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)：

学 号：

姓 名：

课程名称：

任课教师： 赵磊

2023年 10 月 25 日

# 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

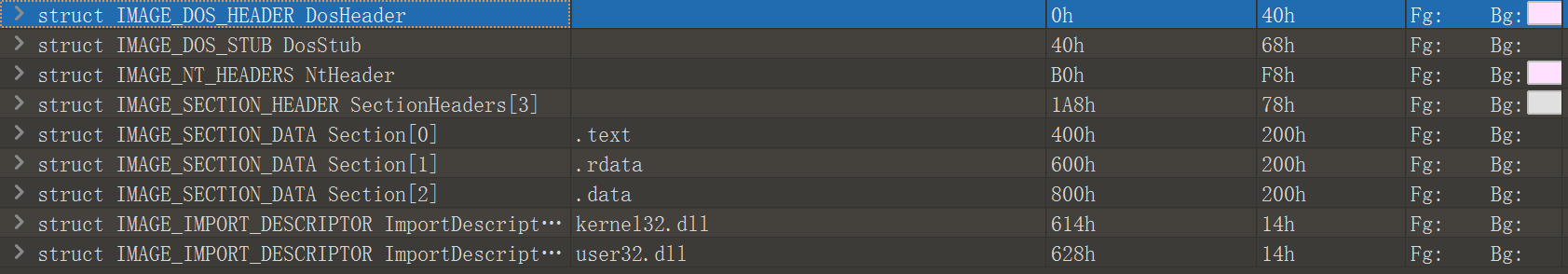
**课后习题思考：**

* 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 PE文件格式初步分析与调试

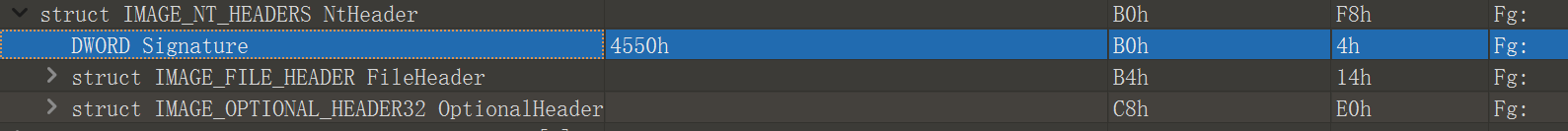
首先使用010Editor打开hello-2.5.exe



编辑软件自带的模板已将我们的文件划分为上述几个部分，对应下图的结构

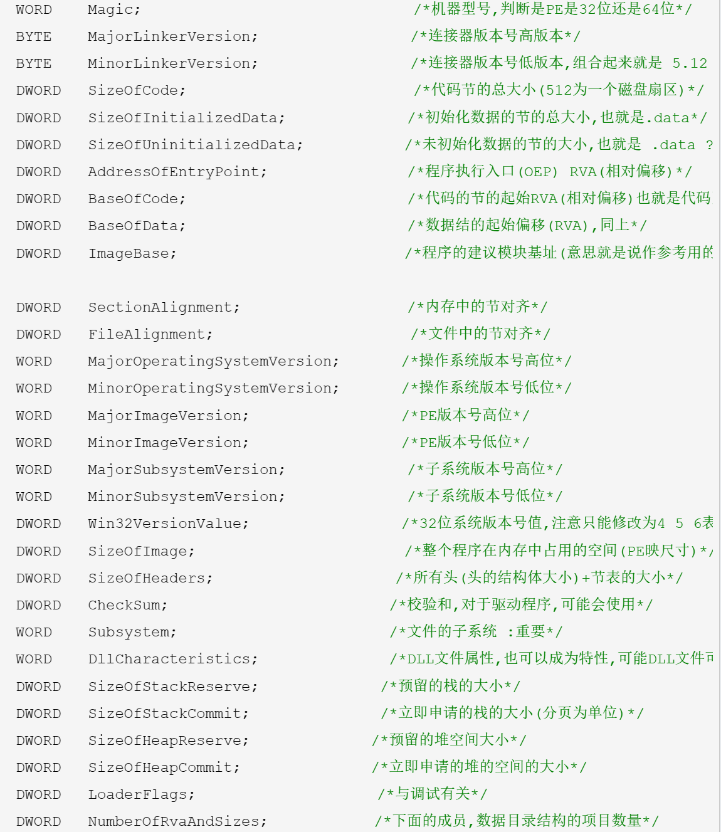
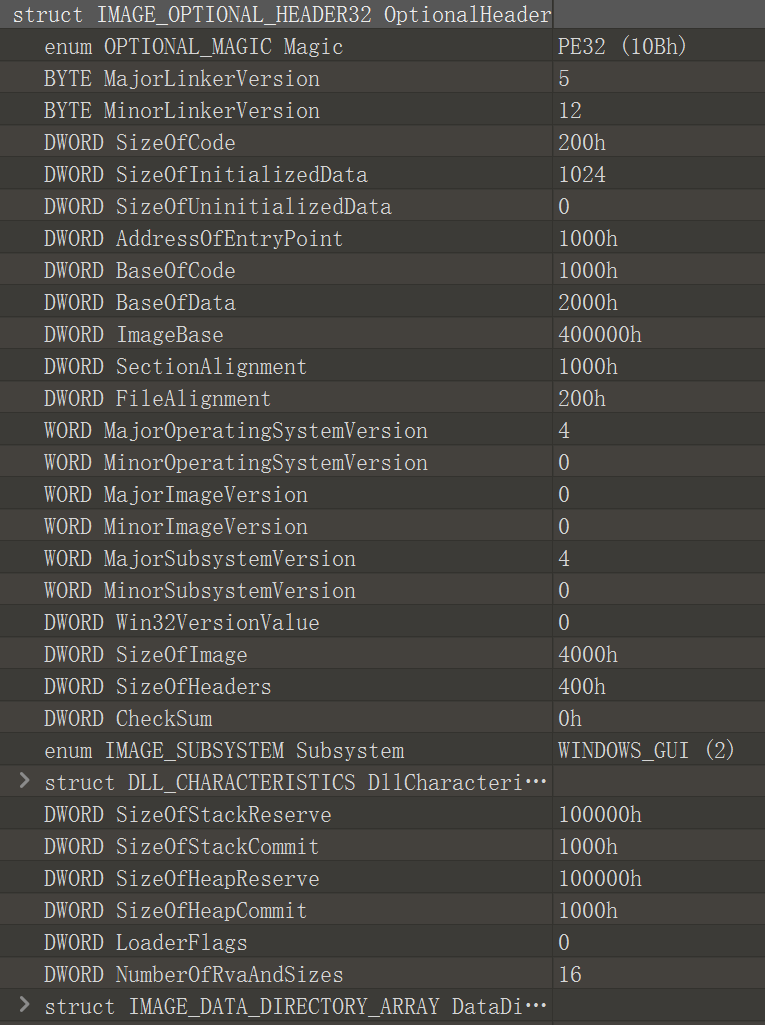


