入LoadLibrary和GetProcAddress函数，而不从user32.dll导入任何函数。
* 在代码节中写入部分代码利用这两个函数获取MessageBoxA的函数地址，使hello-2.5.exe程序原有功能正常。

**第四阶段：图标资源替换与软件手工汉化**

* 资源表资源操作实践
  + 利用PEview.exe分析PEview.exe程序
  + 用二进制编辑工具修改PEview.exe，使得该文件的图标变成csWhu.ico
  + 熟悉eXeScope工具的实用，并利用该工具汉化PEview.exe程序

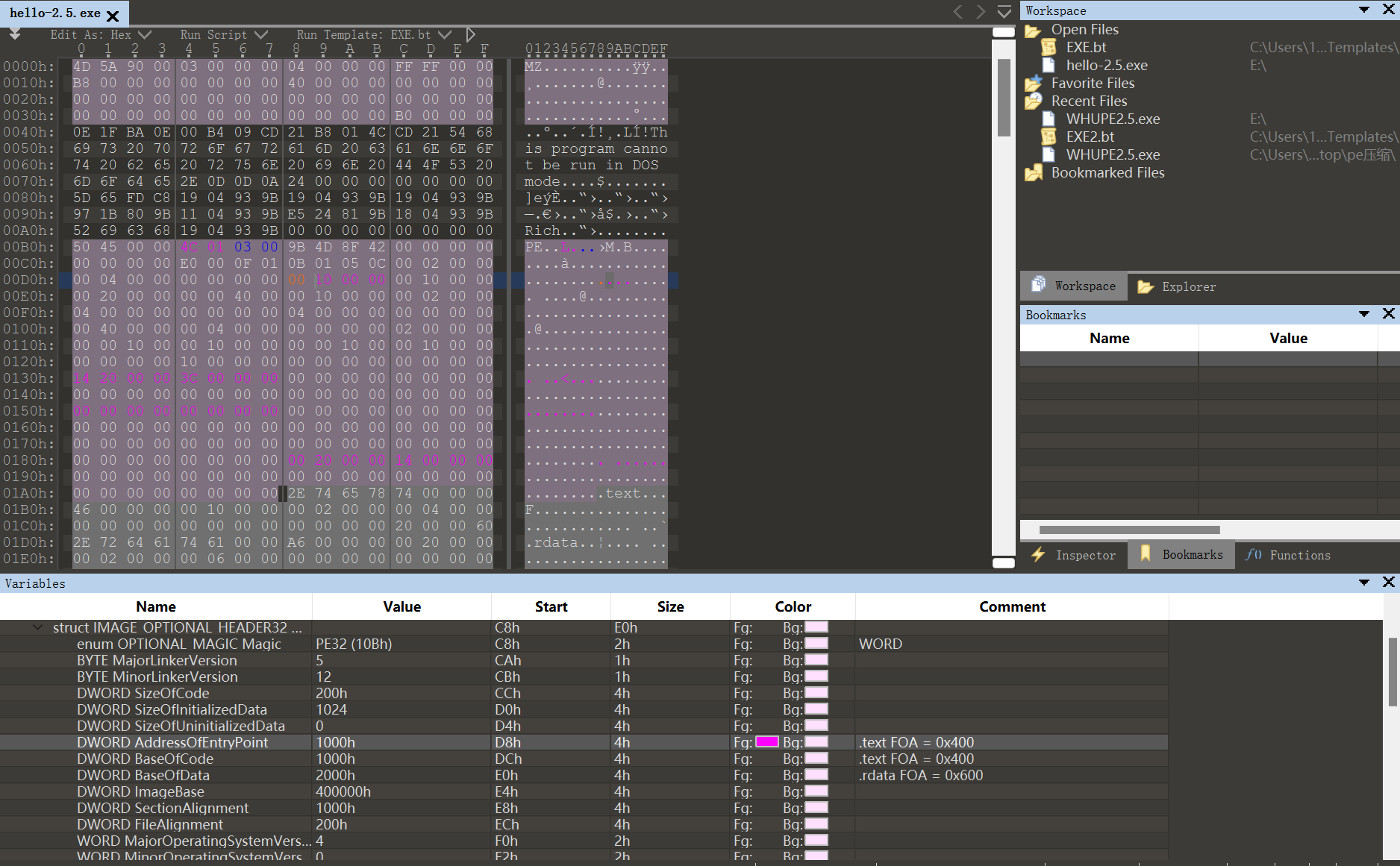
**课后习题思考：**

* 如何打造最小的PE文件
  + 修改WHU\_PE-2.5.exe文件，保持该文件的功能不变，使得该文件大小尽可能小
  + 本文件的最小极限可能是多少？结合tinyPE一文进行描述。<http://www.phreedom.org/research/tinype/>
* 如何编码实现PE程序中对应资源的提取与替换？涉及到哪些关键API函数。
* 当目标程序的图标资源为多个时，每个图标资源分别对应着哪里？此时图标替换策略应该如何调整？
* 资源节与恶意代码有何关联？
* 什么是HOOK？其与本章学习有何关系？

### 1.4实验关键过程、数据及其分析

#### 1.4.1 分析hello-2.5.exe的16进制数据

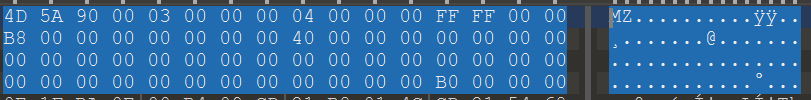
使用010Editor将该PE文件打开后并下载对应的模板后如下图所示。



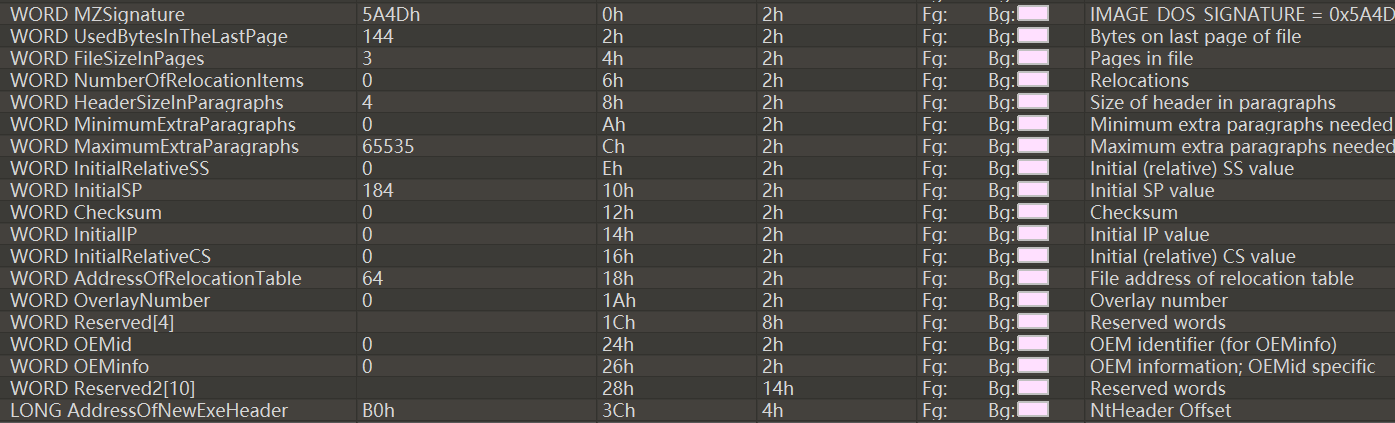
该可执行程序可以分为以下几个部分。

（1）DOS头结构。

PE文件的开头是一个64字节的DOS头。这个DOS头只是为了兼容早期的DOS操作系统。DOS头的结构如下所示，

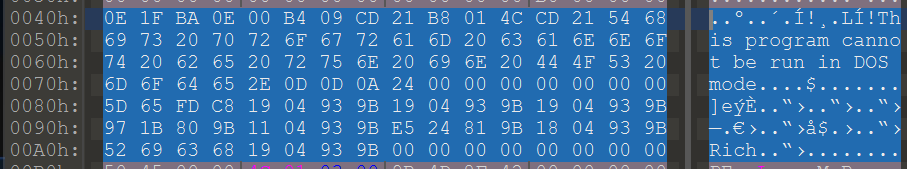


该DOS文件头在010Editor软件中打开模板的具体结构如下所示。其中展示了DOS头文件的每一个字节的数据字节大小和值以及其所代表的内容。



（2）DOS STUB结构

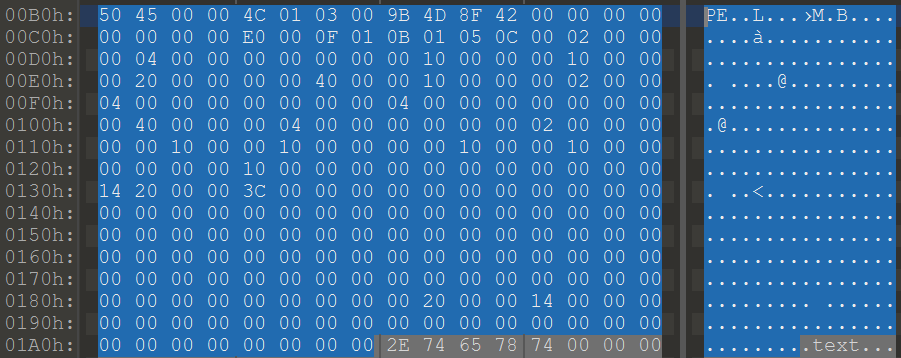
该PE文件的DOS STUB结构如下图所示



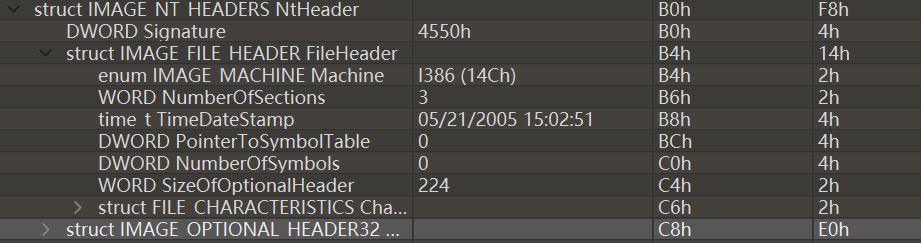
该结构是链接器链接执行文件的时候加入的部分数据，一般是“This program must be run under Microsoft Windows”。这个可以通过修改链接器的设置来修改成自己定义的数据。在本次实验给出的PE文件中显示为This is program cannot be run in DOS mode

（3）PE相关结构NT映像头（IMAGE\_NT\_HEADER）结构

该结构包含许多PE装载器用到的重要字段。执行体在支持PE文件结构的操作系统中执行时，PE装载器将从IMAGE\_DOS\_HEADER结构中的e\_lfanew字段里找到PE Header的起始偏移量，加上基址得到PE文件头的指针。



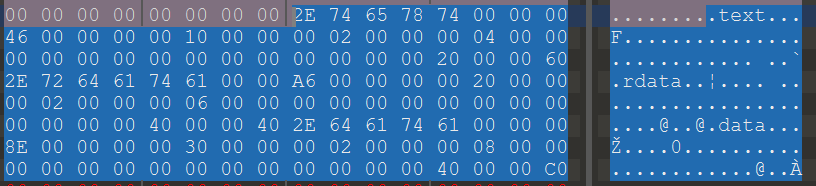
IMAGE\_NT\_HEADER的内部结构使用模板打开后如下图所示。



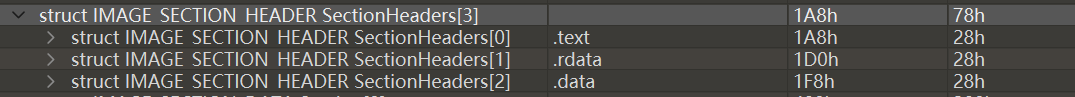
其中包含三个数据结构，第一个是Signature，PE头的标识，双字结构，内容为50 45 00 00代表这是一个PE文件；第二个是IMAGE\_FILE\_HEADER（映像头文件）结构包含了文件的物理层信息及文件属性；第三个是IMAGE\_OPTIONAL\_HEADER（可选映像头）是一个可选的机构，定义了更多的数据， 总共224个字节，最后128个字节为数据目录。

（4）区块表。

在PE相关结构NT映像头之后便是PE文件的块表结构其十六进制数据如下图所示，



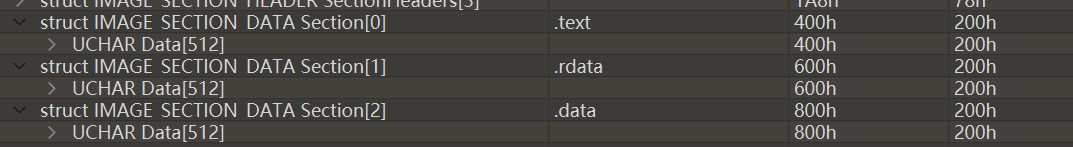
在010Editor中用模板打开后如下图所示。



区块表包含每个块在映像中的信息（如位置、长度、属性），分别指向不同的区块实体。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。由此可得区块表可以看作区块的一个索引。

（5）区块。

在给出的PE文件中共有三个区块，使用模板打开后如下图所示。



其中第一块的属性是.text，是默认的代码区块，它的内容全是指令代码，链接器把所有目标文件的text块连接成一个大的．text块，使用Borland C++，编译器产生的代码存放在CODE的区域里。

第二块是的属性是.rdata,是默认只读数据区块，但程序中很少用到该块中的数据，一般两种情况用到，一是MS的链接器产生EXE文件中用于存放调试目录，二是用于存放说明字符串，如果程序的DEF文件中指定了DESCRIPTION，字符串就会出现在rdata中。

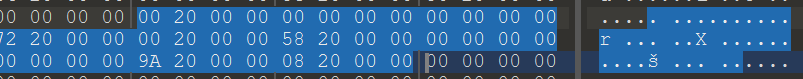
第三块的属性是.data，默认的读／写数据块，全局变量，静态变量一般放在这个区段。

在给出的PE文件中并不包含所有的区块类型，其未包含的区块类型即其内容如下表所示。

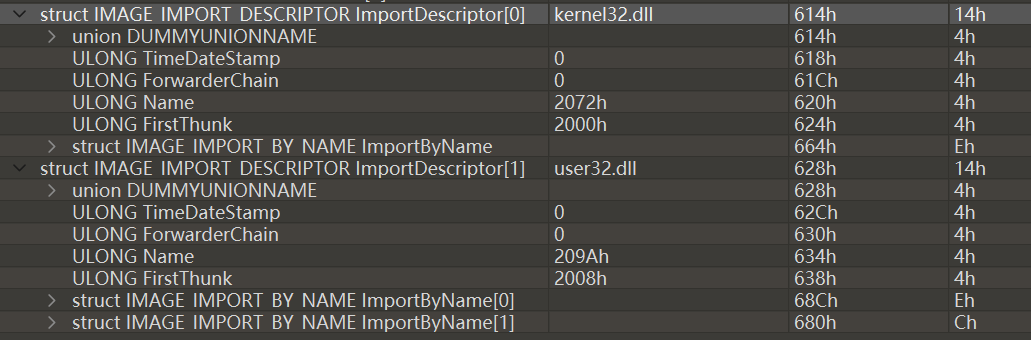
|  |  |
| --- | --- |
| 区块类型 | 内容 |
| .idata | 包含其他外来的DLL的函数及数据信息，即输入表，将．idata区块合并成另一个区块已成为一种惯例，典型的是．rdata区块，默认的，链接器只在创建一个Release模式的可执行文件时才能将idata合并到另外一个区块中 |
| .edata | 输出表，当创建一个输出API或数据的可执行文件时，连接器会创建一个，EXP文件，这 个．EXP文件包含一个．edata区块，其会被加载到可执行文件中，经常被合并到。text或．rdata区块中 |
| .rsrc | 资源，包括模块的全部资源，如图标，菜单，位图等，这个区块是只读的，无论如何不应该吧它命名为．rsrc以外的名字，也不能合并到其他的区块里 |
| .bss | 未初始化的数据，很少在用，取而代之的是执行文件的．data区块的的VirtualSize被扩展大的空间里用来装未初始化的数据。 |
| .crt | 用于C+＋运行时（CRT）所添加的数据 |
| .tls | TLS的意思是线程局部存储器，用于支持通过＿declspec(thread）声明的线程局部存储 变量的数据，这包括数据的初始化值，也包括运行时所需要的额外变量 |
| .reloc | 可执行文件的机制重定位，基址重定位一般仅D11需要的 |
| .sdata | 相对于全局指针的可被定位的 短的读写数据 |
| .pdata | 异常表，包含CPU特定的IAMGE\_RUNTIME\_FUNTION\_ENTRY结构数组，DataDirectory中 的image\_direcTory\_entry\_ExcEPTION指向它 |
| .didat | 延迟装入输入数据，在非Release模式下可以找到 |

（6）输入表

使用010Editor查看给出的PE文件的输入表对应的16进制数据如下图所示。



使用模板将其打开后得到的结构如下图所示。

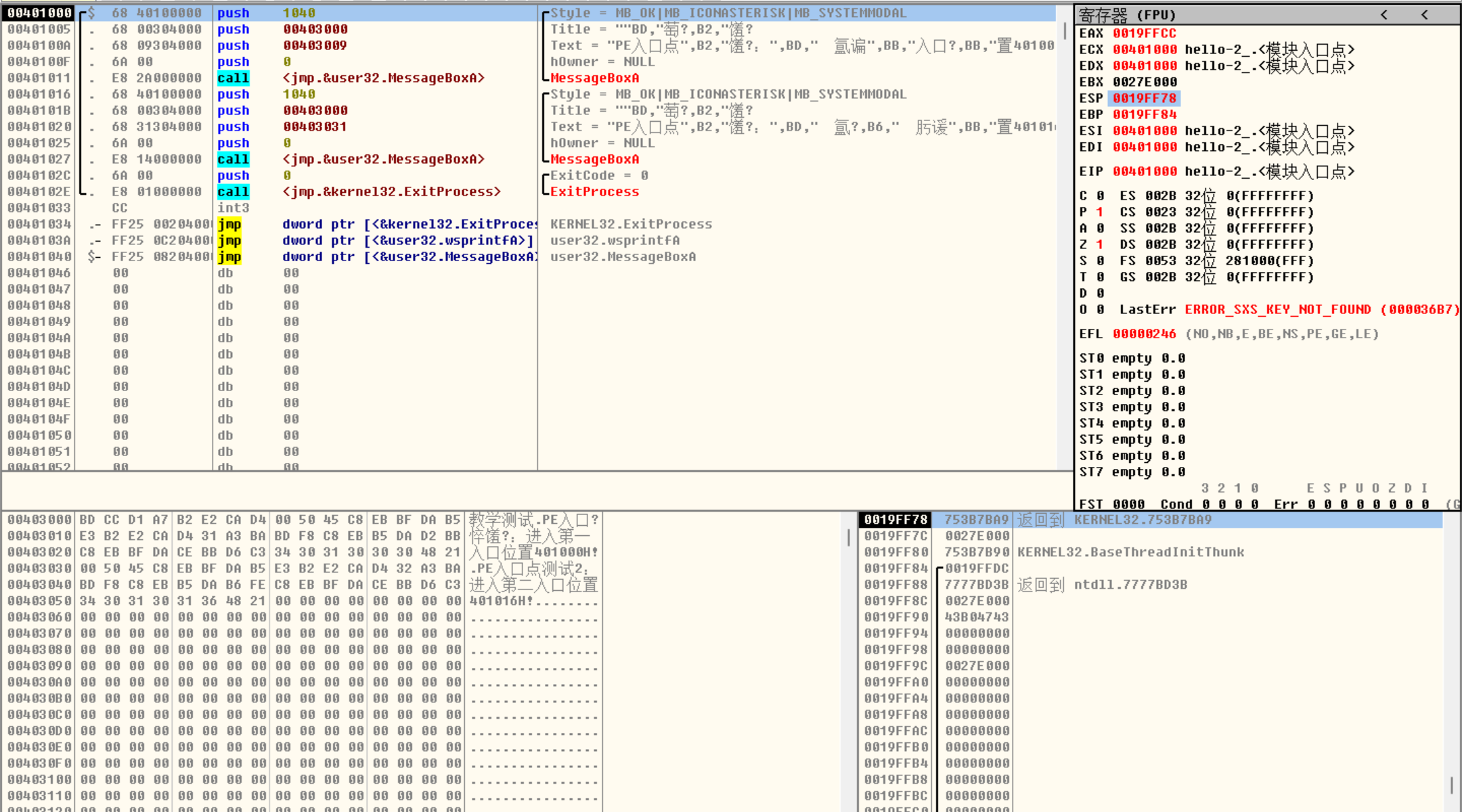


输入表以一个 IAMGE\_IMPORT\_DESCRITPTOR 数组开始，每个被PE文件隐式地链接进来的DLL都有一个IID，在这个数组中没有字段指出该结构数组的项数，但他最后一个单元是NULL。输入表中有五个结构如下所示。

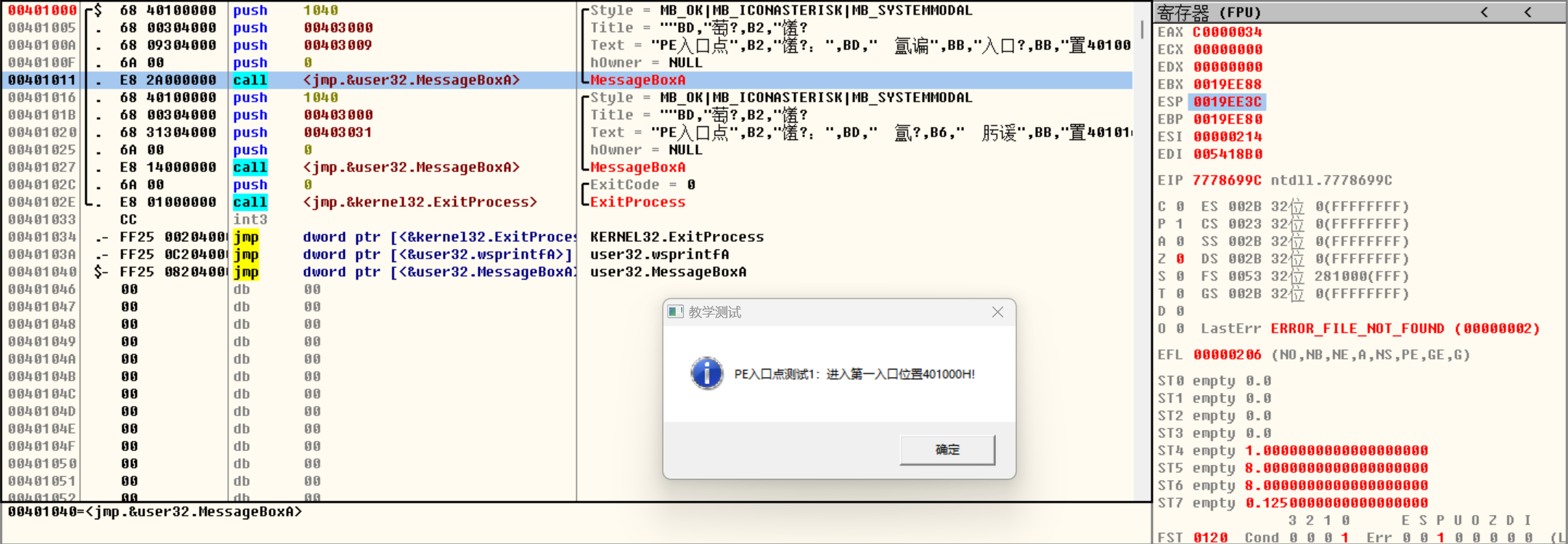
1. OriginalFirstThunk，它指向输入名称表（INT），输入名称表是一个 IMAGE\_THUNK\_DATA 结构的数组，数组中的每个 IMAGE\_THUNK\_DATA 结构的成员 AddressOfData都指向IMAGE\_IMPORT\_BY\_NAME结构；
2. TimeDateStamp，一个32位时间戳，
3. ForwarderChain，当程序引用一个DLL中的API，而这个API又引用别的DLL的API时使用，这种情况很少出现；
4. Name，它表示DLL 名称的相对虚地址；
5. FirstThunk：它指向输入地址表（IAT），输入地址表是一个 IMAGE\_THUNK\_DATA 结构的数组。