我们详细查看一下NT文件头

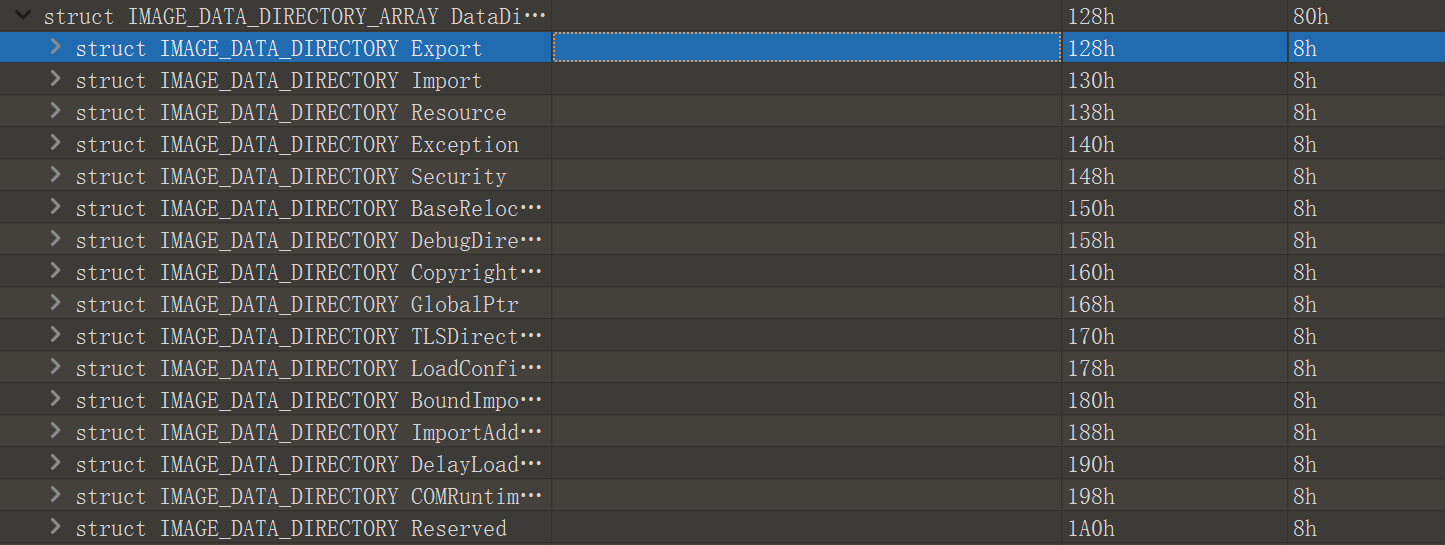


包括PE字段、映像文件头和可选文件头

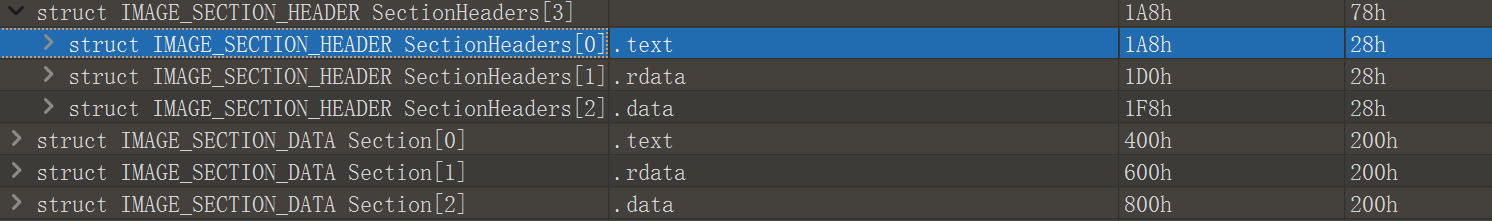
其中，可选文件头包含很多信息



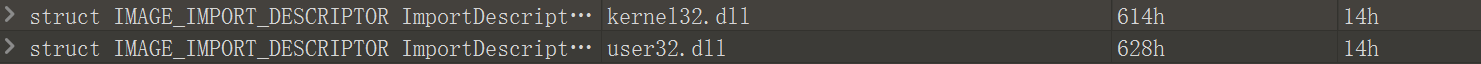
可选文件头的最后是16个数据目录表



下面是节表头和对应的三个节表：

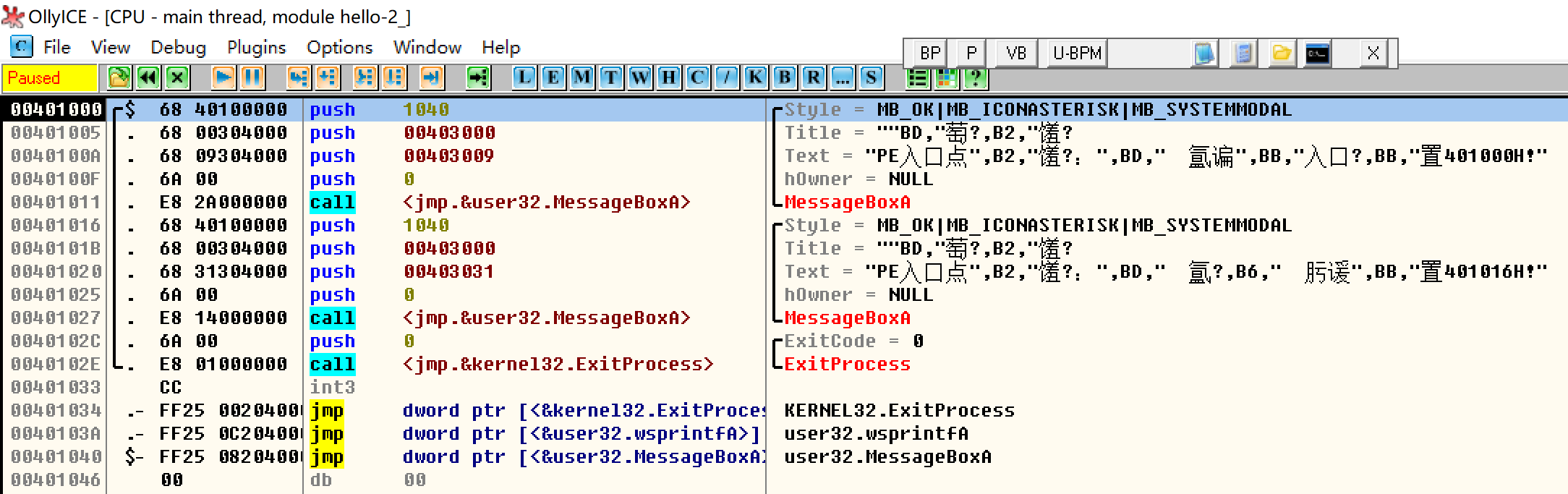


最后是引入函数表：

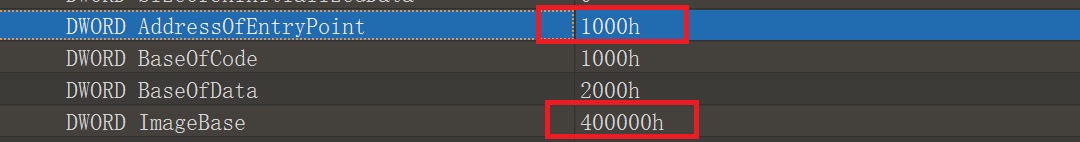


本例中引入了kernel32和user32

我们再使用OllyDBG打开文件

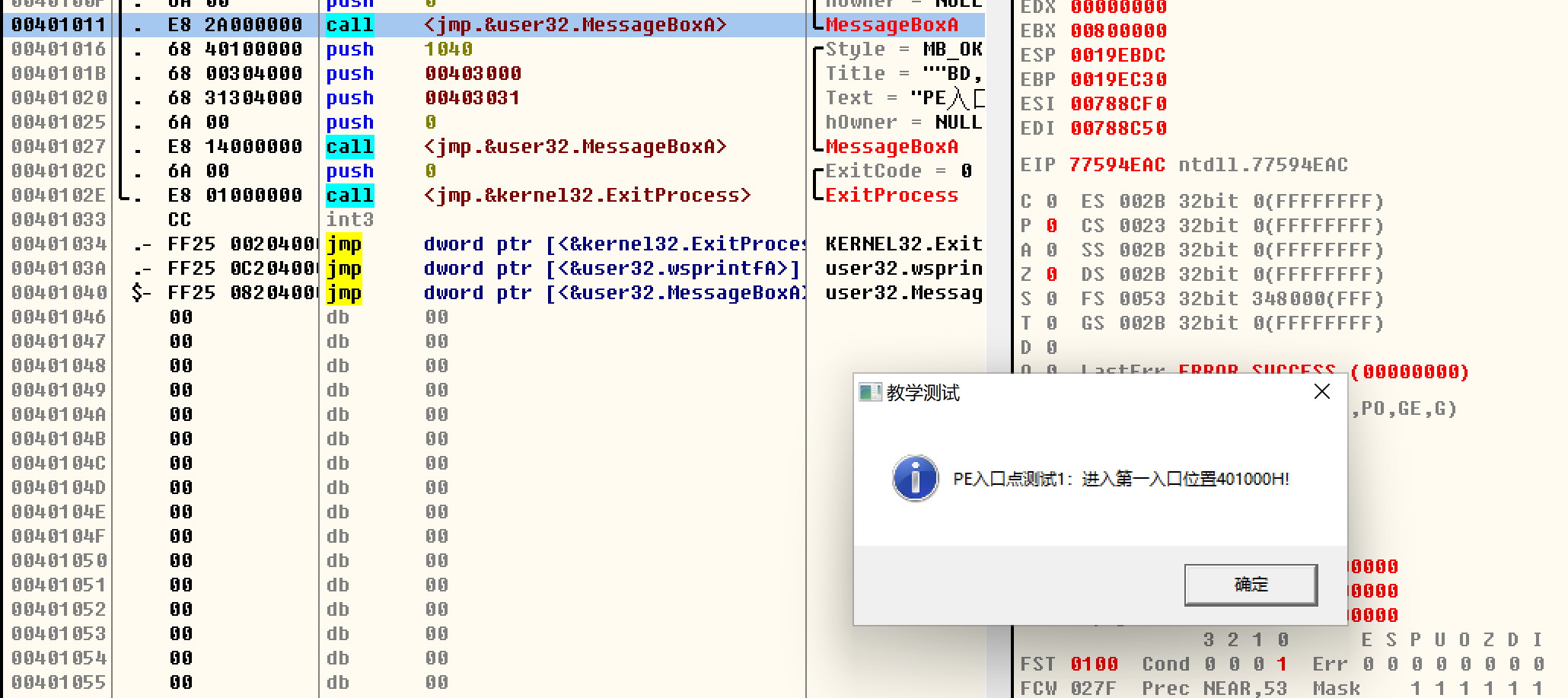


我们发现加载后，程序断点停在程序入口点，地址401000