#### 1.4.2 对hello-2.5.exe调试

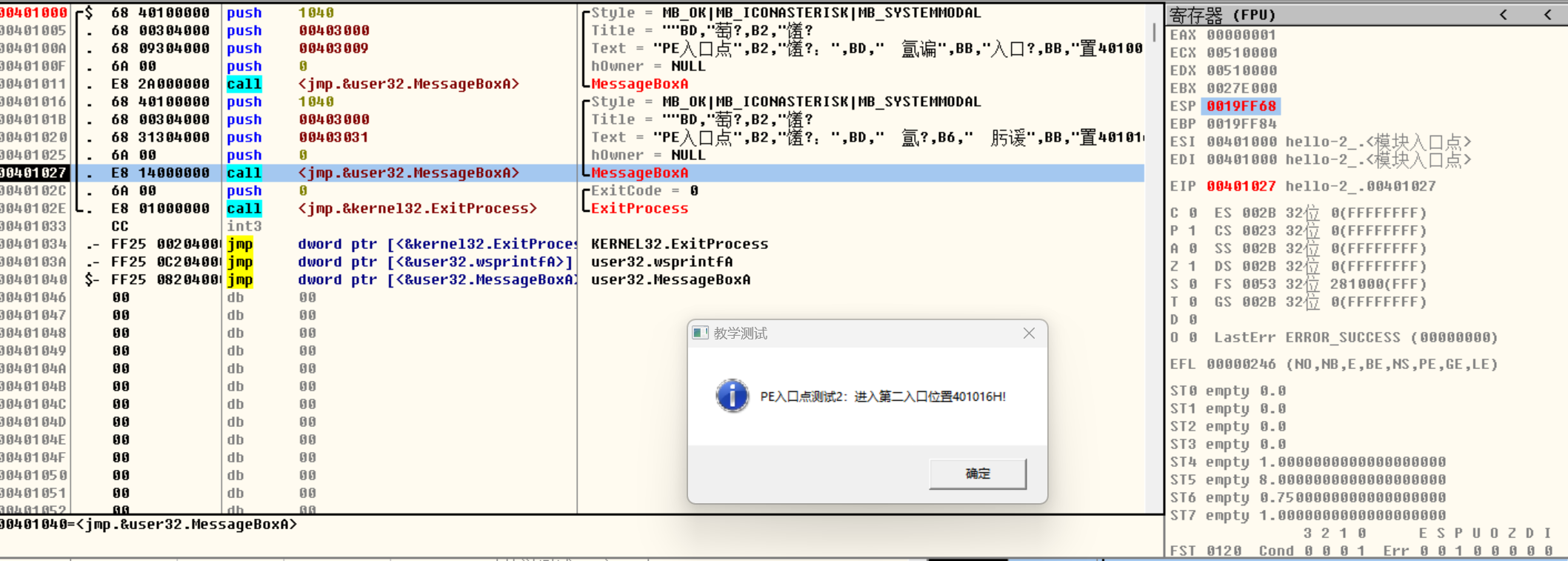
开始调试前程序的寄存器等信息如下图所示。



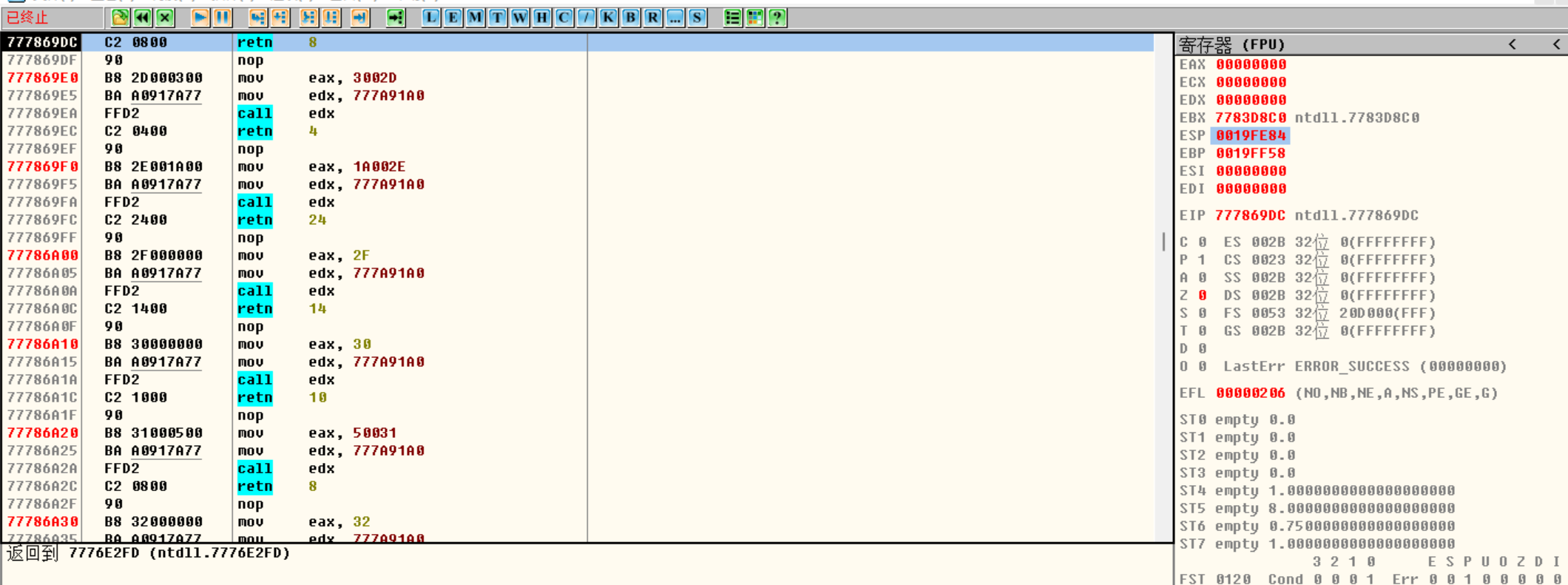
在运行到第一个窗口弹出时的寄存器内容如下图所示。



在运行到第二个窗口弹出时的寄存器内容如下图所示。



程序运行结束时的寄存器信息如下图所示。



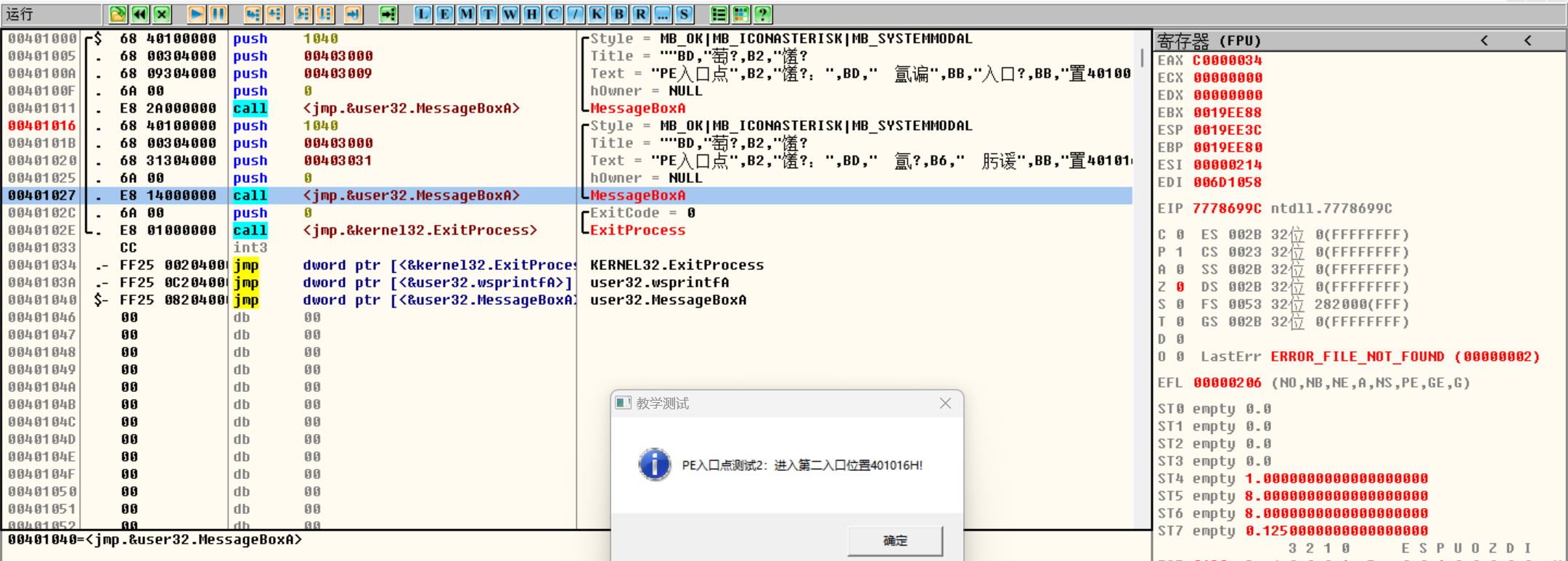
#### 1.4.3 修改程序使其只能弹出第二个对话框。

要修改程序只弹出第二个对话框我们只需要将程序代码段的入口地址该为第二个调用第二个窗口函数的地址。

即将NT文件头结构中OptionalHeader字段中的AddressOfEntryPoint程序入口点地址修改为1016h即可使程序无需弹出第一个对话框即可直接弹出第二个对话框。



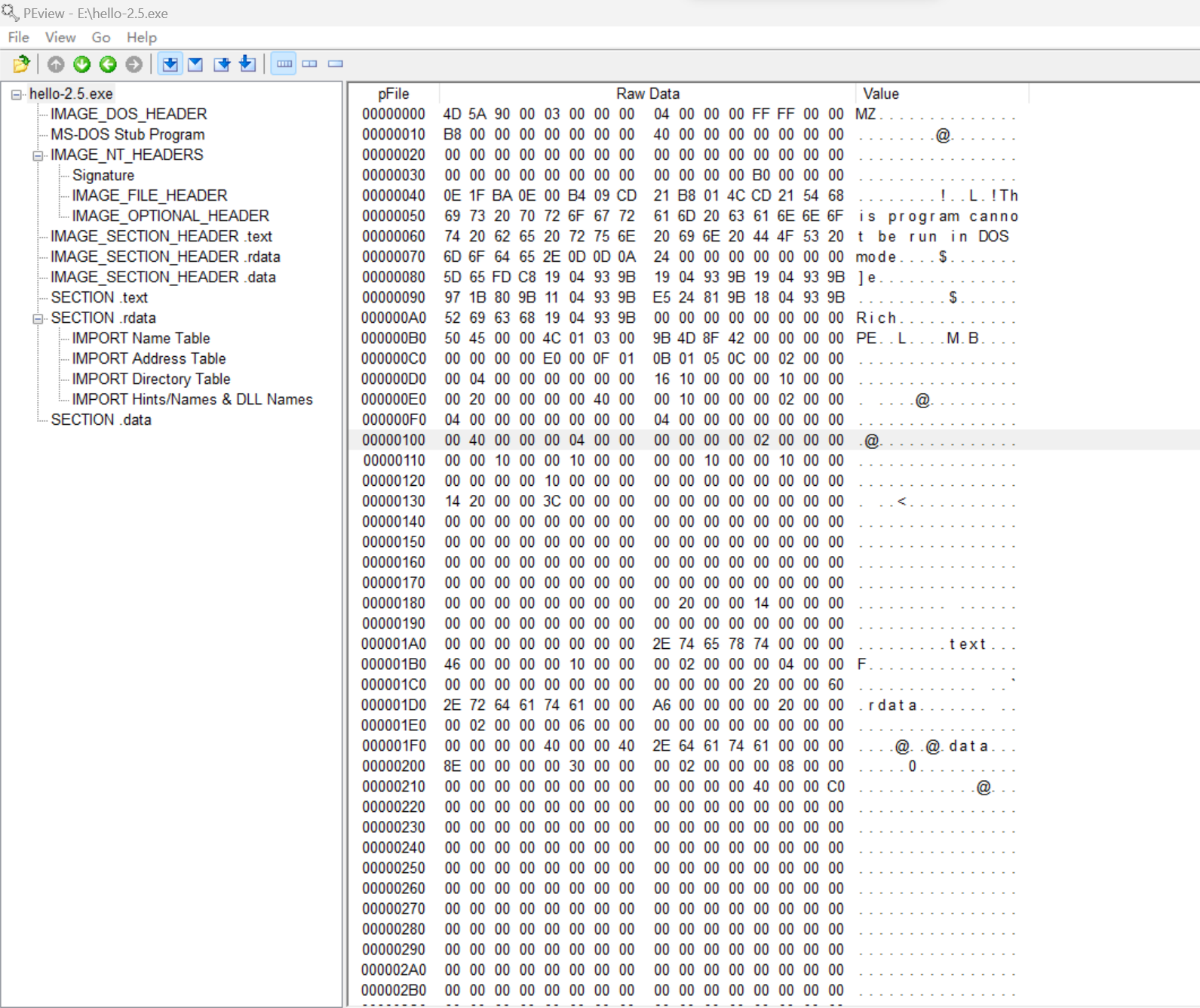
修改完成后对程序进行调试jie'guo结果如下图所示。



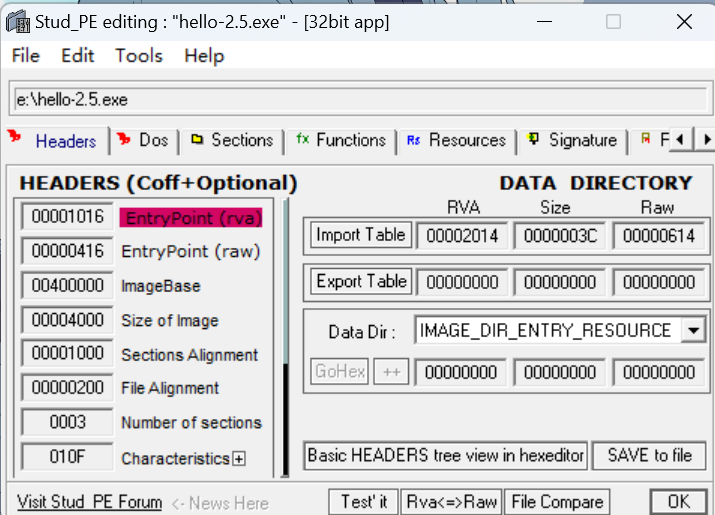
可以看到程序从00401016h位置开始运行，之后直接弹出第二个弹窗。

#### 1.4.4熟悉各类PE文件格式查看工具

使用PE\_view工具将给出的可执行文件打开后如下图所示。



实用工具Stud\_PE工具将给出的PE文件打开后的结果如下图所示



从中可以看到程序的入口地址是00001016h，导入表的位置为00002014h，大小为0000003ch，字段为00000614h，不存在导出表。

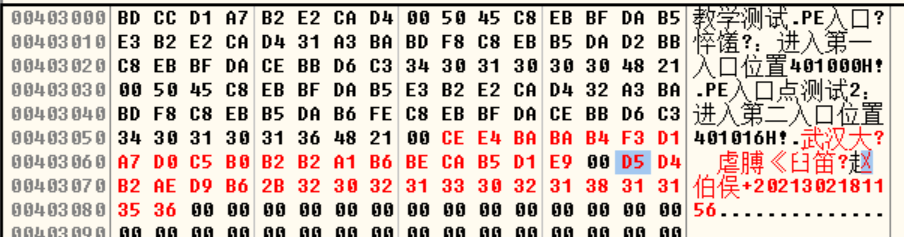
#### 1.4.5分析函数导入基本原理

当我们使用动态链接库中的函数，只有在程序运行时，库里面的代码才会被调入内存，程序不运行时程序只包含了一些关于要导入的链接库和函数的信息，这些信息就在导入表中。（如果有多个程序使用同一个动态链接库，WIndows在物理内中只留一份库的代码，仅通过分页机制将这份代码映射到不同进程的地址空间中）。

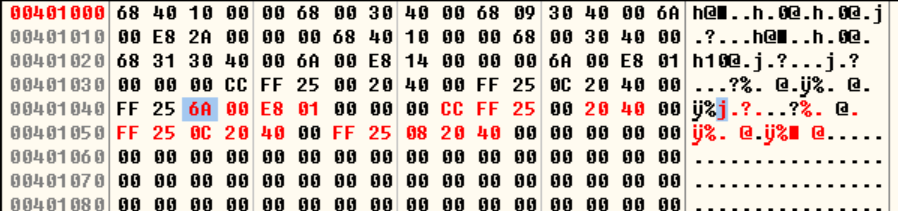
PE文件被装载的时候， Windows装载器会根据导入表中的某个结构处的RVA找到要导入的函数名，然后再根据这个函数名在内存中找到函数的地址，最后再用函数地址将导入表的又某个结构处的内容替换成真正的函数地址。然后我们我们每次用函数名调用函数的时候，系统就会通过函数名可以直接找到函数地址，然后就调用函数了。

#### 1.4.6修改文件弹出第三个对话框

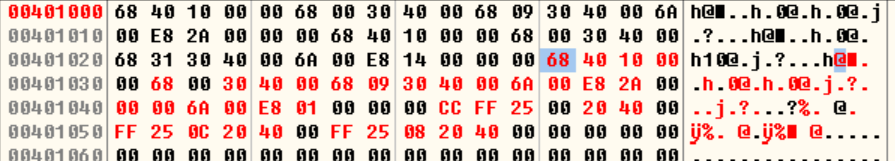
(1)首先修改数据段，在原有的数据段后加入标题“武汉大学信安病毒实验”和内容“赵伯俣+2021302181156”修改完后的数据段如下图所示。



（2）然后将文件的代码段中空出一块位置，通过观察第一个弹窗的程序部分我们发现调用弹窗的部分需要占用22字节，所以我们将函数最后的退出代码后移22个字节。



（3）填充第三个弹窗的调用。我们将空出的22个字节暂时复制第一个弹窗的内容。

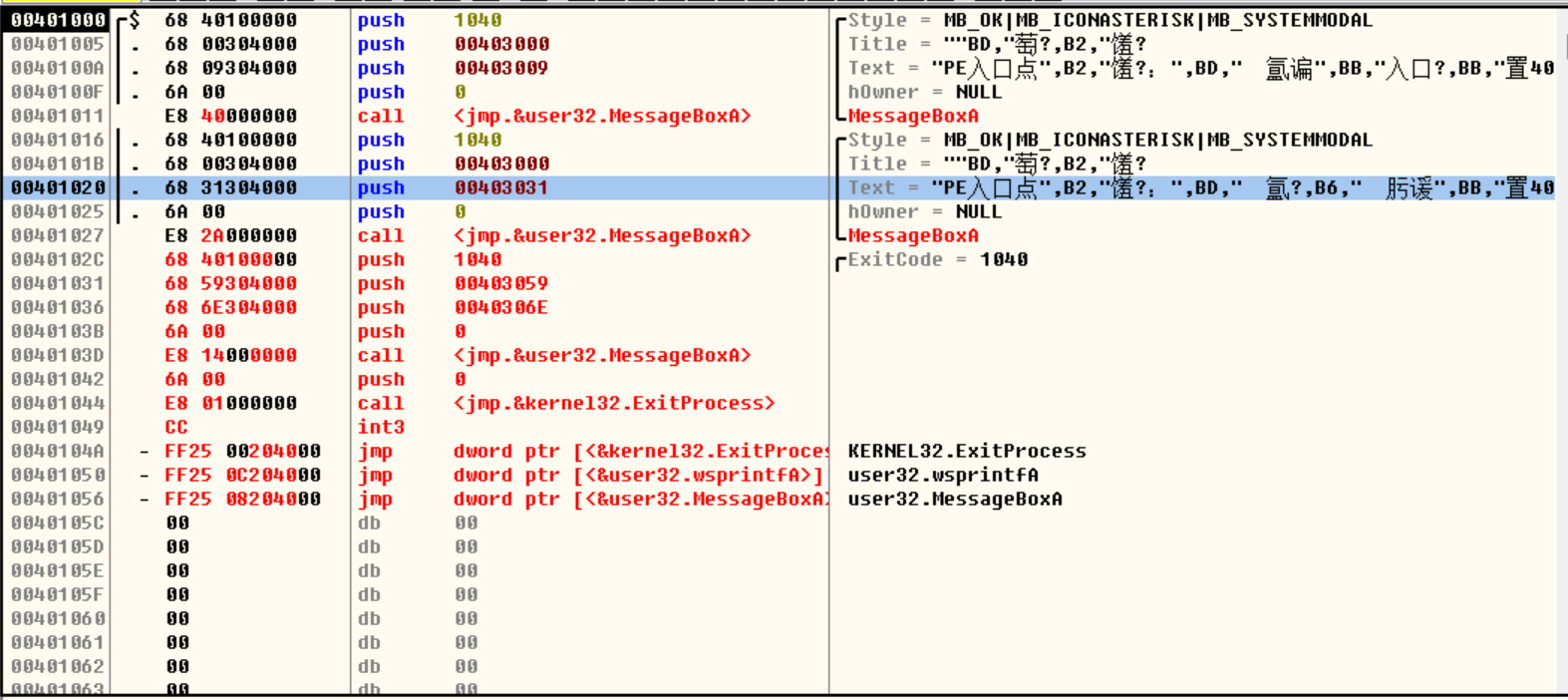


（4）修改数据段地址。由于我们的第三个弹窗直接复制的第一个弹窗的调用代码，所以我们需要修改其调用的数据段的地址，我们将数据段的标题地址改为403059H指向我们在第（1）步中编好的数据段的标题，然后将数据段的内容地址该为40306AH指向我们在第（1）步中编号的数据段的内容。修改后的结果如下图所示。



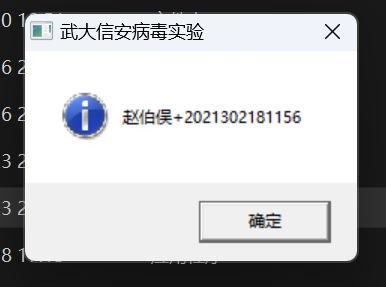
（5）修改跳转指令

由于我们将函数代码段后面部分整体后移所以导致所有的跳转指令失效，我们需要重新修改所有的call指令的偏移量。通过观察整个代码我们知道，函数调用的位置在401056h，修改第一个call的偏移量为56-16=40h，第二个call指令的偏移量为56-2c=2ah，第三个call指令的偏移量为56-42=14h。以下是修改后的结果。



（6）运行可执行文件。

将修改后的文件保存运行后查看有第三个弹窗弹出，代码修改成功。

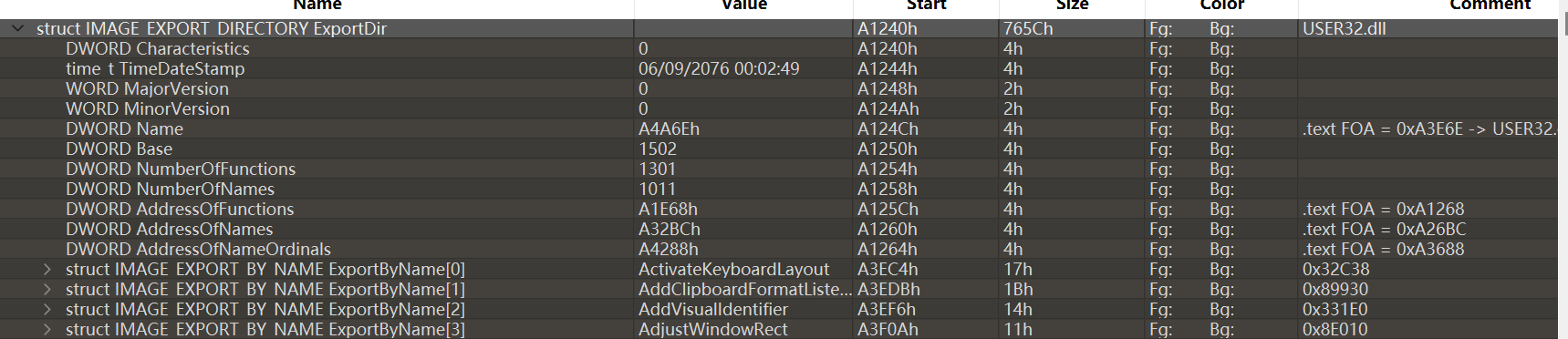


#### 1.4.7寻找user32.dll文件，分析其引出表

PE文件导出表的具体结构如下表所示

|  |  |  |
| --- | --- | --- |
| 变量名称 | 大小 | 描述 |
| Characteristics | 4B | 保留值 恒定为0 |
| TimeDateStamp | 4B | 和文件头中的地址一样 |
| MajorVersion | 2B | 主版本号 |
| MinorVersion | 2B | 次版本号 |
| Name | 4B | 本PE的名称 谁导出了这些函数 |
| Base | 4B | 序号基数 |
| NumberOfFunctions | 4B | 函数导出的总数量 |
| NumberOfNames | 4B | 有名字导出的函数的数量 |
| AddressOfFunctions | 4B | 导出函数地址表 |
| AddressOfNames | 4B | 导出函数名称表 |
| AddressOfNameOrdinals | 4B | 序号表 |

在010Editor中打开查看其结构如下图所示。

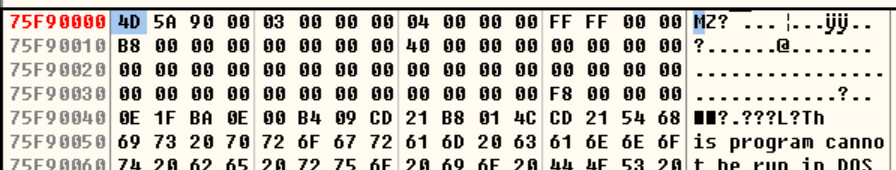


#### 1.4.8找出函数MessageBoxA的地址，并验证该地址是否正确

（1）查看实际进程内存空间中的user32地址为75F90000。



跳转到文件的75F90000位置如下图所示确实存在一个PE文件



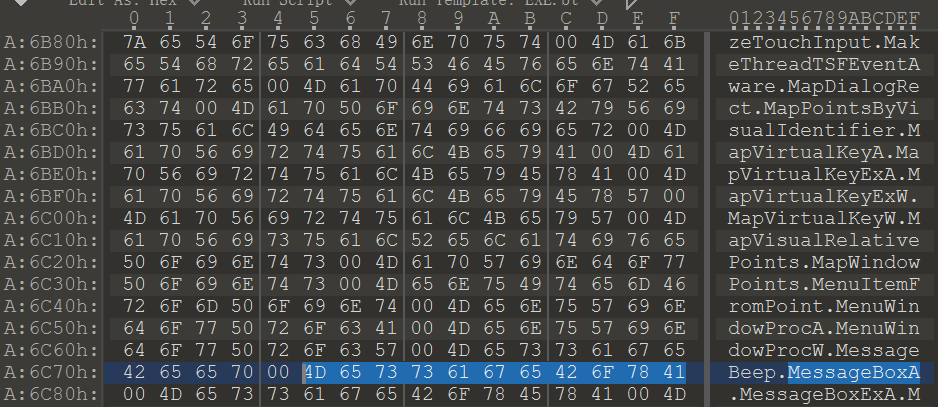
打开给出的可执行程序得到MessageboxA的调用地址为76009D50。



所以MessageBoxA的地址应为76009D50-75F90000=00079D50

下面对得到的MessageBoxA的地址进行验证。

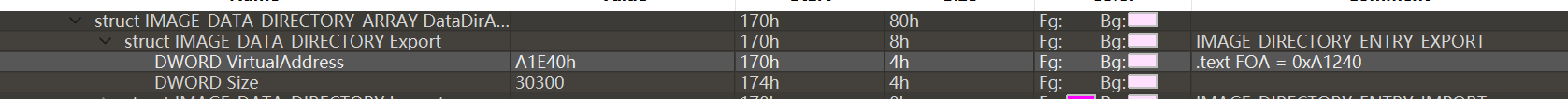
(2)找到MessageboxA在文件中的偏移地址（FOA）为A6C75H



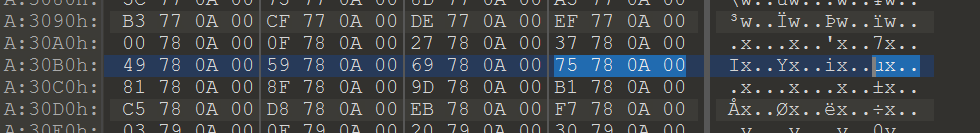
（3）计算MessageboxA函数名的相对序列地址（RVA）。

在文件中找到导出表的相对序列位置A1E40H,文件内偏移FOA=A1240

计算出MessageboxA的RVA=A6C75+(A1E40-A1240)=A7875h



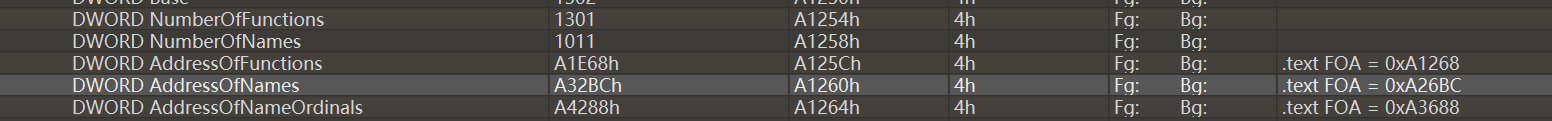
(4)找到值“75 78 0A 00”所对应的文件内偏移FOA=A30BC,



计算出其对应的相对序列地址RVA=A3CBC

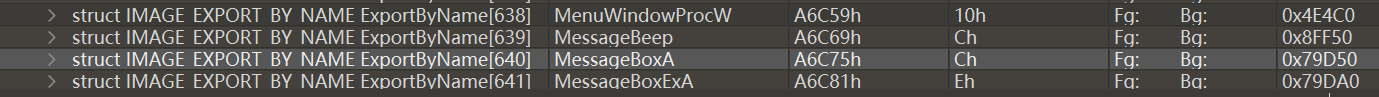
(5)找出数组序号x

在导出表中找到AddressOfNames的RVA=A32BC



求得x=(A3CBC-A32BC)/4=280h=640

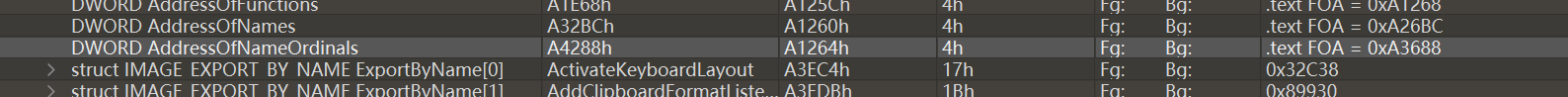
下面验证计算的正确性，在该结构中找到MessageBoxA发现其是数组的第640项，计算正确。



（6）在AddressOfNameOrdinals数组定位第x项成员得到序号y

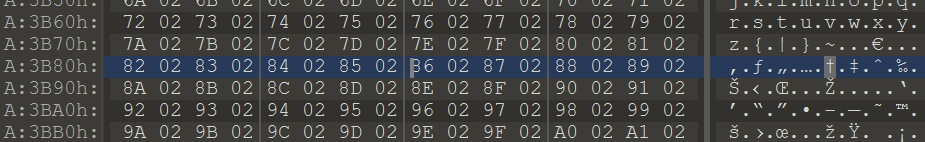
在文件中找到AddressOfNameOrdinals的值=A4288H

其在文件中的偏移FOA=A3688



其第x项成员的序号y=A3688+(280\*2)=A3B88

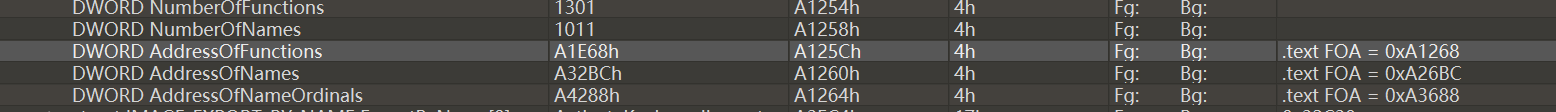
在文件中跟随到该地址后的结果如下图所示。



可得y指向的数值为0286

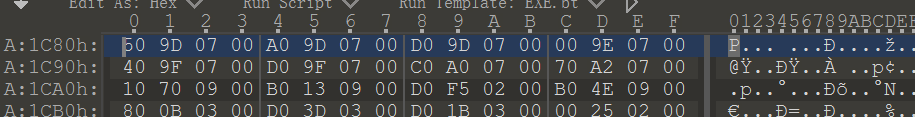
（7）在AddressOfFunctions数组中定位第y项得到函数的RVA地址。

找到AddressOfFunctions的文件内偏移FOA=A1268



其对应的第y项的文件内偏移为A1268+(0286\*4)=A1C80

在文件中跟随到A1C80位置其对应的值00079D50

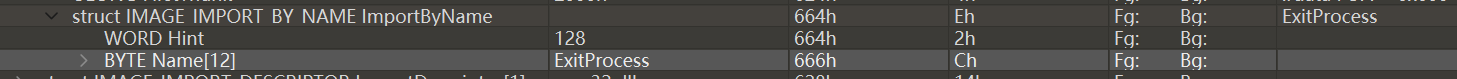


由此可得MessageboxA的地址为00079D50，与在进程的内存空间中得到的地址相同，完成验证。

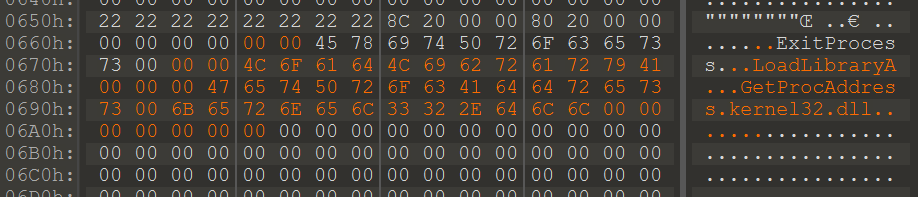
#### 1.4.9修改程序引入表不使用user32.dll

（1）修改函数导入名字表

首先找到程序的退出函数ExitProcess的Hint为128以及其名称如下图所示

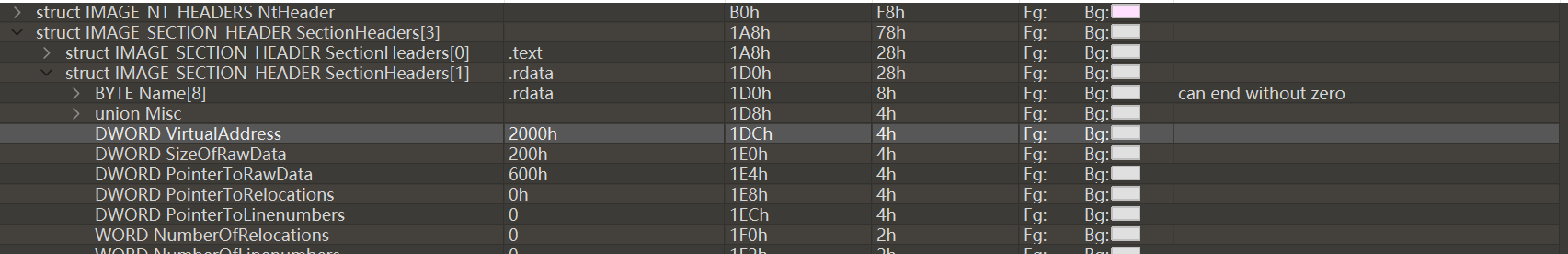


因为我们要在kernel32.dll中调用LoadLibrary和GetProcAddress两个函数，将其Hint定为00 00后加入到ExitProcess之后，在最后写入被调用的文件kernel32.dll，将之后的无关字段清零。修改完成后的结果如下图所示。