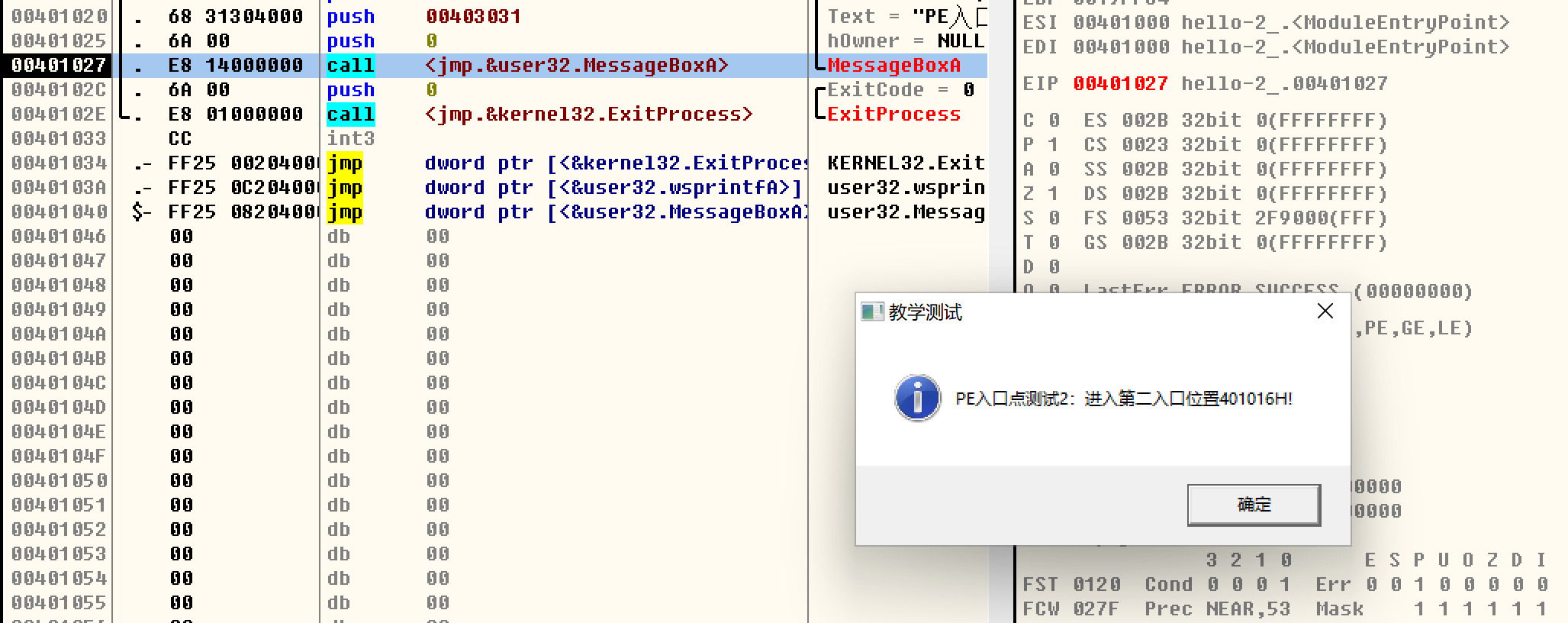


这与我们在010中看到的完全一致，基地址+偏移正好等于401000

我们对程序进行单步调试，发现当进入到如图所示语句时，会弹出一个框



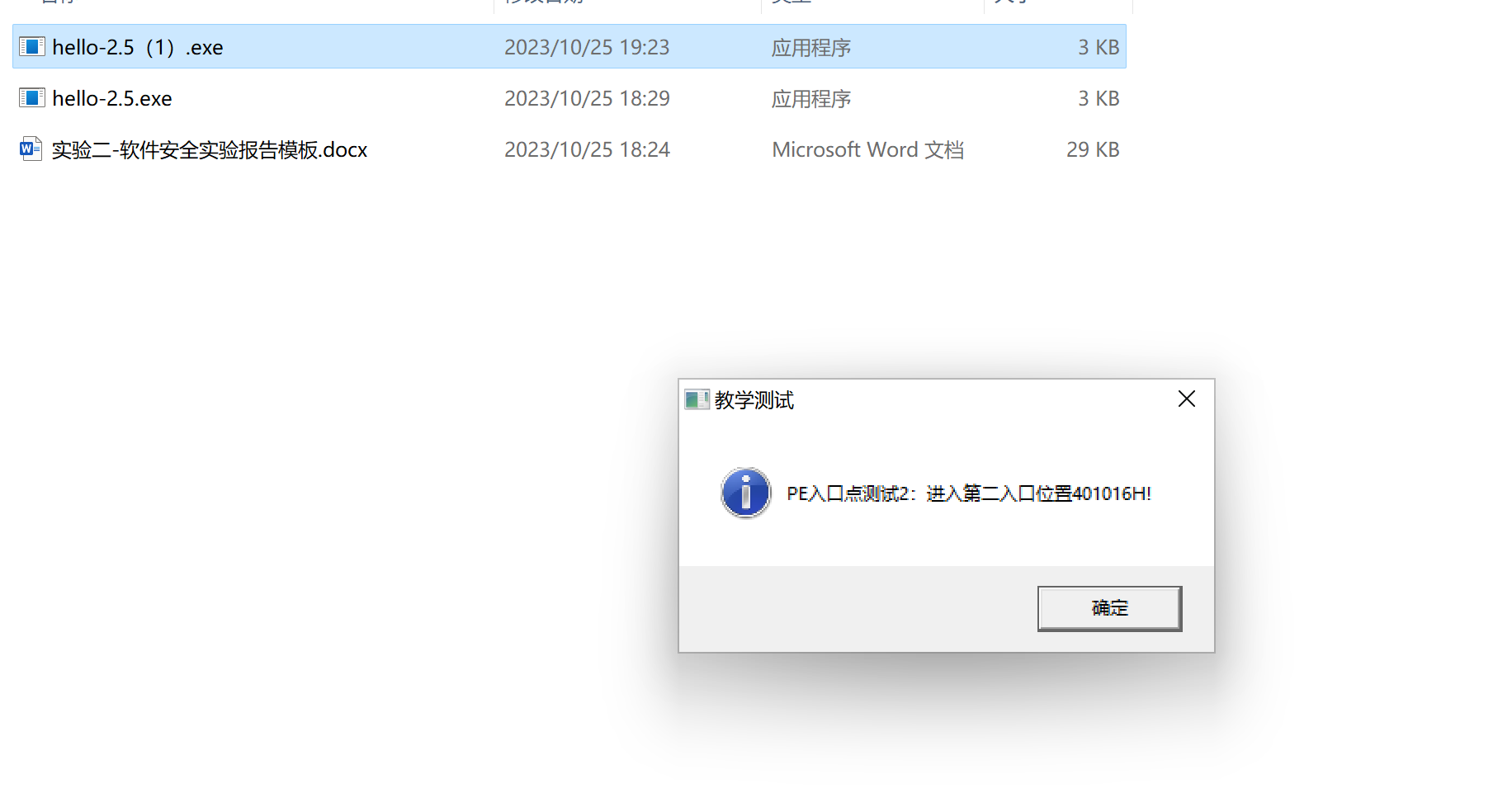
走到下一个call指令弹出第二个窗口



因此我们想修改程序使其只弹出第二个框，思路就是修改程序入口点的地址，即从原来的401000开始变为401016开始。



将其另存为一个新的PE文件hello-2.5(1).exe



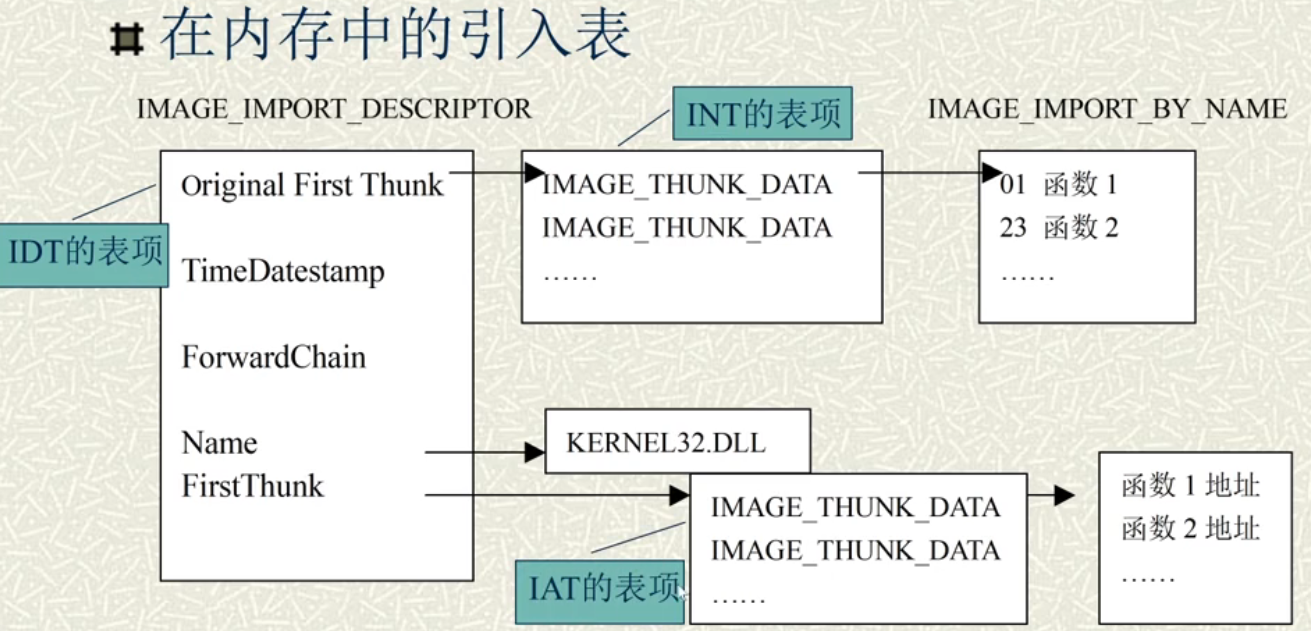
运行，可以看到它只弹出第二个对话框

### 1.4.2 函数的引入与引出机制

1.使用StudyPE查看PE文件头部



下图是函数引入表的结构



上图涉及了四个关键部分：

1. IAT（Import Address Table）：IAT是一个存储被导入函数地址的表。程序在运行时会通过IAT来访问被导入函数，其中的地址是在程序加载时由操作系统填充的。

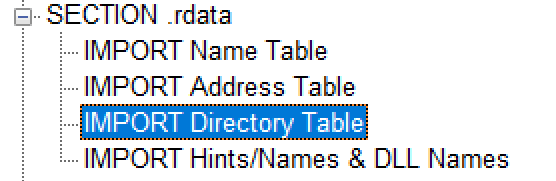
2. IDT（Import Descriptor Table）：IDT包含了被导入函数的描述信息，包括函数的名称、序号以及在DLL文件中的位置等。程序可以根据IDT中的信息来定位并访问被导入函数。

3. INT（Import Name Table）：INT存储了被导入函数的名称。当程序需要调用某个被导入函数时，首先根据IDT获取函数的序号，然后通过INT获取函数的名称。

4. DLL文件名：DLL文件名是指包含被调用函数的动态链接库（DLL）的名称。程序需要通过DLL文件名来定位和加载相应的DLL文件，以便在其中找到被调用的函数。

整体流程是在程序运行时，根据这些表的组合，程序可以根据提供的函数IDT表项位置，读取IAT表项、INT表项和DLL文件名。随后，程序可以根据获取的信息来定位和调用所需的函数，实现相应的功能扩展和共享。

在PE文件中查看rdata段



最重要的IDT的位置，在可选文件头的数据目录表中确定

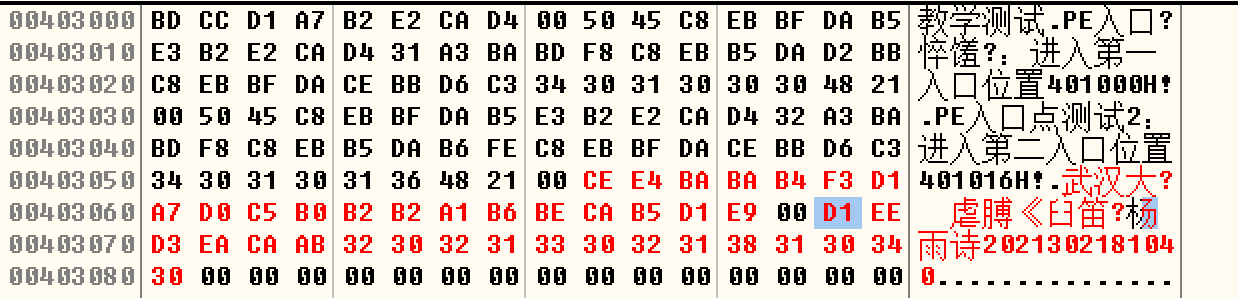


2.手工修改hello-2.5.exe程序，使得其可以弹出第三个对话框（提示框标题为“武汉大学信安病毒实验”，内容为：你的姓名+学号）

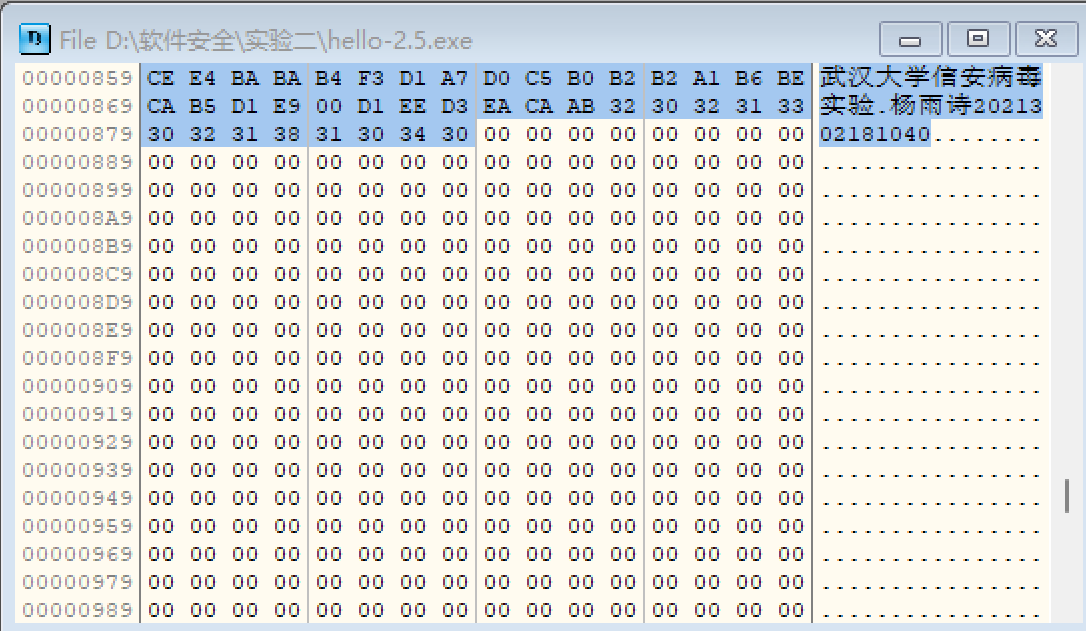
我们的思路为：

1. 修改.text段：添加一个MessageBoxA的函数调用
2. 修改.data段：添加一个MessageBoxA的参数信息
3. 修改.text和.data段的节表信息