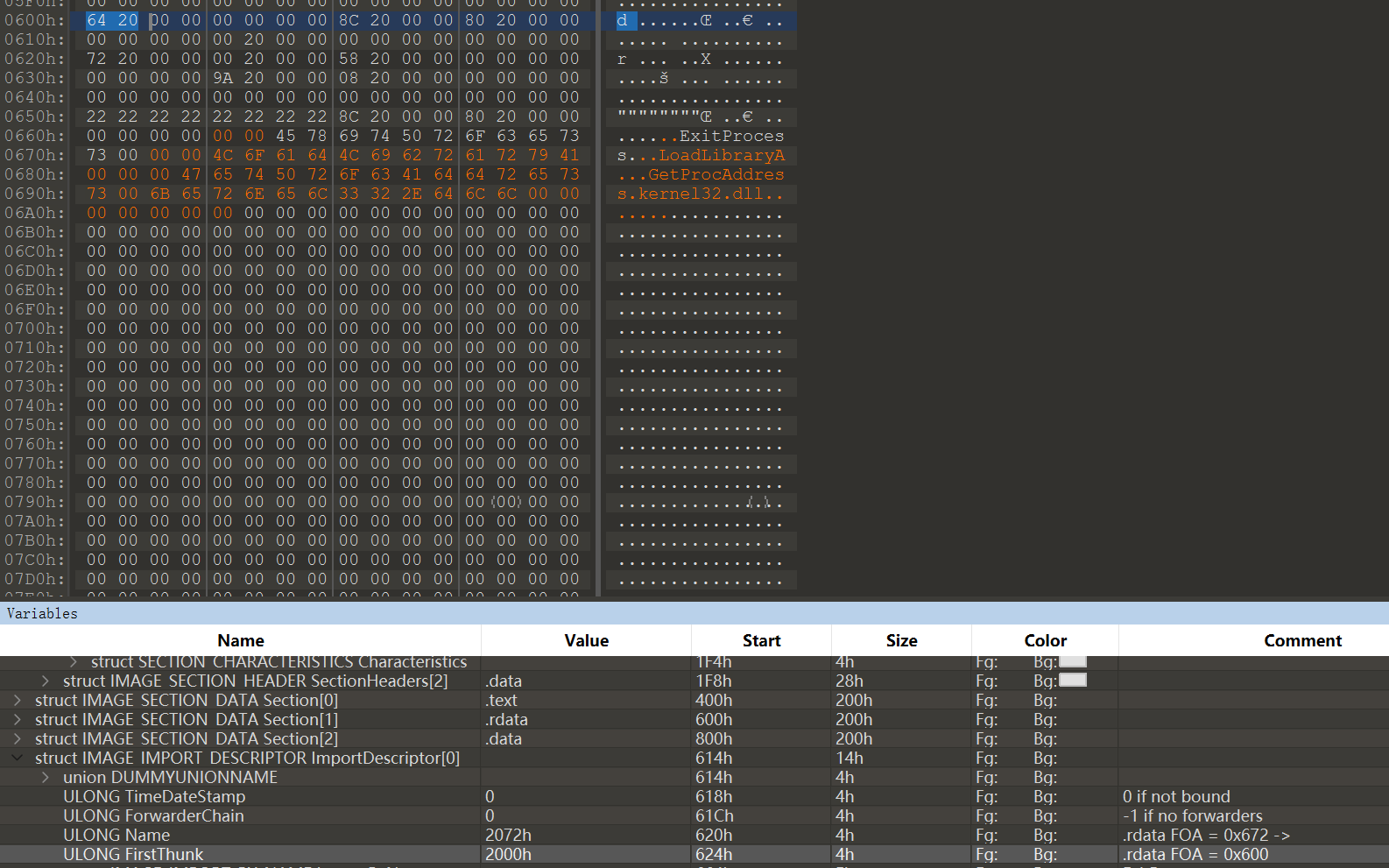


（2）计算第一个Thunk值分析Thunk值的计算原理

首先找到代码段的内容观察其RVA=2000h其文件内偏移为600h



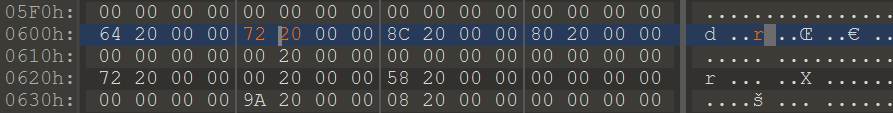
观察FirstThunk变量的值，其文件内偏移指向600h的位置，在PE文件中找到该位置观察其数据为2064。



由此可得第一个Thunk在文件中对应的偏移为2064-2000+600=664h指向我们需要调用的第一个函数ExitProcess。

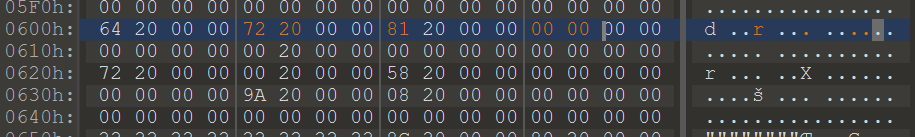
（3）计算第二个Thunk的值。

首先找到在第一步中修改过后的第二个函数LoadLibrary的文件内偏移地址为0672h，所以可以得到第二个Thunk的值应为0672-600+2000=2072，将第二个Thunk值修改为2072h，修改后的结果如下图所示。



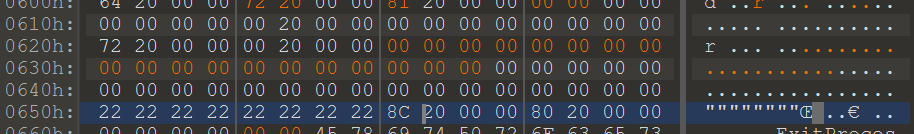
（4）计算第三个Thunk的值。

首先找到第一步中修改的第三个函数GetProcAddress的文件内偏移地址为0681h，所以可以得出第三个Thunk的值应为0681-600+2000=2081，将第三个Thunk的值修改为2081h，并且由于该函数是最后一个函数所以我们将其后面的内容清零，最终修改后的结果如下图所示。



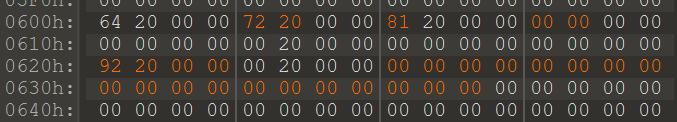
（5）清空调用user32.dll函数的调用表。

因为在修改后的程序中不再使用user32.dll中的函数，所以需要将其调用清空，修改完成后如下图所示。



（6）修改ULONG Name变量。

因为加入了两个新的调用函数在kernel32中，所以需要将其ULONG Name的变量值该为kernel32的地址即为0692h，修改后的结果如下图所示。



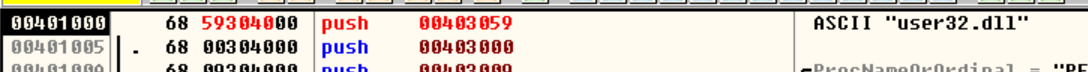
#### 1.4.10使修改后的程序恢复正常。

（1）在文件的数据字段中添加字符。

因为在调用时需要使用到字符串“user32.dll”和“MessageBoxA”，将其写入数据段中，写入后的结果如下图所示。

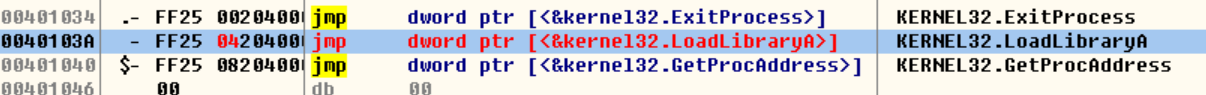


（2）将user32.dll字符串的起始地址压入栈中。



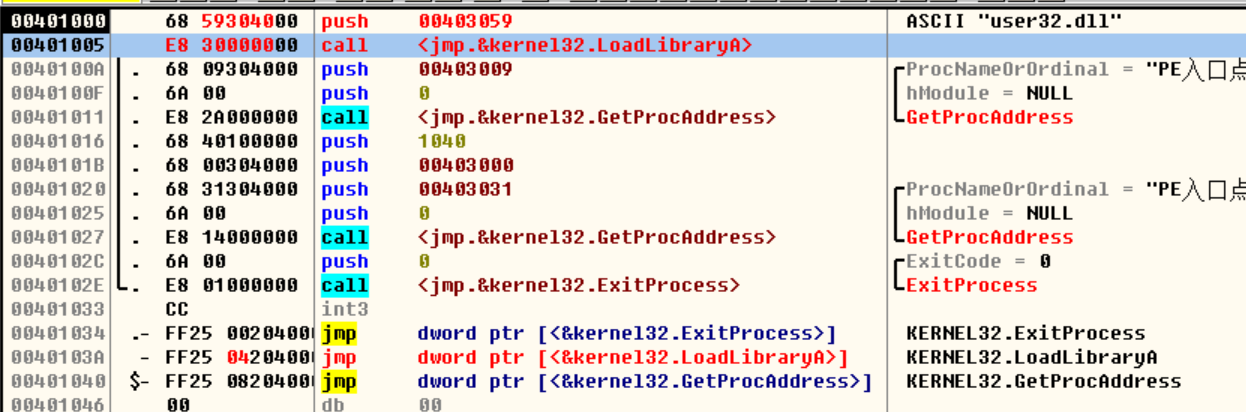
（3）修改第二个调用的地址。

因为第一个调用和第三个调用的地址分别为00和08而第二个调用的地址没有做修改对应着0C位置的内容，将第二个调用的地址改为04，使其指向Loadlibrary函数。修改后的结果如下图所示。



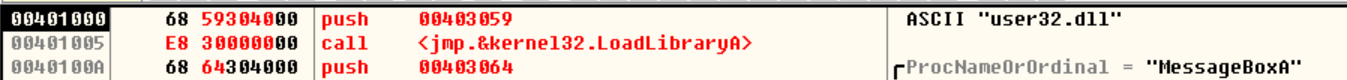
（4）调用函数LoadLibrary。

修改代码段的第二行，使用call指令调用函数LoadLibrary，具体的偏移为40103A-40100A=30h。修改后的结果如下图所示。



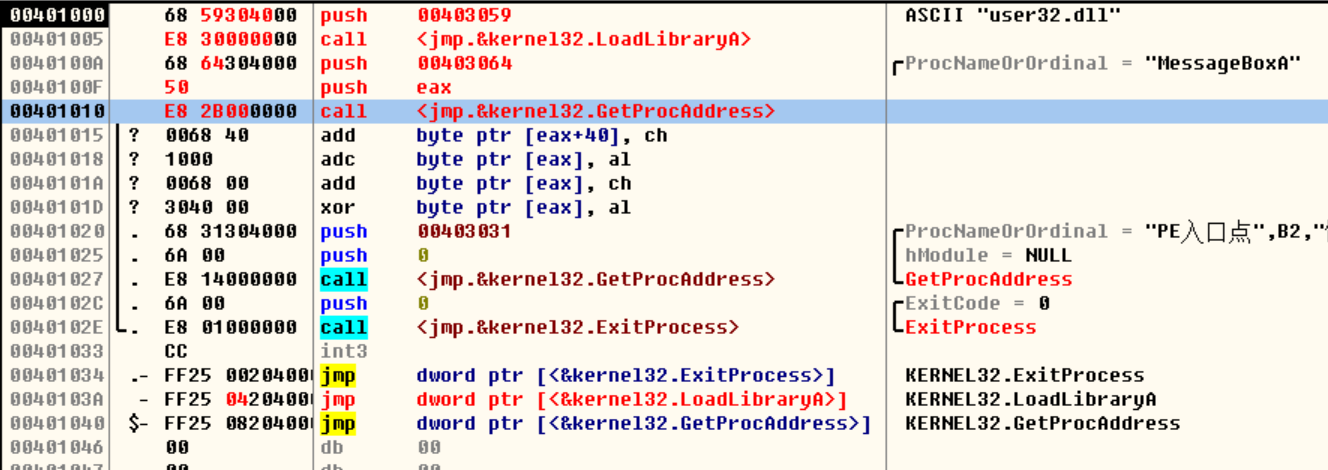
（5）将LoadLibrary函数调用的参数压入栈。

将LoadLibrary函数需要调用的字符串“MessageBoxA”压入栈中。修改后的代码如下图所示。



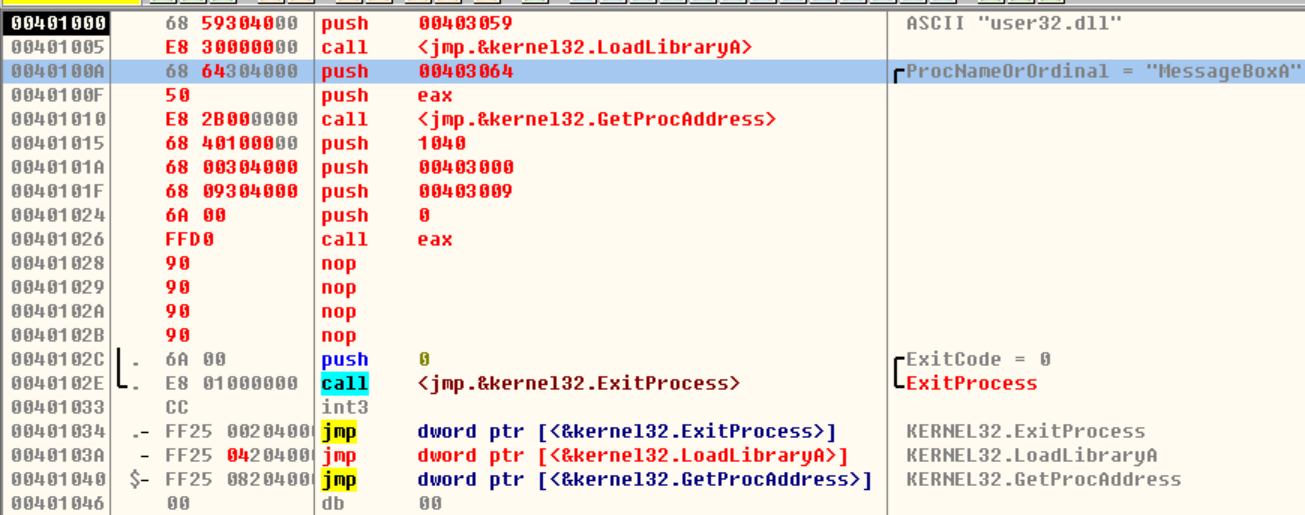
（6）调用函数GetProcAddress。

首先编写汇编指令push eax到代码段中，下一条指令的地址为401010h，Call指令的长度应该是五个字节，所以要知道call跳转的距离需要将GetProcAddress函数所在的位置401040h减去call指令的下一个指令的位置即为1040-1010=2Bh，所以CALL指令跳转的距离应该为2B，将计算好的CALL指令添加入代码段中。修改后的代码段如下图所示。