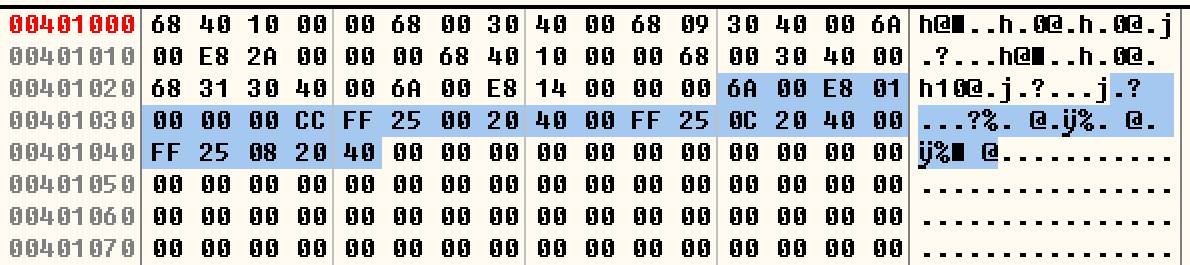
修改数据区



然后我们将上述写入内存复制到PE文件中

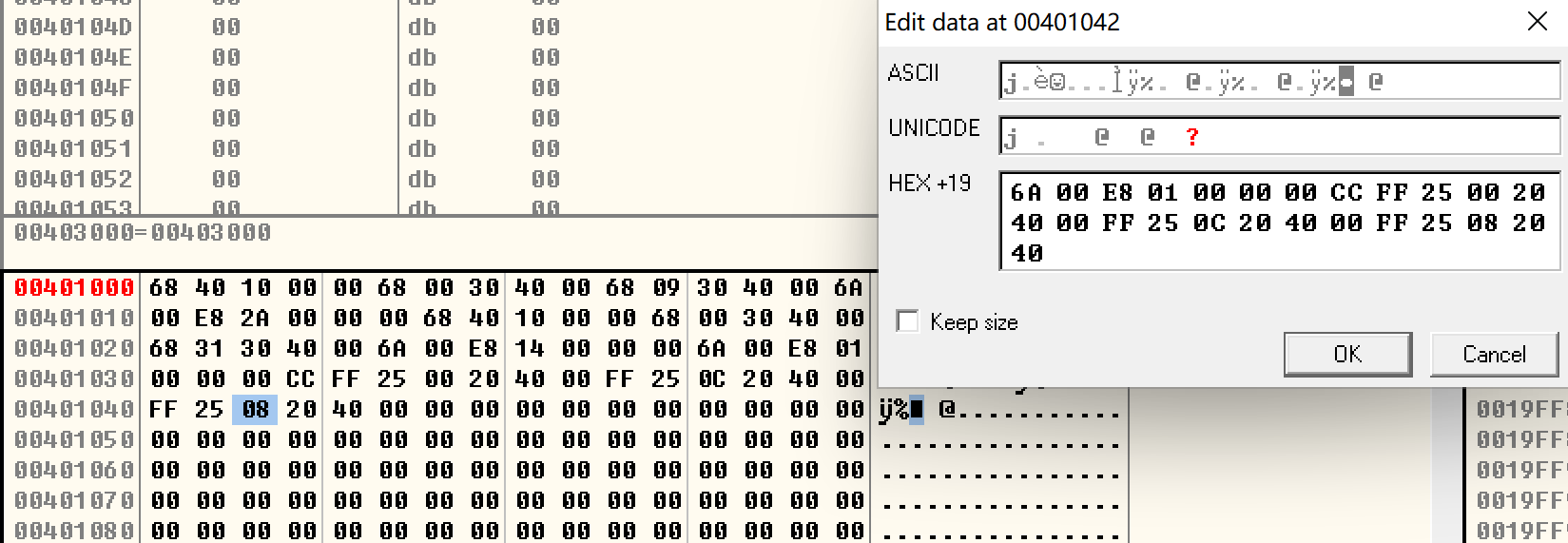


我们查看一下代码段数据区

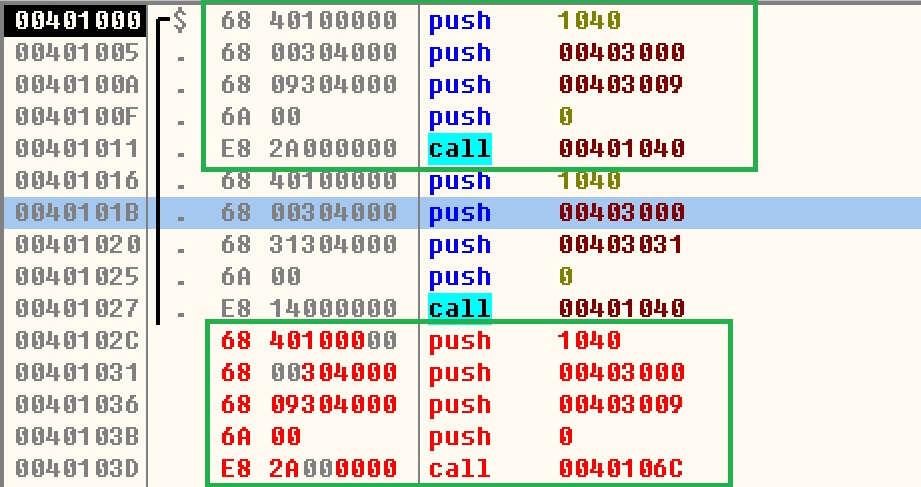


上图蓝色区为退出函数以及它引入函数的代码块

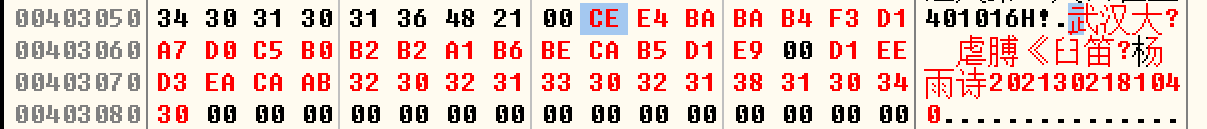
因为要让退出指令的代码段后移，我们将整体复制，预留出第三个弹框的22个字节后，将退出函数粘贴



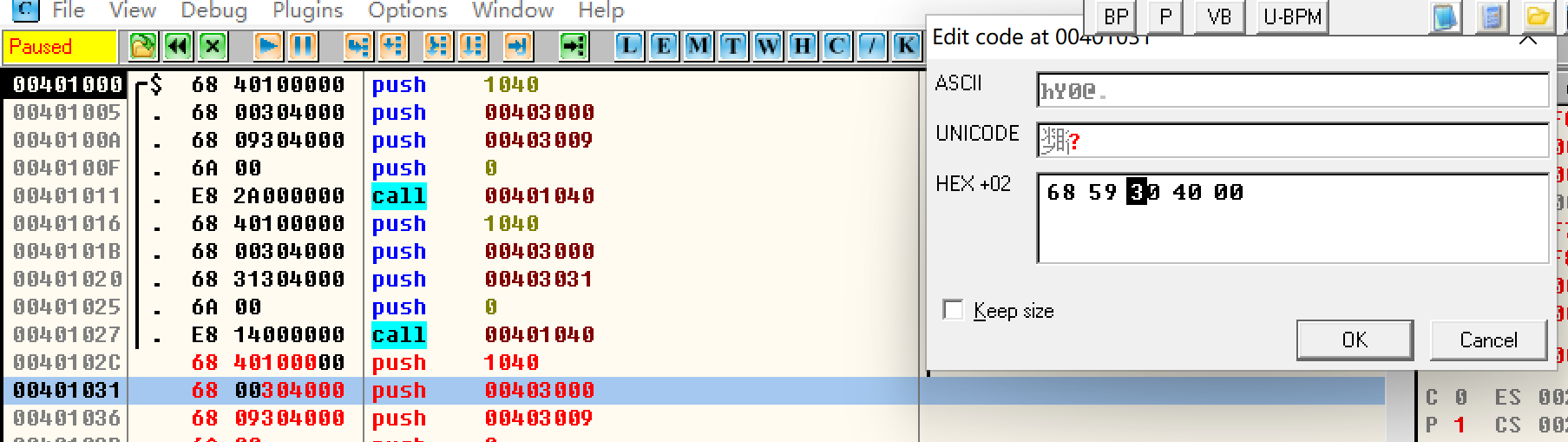
出于便捷，我们将第一个框的代码复制到第三个框处，再做修改



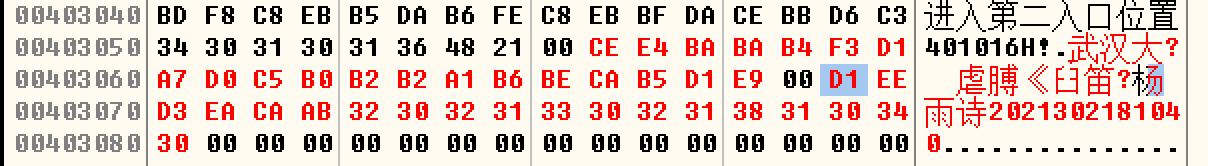
我们要修改上述字段，首先是数据框的标题，我们要将标题定位到我们刚才写入数据区的内容“武汉大学信安病毒实验”



因为我们的标题是从403059开始的，因此我们要把地址改为403059



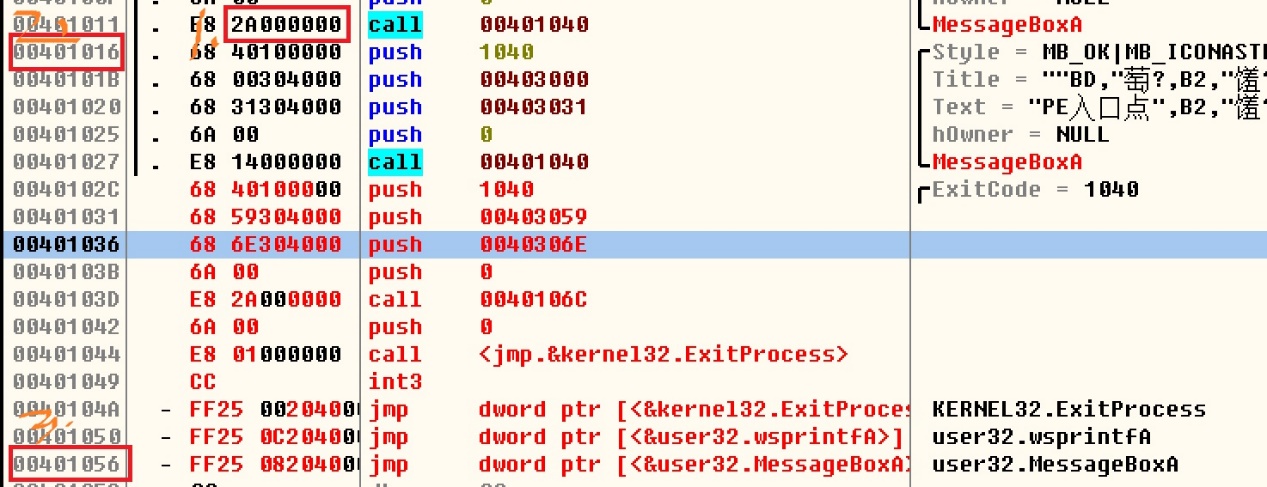
然后内容也要改成我的姓名+学号



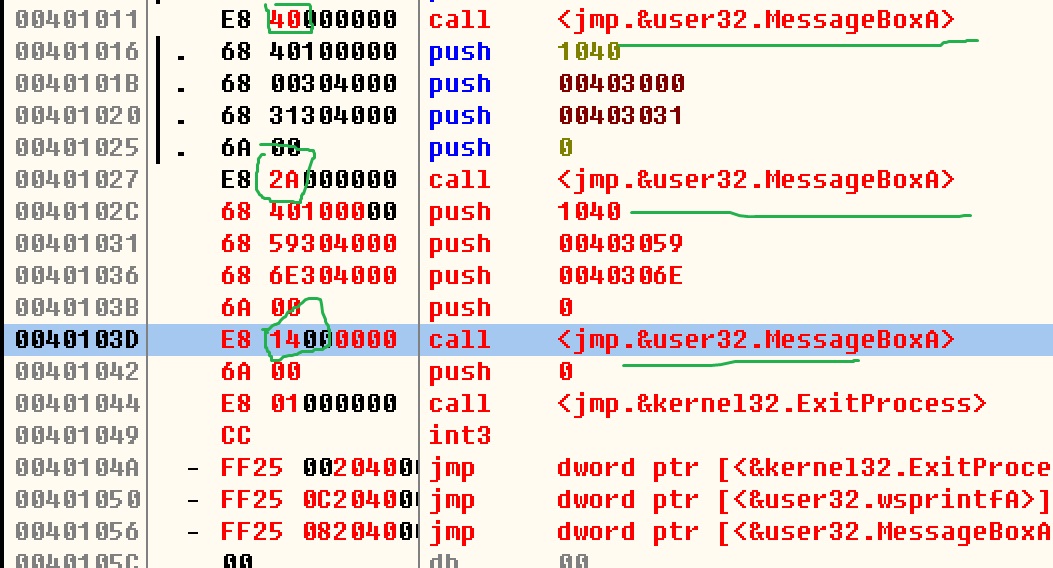
内容的起始位置为40306E



此时标题和内容均改好了，接下来我们要改MessageBox

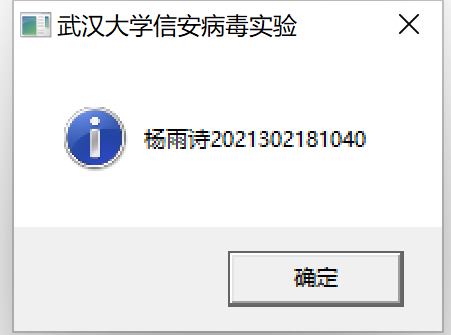


要求满足E8（call）后面的地址加上这条语句的下一条语句的位置等于MessageBoxAs所在的位置，如图所示就是1+2=3



如图所示，三个位置均已成功修改

然后保存，并命名为hello-2.5(2).exe,运行



弹出第三个对话框，修改正确

3. 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】

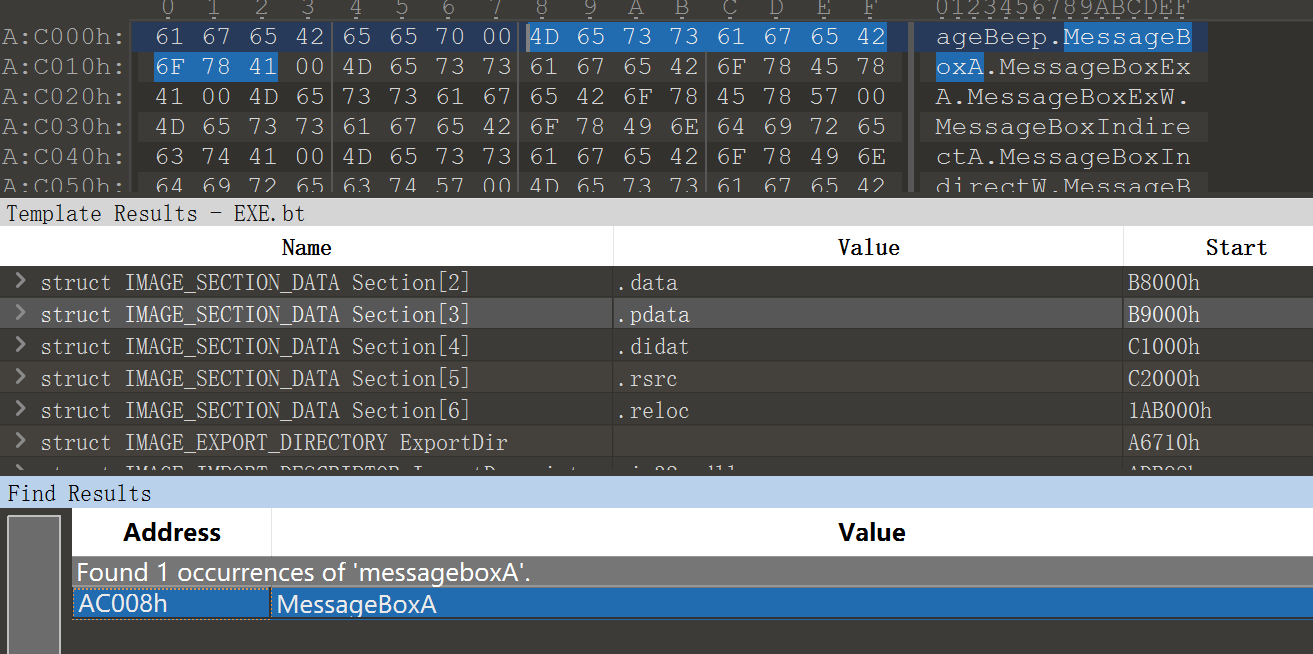
思路如下：

（1）从AddressOfNames指向的指针数组中找到“MessageBoxA”字符串，并记下该数组序号x

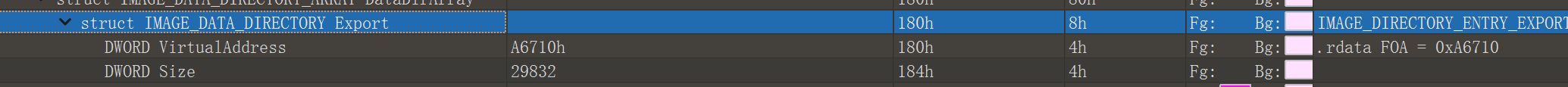
（2）从AddressofNameOrinals指向的数组中，定位第x项成员，得到一个序号y，y为MessageBoxA函数地址在AddressOfFunction中的索引号。