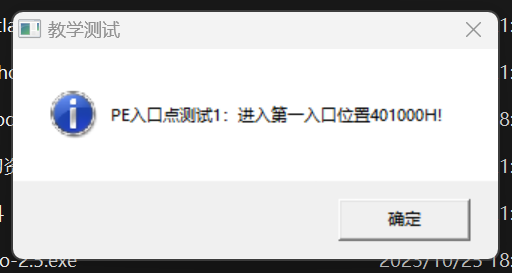
（7）对话框所需参数入栈。

将对话框的内容和标题所在的地址压入栈中。并将其之后的代码用空指令填充。修改完成后的内容如下图所示。



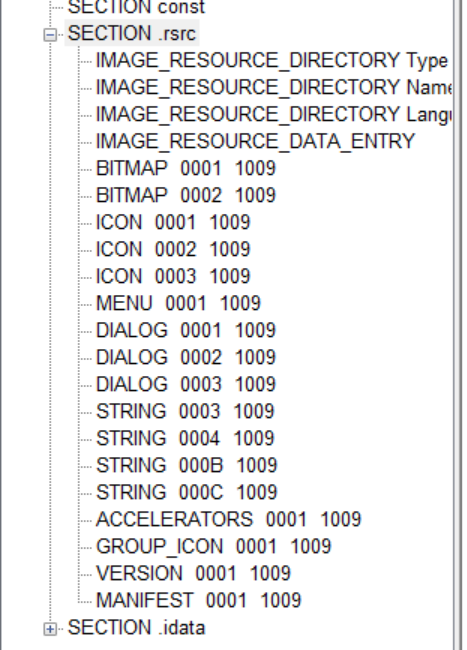
（8）运行文件

将文件保存后运行，运行结果如下图所示。可以成功运行。



#### 1.4.11分析PEview.exe程序了解资源节结构。

资源节一般放有如图标、对话框等程序要用到的资源，打开该PE文件的资源节如下图所示。



资源节的总体结构是一个树形结构，如下图所示

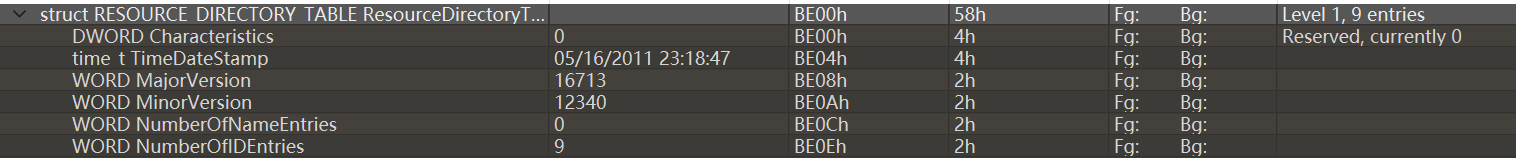


在资源节中有三个重要的数据结构，分别是

（1）目录的结构是IMAGE\_RESOURCE\_DIRECTORY。其内部的结构如下表所示。

|  |  |  |
| --- | --- | --- |
| 变量名 | 大小 | 描述 |
| Characteritics | 4B | 通常为0 |
| TimeDateStamp | 4B | 资源生成时间 |
| MajorVersion | 2B | 主版本号 |
| MinorVersion | 2B | 次版本号 |
| NumberOfNameDEntries | 2B | 以名字标识的资源数 |
| NumberOfldEntries | 2B | 以ID标识的资源数 |

本次实验所采取的PE文件使用010Editor打开后对应结构的信息如下图所示。



（2）目录项的结构是IMAGE\_RESOURCE\_DIRECTORY\_ENTRY 。

其内部结构如下表所示。其中前四个变量同处在结构体DUMMYUNIONNAME中，而后三个变量同处在DUMMYUNIONNAME2结构中。

|  |  |  |
| --- | --- | --- |
| 变量名 | 大小 | 描述 |
| NameOffset | 4B | DUMMYUNIONNAME 的低31位值. |
| NameIsString | 4B | 用于判断 NameOffset 为名称还是资源类型. |
| Name | 4B | DUMMYUNIONNAME 的值. |
| Id | 2B | 资源类型的ID. |
| OffsetToData | 4B | DUMMYUNIONNAME2 的值 |
| OffsetToDirectory | 4B | DUMMYUNIONNAME2 值的低31位值. |
| DataIsDirectory | 4B | DUMMYUNIONNAME2 值的最高位值 |

（3）IMAGE\_RESOURCE\_DATA\_ENTRY。

其内部结构如下表所示

|  |  |  |
| --- | --- | --- |
| 变量名 | 大小 | 描述 |
| OffsetToData | 4B | 资源数据的RVA |
| Size | 4B | 资源数据大小 |
| CodePage | 4B | 代码页 |
| Reserved | 4B | 保留 |

#### 1.4.12分析ICO文件的格式信息。

一个完整的ico文件的结构如下所示

目录段

目录头

idreserved //2B //保留字  
 idtype //2B //资源类型

idcount //2B //图像数量

目录索引1

目录索引2

目录索引x //x最大值为32767

图像数据段

图像1

icheader

iccolors

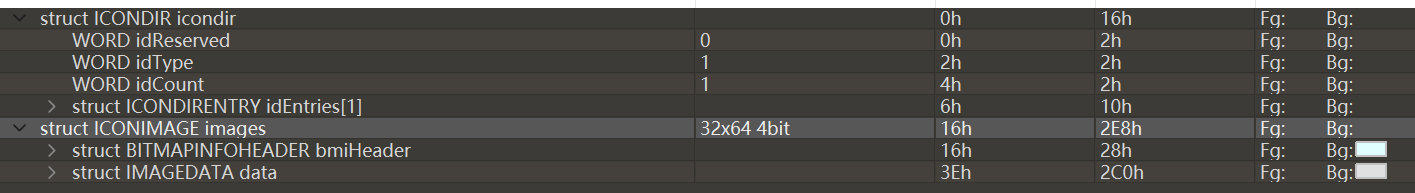
icxor

icand

图像2

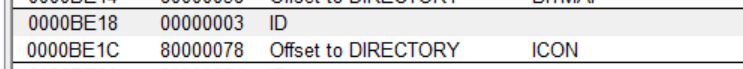
图像x //x最大为32767

本次实验采用的ico类型的文件格式打开后如下图所示。



#### 1.4.13图标替换

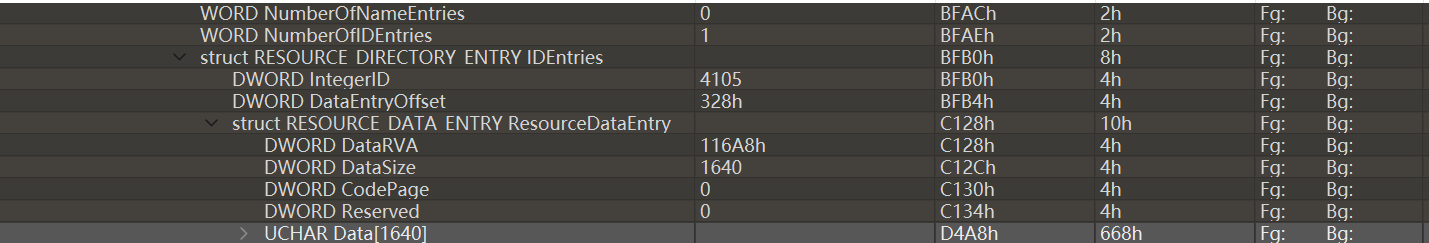
在PEview中找到PEview.exe的图标资源存放位置如下图所示为BE18



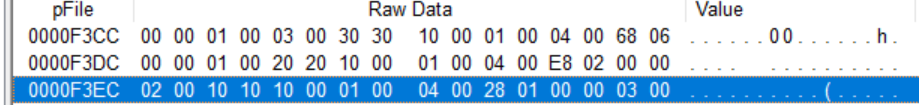
其GROUP\_ICON资源如下图所示。



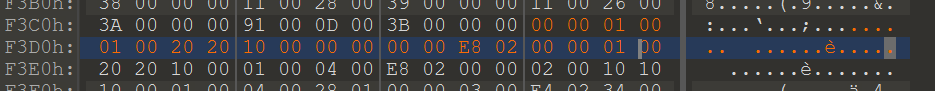
在PEview文件中找到存储图标的位置如下图所示。将cswhu.ico文件的图表数据替换到Data位置。



找到PEview文件中GROUP\_ICON结构的起始地址为F3CC，ID是E



将cswhu.ico文件中的头结构的20个字节复制到其Data结构中，修改完成后的结果如下图所示。



替换完成后查看文件发现图表被成功替换。

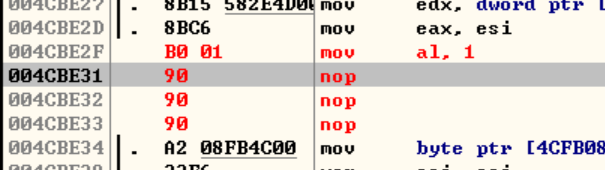


#### 1.4.14软件手工汉化

（1）exescope破解

Exescope在修改完文件之后只可以保存一次所以在进行软件汉化前先对exescope进行破解。

004CBE2F位置的代码为验证ID代码，将其al永远赋值为1即可破解，将原exe文件的对应位置的代码该为mov al,0x1如下图所示



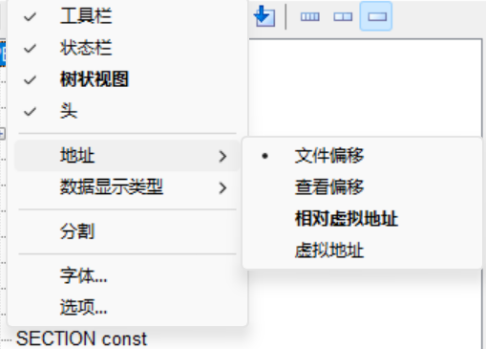
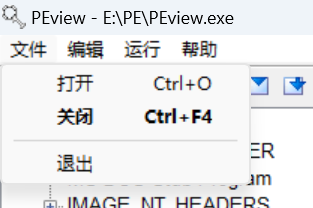
将对exescope.exe的修改保存后重新运行新的程序即可解决。

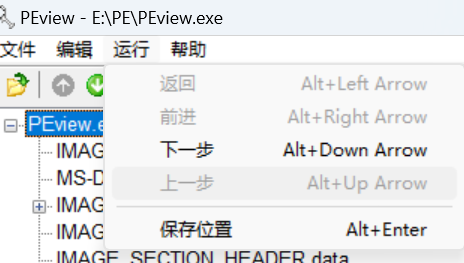
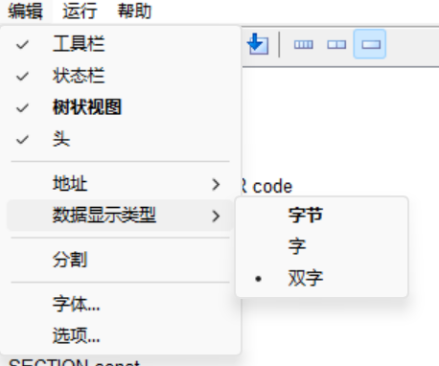
（2）PEview汉化

     在exescope中打开PEview软件后将其目录中的英文汉化为中文后的结果如下图所示。



汉化结束后保存文件，重新运行PEview程序结果如下图所示。

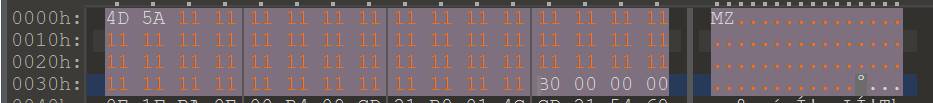




#### 1.4.15打造最小PE文件

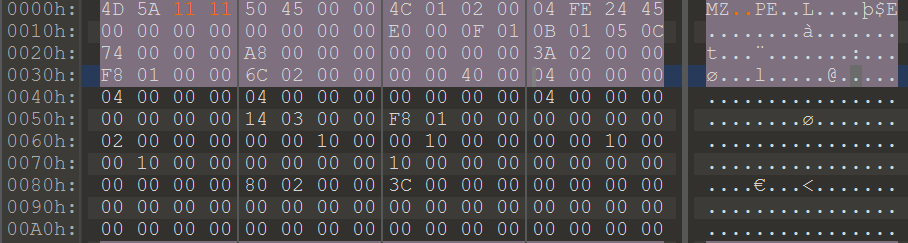
（1）删除DOS头和DOS\_STUB结构中不重要数据。

打开文件的DOS头部分将不重要的部分标记为1后如下图所示。



DOS头文件中重要的数据仅有两个，其一是DOS的签名占有两个字节，其二是DOS头的末尾003C位置的四个字节，指向NT头的文件内偏移。其余的数据都可以删去。

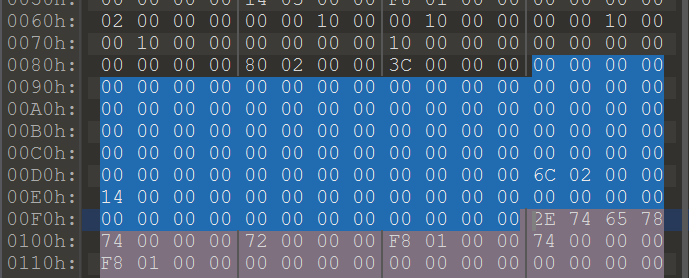
同时DOS\_STUB结构中的所有数据都是不重要的数据可以全部删除，将二者完全删除后的结果如下图所示。



保留DOS签名后的两个字节使得NT头的位置为0004H方便寻址与粒度的配合。这里003C位置的数据内容是00000004，代表SectionAlignment指定了节段在内存中的最小单位为4，指向NT文件头，正好是PE文件头的位置。

（2）修改可选文件头的数据目录

由于数据目录一共有16项，但在该文件中只有导入表会被使用所以将导入表即数组第二项之后的内容全部删除，并且将变量NumberOfRvaAndSizes的值改为00000002，代表数据目录只有两项。即将下图中选中的部分进行删除。

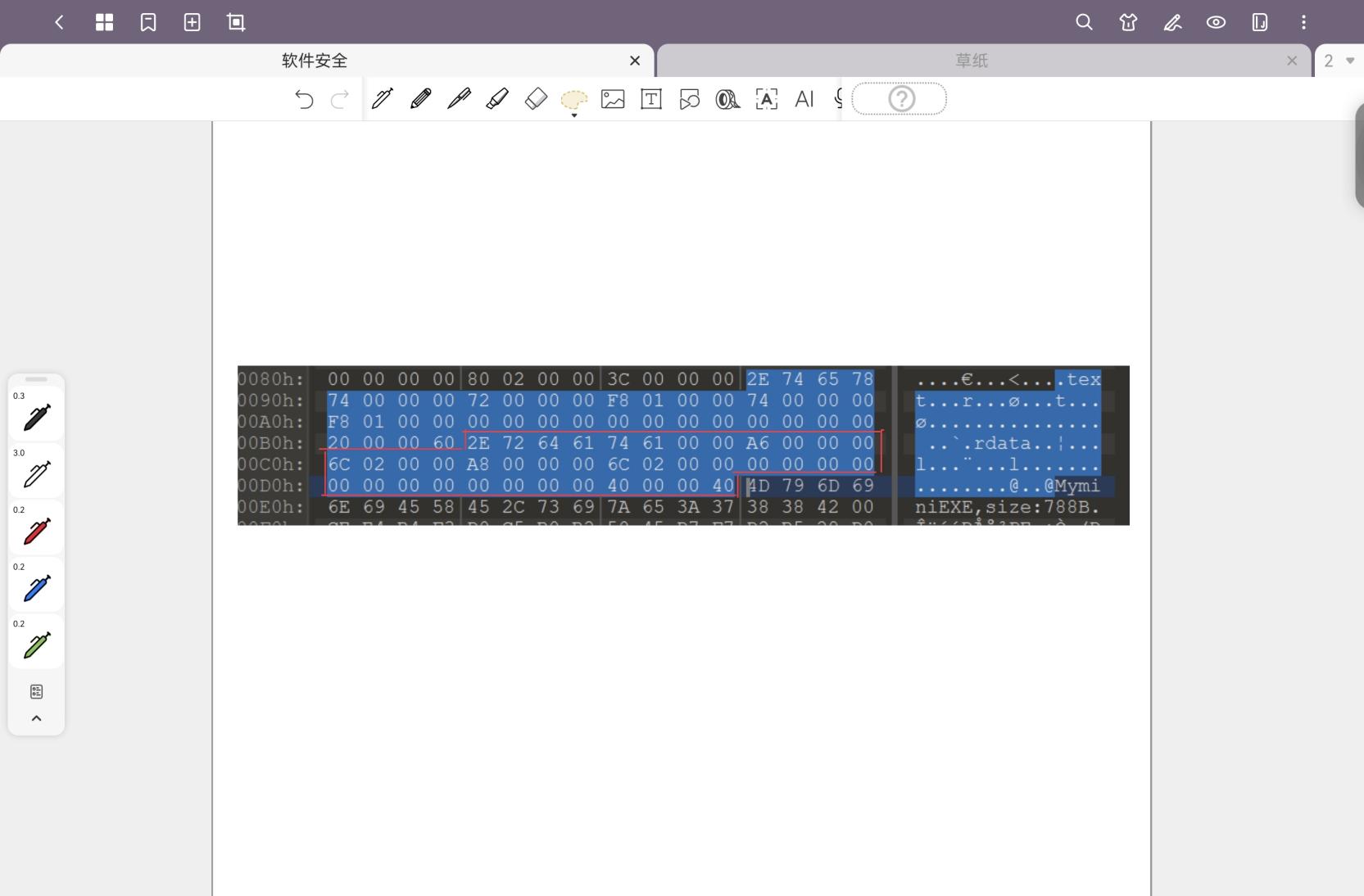


修改完成后的结果如下图所示。



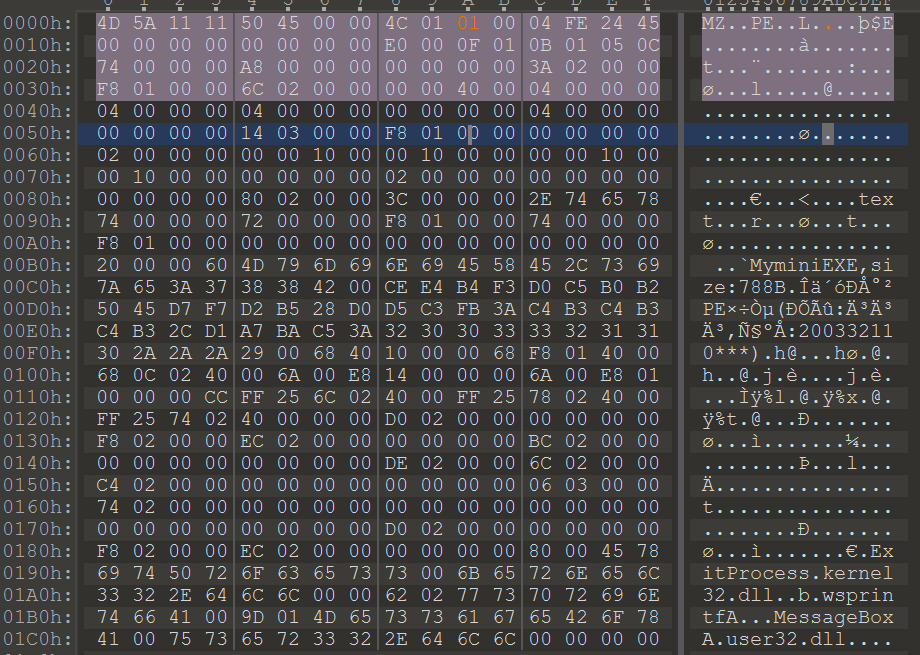
（3）修改节表内容。

接下来修改文件的节表相关的内容，打开可执行文件的节表如下图所示。

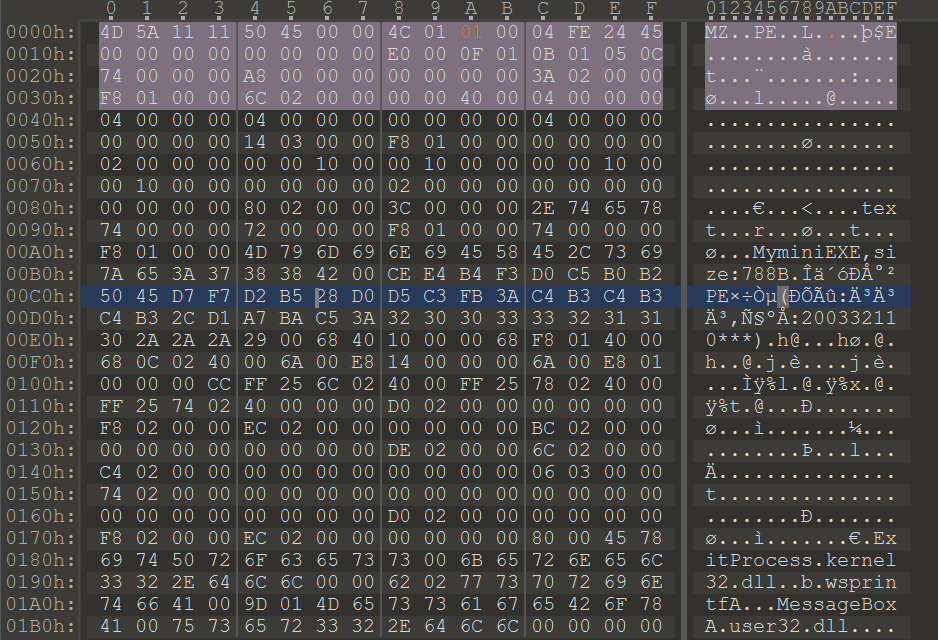


在本次实验中需要保留代码节的相关部分，将只读数据节的部分即上图中红框中的部分进行删除。

将只读数据节删除后的数据删除后的PE文件如下图所示。同时因为我们将一个节进行了删除所以需要将NT文件头中NumberOfSections变量的值改为0001h代表文件中只存在有一个代码节。修改完成后的结果如下图所示。



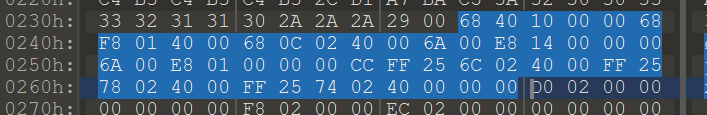
代码节中也存在有一些不重要的部分即代码节的一些属性，代码节的后16个字节。将其也删除后的结果如下图所示



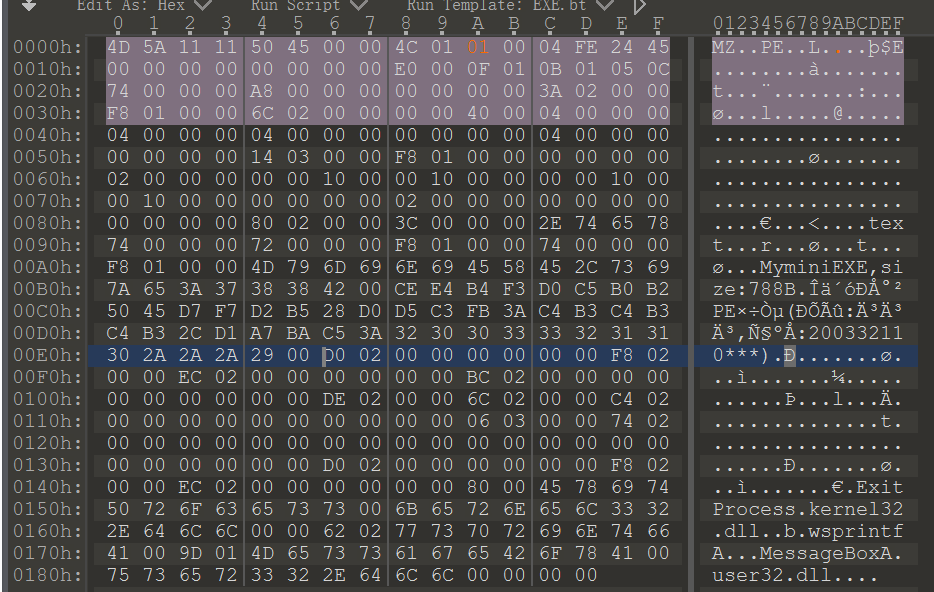
（4）修改代码节的内容。

由于需要尽可能将文件的大小压至最小，所以代码部分需要进行修改。

首先在尚未修改的PE文件中找到可选文件头中AddressOfEntryPoint变量指向的程序代码的起始部分为023A然后在原PE文件中查找目录项中的ImportAddressTable指向的位置为026C由此可以得到整个代码段的位置如下图所示。



将当前的代码段删除方便以后的步骤中重新编写新的代码段。删除后的结果如下图所示。

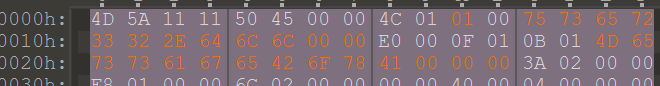


（5）修改函数调用内容。

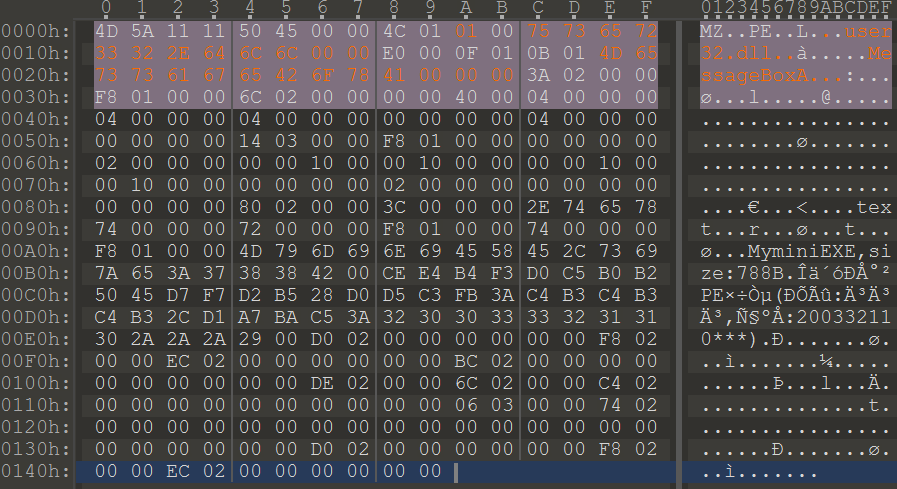
在之前的步骤中只是删掉了只读数据节的节表内容，在此要对只读数据节进行删除，首先只读数据节中保存有字符串“MessageBoxA”及其对应的dll“user32.dll”。

“user32.dll”占有12个字节，在当前PE文件中寻找一个连续的不重要的位置存放该数据，经观察比对后发现0CH-17H位置保存的数据为文件的时间戳TimeDateStamp，COFF文件符号表在文件中的偏移PointerToSymbolTable和符号表的数量NumberOfSymbols三个变量的大小恰好为12个字节，于是用“user32.dll”替换三个变量的内容。

“MessageBoxA”的长度是14个字节，在当前文件中查找14个字节的不重要的块。在可选文件头中找到五个连续的变量MajorLinkerVersion链接器的主版本号，MinorLinkerVersion链接器的次版本号，SizeOfCode代码节大小，SizeOfInitializedData已初始化数大小和SizeOfUninitializedData未初始化数大小，这五个变量对于程序的运行并不重要，所以将其替换为“MessageBoxA”将两个字符串替换后的结果如下图所示。



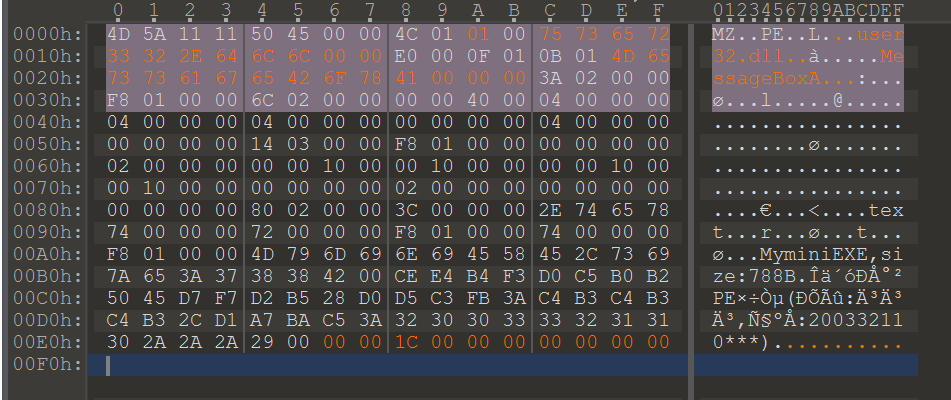
同时由于ExitProcess在函数运行过程中并不是必须的函数调用，所以在当前PE文件中将其调用的部分删除。删除后的整个PE文件如下图所示。



（6）只读数据段重构。

首先我们将原本的只读数据段进行删除。因为文件中的对其粒度为4，所以需要在文件的末尾填充两个字符保证下一部分的起始地址是4的倍数保证对齐。

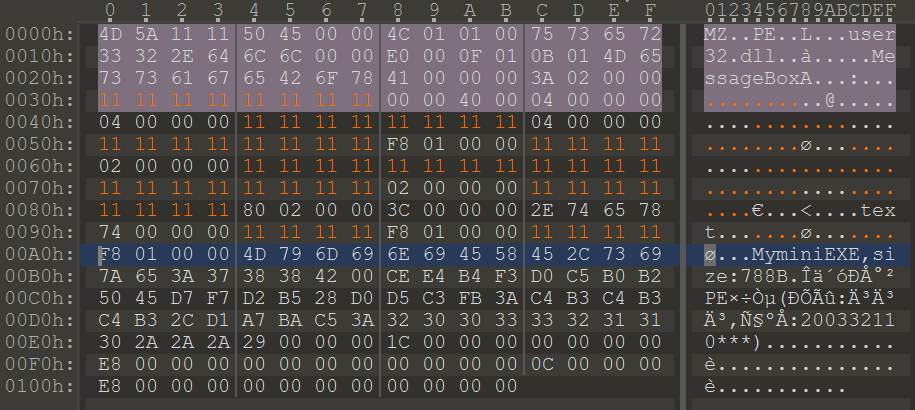
对于IAT表我们需要让其指向我们从user32.dll中调用的函数“MessageBoxA”因此我们将其设置为1C 00 00 00 00 00 00 00h。重构后的PE文件如下图所示。



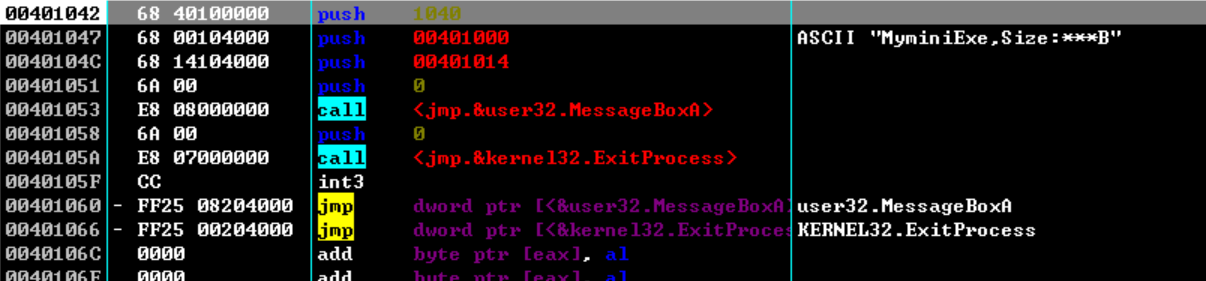
对于IDT表，我们需要让其OriginalFirstThunk变量和FirstThunk变量指向IAT的位置，即为E8 00 00 00h。将变量ForwarderChain变量指向第一个API传递器链表）即为0C 00 00 00H。最终将PE文件修改结果如下图所示。

（7）补充函数代码。

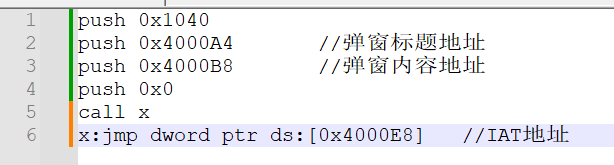
由于我们在第四步中将程序的代码删除，所以在此需要重新写程序的代码部分。首先我们先把当前PE文件中能够放置数据的不重要的位置标出来如下图所示。



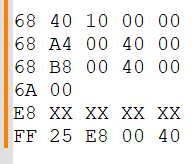
将原PE文件使用OllyICE打开后如下图所示。



仿照原PE文件的汇编指令编写一段汇编指令如下图所示。



将汇编指令逐个翻译成机器码后写入到当前PE文件中，并根据其在PE文件中的位置调整jmp和call函数的参数。翻译成的机器码如下图所示



在将机器码填入PE文件的过程中，由于代码位置不连续所以应该在代码断开的位置补上jmp指令进行跳转。将机器码填入后的PE文件如下图所示。



其中call指令的操作码为7CH-71H=0BH

（8）修改参数

首先修改变量SizeOfOptionalHeader整个可选NT头的大小，在第2步中删除了14个目录项即14\*8=112个字节，所以该变量的值应该改为E0-70=70H。修改后的结果如下图所示。



然后修改变量AddressOfEntryPoint的值，将其指向程序代码的起始位置。



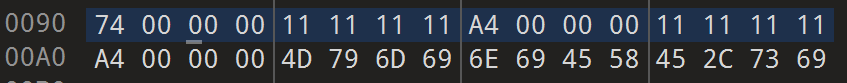
然后修改变量SizeOfHeaders指向第一个节开始的位置，在这里让其指向程序开始地址。



然后是修改导入表的RVA和Size，RVA指向F0位置，大小为28个字节。

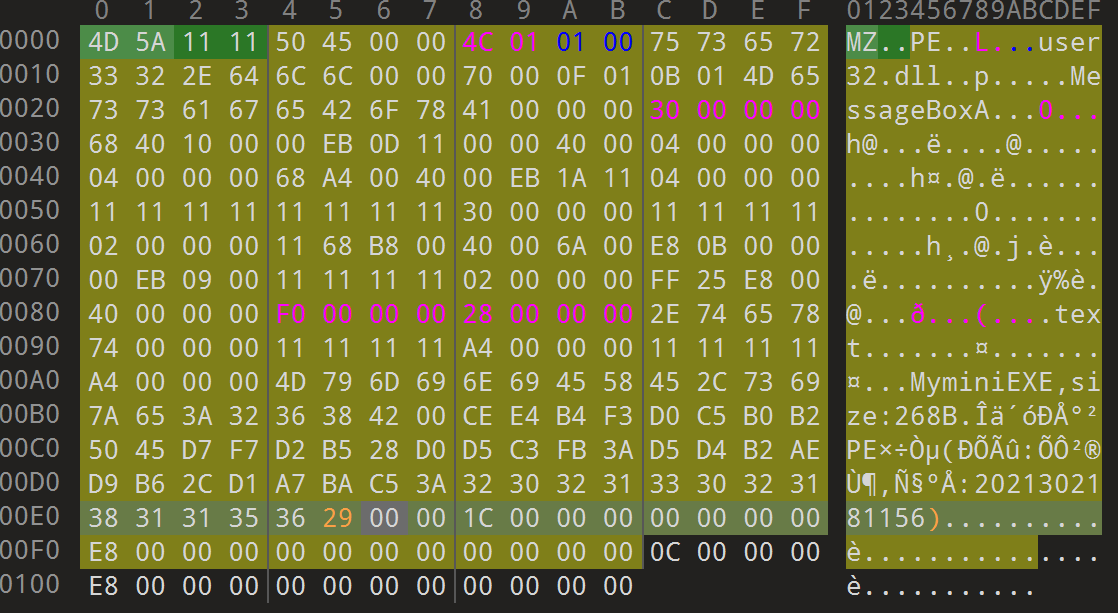


最后修改节表的VirtualAddress和PointerToRawData两个变量的值使其指向A4H位置。

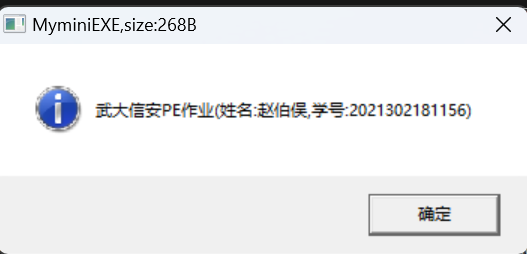


（9）结果展示

最终修改后的PE文件如下图所示



执行该PE文件运行结果如下图所示



（10）本文件的最小极限可能是多少

本文件的最小极限应该为192个字节。

因为在已知最小的PE文件为97个字节，但是在本文件中的代码指令有29个字节，同时本文件需要有保存弹窗标题和内容的位置占用66个字节，所以本文件的最小极限应该为97+66+29=192个字节。

#### 1.4.16如何编码实现PE程序中对应资源的提取与替换？涉及到哪些关键API函数。

（1）实现PE程序中对应资源的提取与替换步骤

1. 打开可执行文件或动态链接库：使用操作系统提供的函数（如CreateFile）打开目标文件，获取文件的句柄。
2. 定位图标资源表：PE文件的结构中包含了图标资源表的位置信息。可以通过解析PE文件头和节表来定位到图标资源表的位置。
3. 解析图标资源表：图标资源表是一个数据结构，它包含了多个图标资源项。每个图标资源项都有相应的偏移和大小信息。需要解析这个表，以获取每个图标的偏移和大小。
4. 提取图标资源：使用文件句柄和图标资源项的偏移，可以将图标资源从文件中读取到内存中。可以创建一个新的文件来存储提取出的图标资源。
5. 替换图标资源：如果想替换图标资源，首先需要准备好想要替换的新图标文件。然后，可以使用文件句柄和图标资源项的偏移，将新的图标资源写入到文件中。
6. 更新文件头：如果替换了图标资源，还需要更新PE文件头中的相应字段，以确保文件的正确性。

（2）涉及到的关键API函数

BeginUpdateResource：此函数开始修改资源。它打开指定的可执行文件以进行资源修改，并返回一个句柄，该句柄可用于后续资源操作。

UpdateResource：此函数将新的资源添加到资源表中或替换现有的资源。它使用先前获得的句柄和资源类型/名称/语言标识符来标识要操作的资源，并提供资源数据进行替换。

EndUpdateResource：此函数完成资源的修改并保存更改。它将之前打开的可执行文件句柄作为参数，以及一个布尔值，指示是否应将资源更新写入文件或仅保留在内存中。

#### 1.4.17当目标程序的图标资源为多个时，每个图标资源分别对应着哪里？此时图标替换策略应该如何调整？

图标资源的对应关系：

（1）应用程序图标: 这是程序的主要图标，用于在操作系统的文件资源管理器、任务栏、快捷方式等位置显示。

（2）窗口图标: 当程序打开一个窗口时，窗口的标题栏、任务栏按钮等位置通常会显示一个窗口图标。

（3）文件关联图标: 程序可能会关联特定类型的文件，为这些文件类型设置一个关联图标，用于在文件资源管理器中显示该文件类型的图标。

（4）程序状态图标: 对于某些程序，不同的图标可能表示不同的程序状态，例如活动、停止、错误等。这些图标可以用于在用户界面中传达程序的当前状态。

能够采取的策略可以是

（1）替换特定索引的图标资源：根据资源表中的索引，确定要替换的特定图标资源，并将其替换为新的图标文件。这样可以精确地替换目标程序中的特定图标。

（2）替换所有图标资源：遍历资源表中的所有图标资源，并将它们全部替换为新的图标文件。这样可以将目标程序中的所有图标资源都更新为相同的新图标。

（3）添加新的图标资源：在资源表中添加一个新的图标资源项，并将新的图标文件与该项关联。这样可以在目标程序中添加一个额外的图标资源，而不是替换现有的图标资源。

#### 1.4.18资源节与恶意代码有何关联？

资源节是PE（Portable Executable）文件格式的一部分，用于存储程序所需的资源，例如图标、位图、字符串、对话框等。这些资源可以通过资源节在运行时被程序动态加载和使用。资源节通常由资源目录表和资源数据组成。

恶意代码可以利用资源节来实现一些特定的目标，例如：

(1)隐藏恶意代码：攻击者可以将恶意代码嵌入资源节中，以混淆或隐藏其存在。资源节通常不被反病毒软件或分析工具所关注，因此攻击者可能试图通过将恶意代码放置在资源节中来避免被检测或分析。

(2)传播恶意内容：资源节可以包含恶意的图像、文本或其他文件类型，这些文件可能会误导用户或触发其他安全漏洞。例如，一个看似无害的图标文件可能实际上包含恶意脚本或代码，当用户尝试打开或预览该图标文件时，恶意代码可能被执行。

(3)加载外部资源：恶意代码可以使用资源节作为一个容器来存储加密的或编码的外部资源，例如恶意下载器。恶意代码可以解析资源节中的数据，并在运行时解密或解码这些资源，从而获取进一步的恶意功能或指令。

尽管资源节本身不包含可执行代码，但它可以被恶意代码滥用以实现隐藏、传播或加载其他恶意功能。因此，在进行恶意代码分析或病毒检测时，资源节也是一个需要关注的部分。

#### 1.4.19什么是HOOK？其与本章学习有何关系？

对于Windows系统，它是建立在事件驱动机制上的，整个系统都是通过消息传递实现的。hook（钩子）是一种特殊的消息处理机制，它可以监视系统或者进程中的各种事件消息，截获发往目标窗口的消息并进行处理。所以说，我们可以在系统中自定义钩子，用来监视系统中特定事件的发生，完成特定功能，如屏幕取词，监视日志，截获键盘、鼠标输入等等。  
        钩子的种类很多，每种钩子可以截获相应的消息，如键盘钩子可以截获键盘消息，外壳钩子可以截取、启动和关闭应用程序的消息等。钩子可以分为线程钩子和系统钩子，线程钩子可以监视指定线程的事件消息，系统钩子监视系统中的所有线程的事件消息。因为系统钩子会影响系统中所有的应用程序，所以钩子函数必须放在独立的动态链接库(DLL) 中。  
       所以说，hook（钩子）就是一个Windows消息的拦截机制，可以拦截单个进程的消息(线程钩子)，也可以拦截所有进程的消息(系统钩子)，也可以对拦截的消息进行自定义的处理。Windows消息带了一些程序有用的信息，比如Mouse类信息，就带有鼠标所在窗体句柄、鼠标位置等信息，拦截了这些消息，就可以做出例如金山词霸一类的屏幕取词功能。

与本章的学习的关系：

（1）钩子要求必须放在一个独立的动态链接库中，在本章的实验中将PE程序对user32.dll动态库的调用删除，改用同一个动态链接库kernel32.dll能够满足钩子的要求。

（2）在了解函数导入的基本原理和修改hello-2.5.exe程序以弹出第三个对话框时，HOOK技术可以用于拦截并修改特定函数的调用。例如，可以使用钩子技术拦截MessageBoxA函数的调用，使其显示自定义的标题和内容。  
 （3）在修改hello-2.5.exe程序的引入表，仅从kernel32.dll中引入LoadLibrary和GetProcAddress函数时，HOOK技术可以用于拦截这些函数的调用，并在其执行前后执行自定义的代码。这样可以实现在hello-2.5.exe程序中添加额外的功能或修改原有功能的行为。  
 （4）在利用PE编辑工具修改PEview.exe程序，使其图标变成csWhu.ico时，HOOK技术可以用于拦截对LoadIcon函数的调用，并将其重定向到自定义的图标资源。这样可以实现对程序图标的替换。

### 1.5实验体会和拓展思考

（1）在最小PE的改写过程中会触发Windows的安全防护将整个PE文件识别为病毒，可能在修改过程中有某些字符串相互拼接形成了病毒的标识。

（2）在使用工具的过程中可能会出现某个软件会弹出未注册不可使用的弹窗，可以使用实验中学到的OllyDbg将软件的可执行文件打开修改后进行使用。

（3）虽然只是一个很小的可执行文件，但是其内部却有着非常多的格式与关系，且各种变量之间的关系错综复杂。

（4）每一个可执行文件都因为文件字节对齐等原因会有很多的冗余位，删除掉冗余位后程序仍然能够正常工作。但是如果直接将冗余位删除有会破坏文件中地址相关变量的指示关系使得程序无法运行。