（3）从AddressOfFunction指向的数组中定位第y项，获得函数的RVA地址

首先搜索MesssageboxA

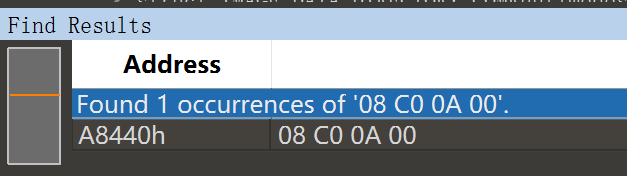


得到MessageBoxA在文件中的偏移地址为AC008h，然后我们要由此计算MessageBoxA这个文件名的相对序列地址

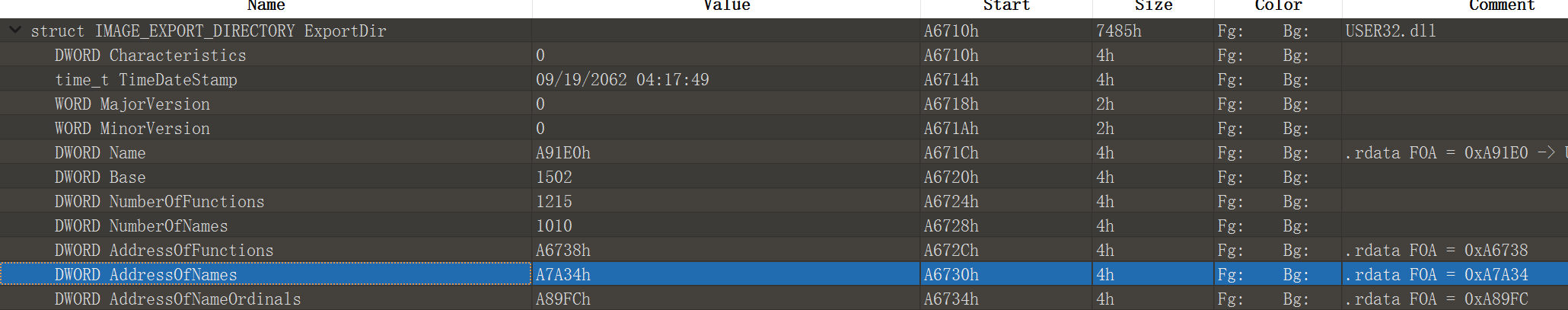


我们根据导出表的相对序列位置以及它对应文件内的偏移都是A6710可以计算出MessageBoxA的相对序列偏移与FOA相同，都是AC008h

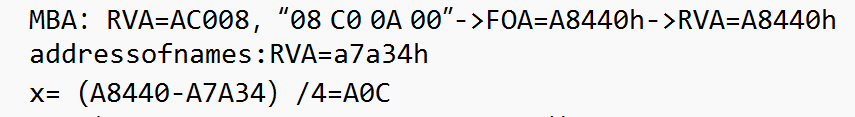
然后我们对序列偏移进行搜索



然后我们就定位到这个字符串的地方,它的文件偏移是A8440h



在导出表目录中它的AddressOfNames的RVA为A7A34h

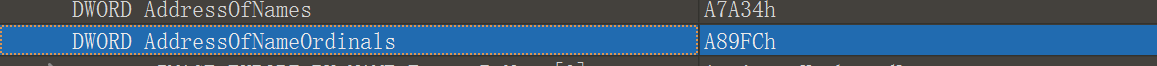


因此在addressofnames中MessageboxA的序号为A0C（十进制为643）

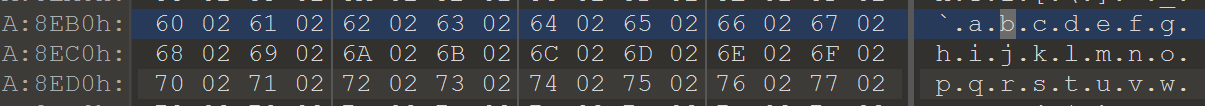
我们可以在表中进行验证



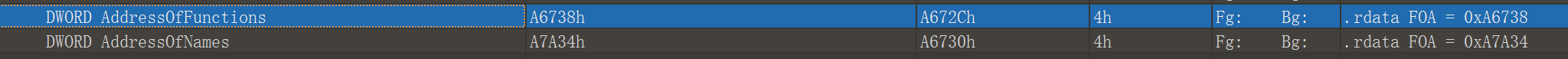
我们查看序号为643的导出表名，发现正好是MessageBoxA



AdressOfNameOridinals的序列地址为A89FCh，我们需要在该文件中定位到第x个元素：y=A89FC+x\*2=A89FC+A0C\*2=A8EB4



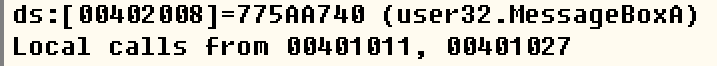
得到它的成员值是0262，最后在AddressOfFunctions定位到0262项



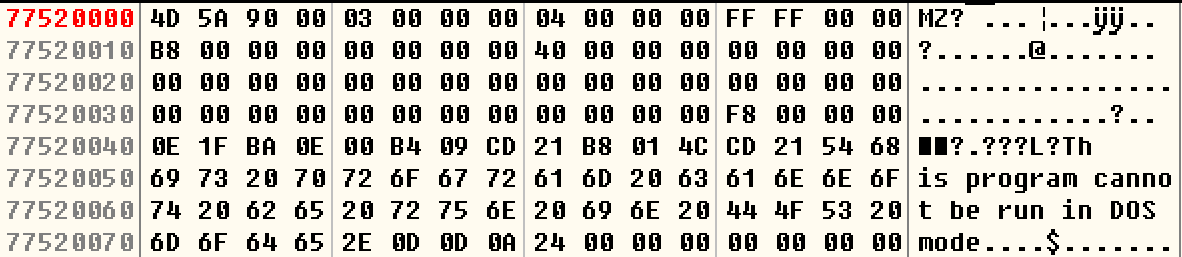




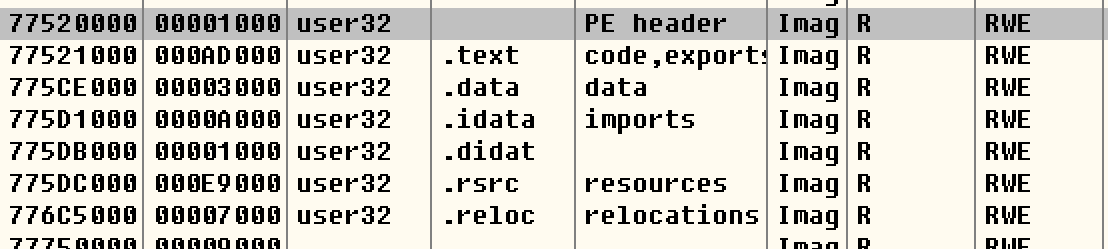
因此得出MessageBoxA的地址是00032C40



查看地址是775AA740，发现与我们的地址不同，是因为开启了地址随机化，相减得到基地址775200正好是user32的文件头

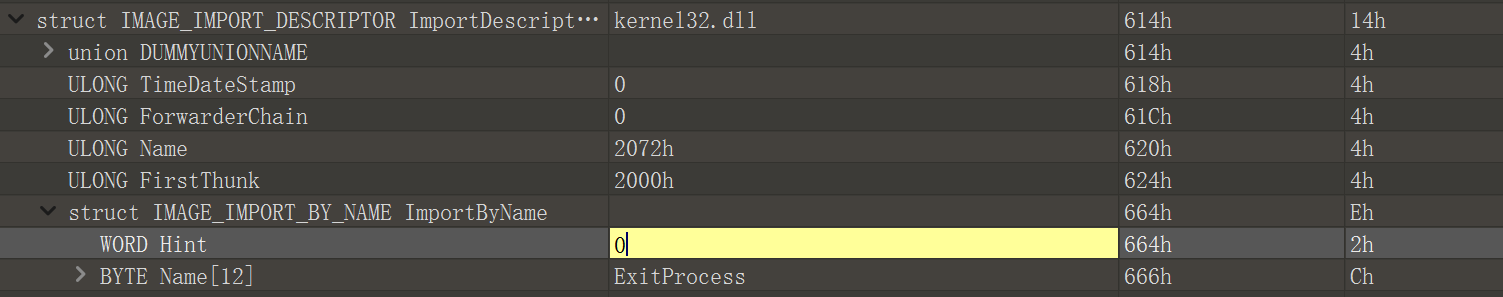


经检验我们找到地址是正确的

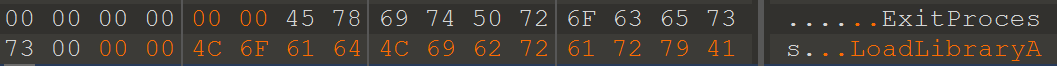


#### 1.4.3 在目标程序中新增功能代码

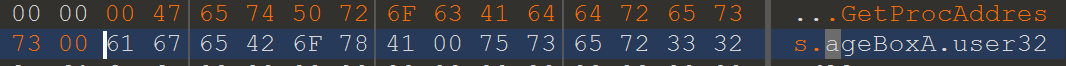
1. 修改IMPORT Hints/Names&DLL Names



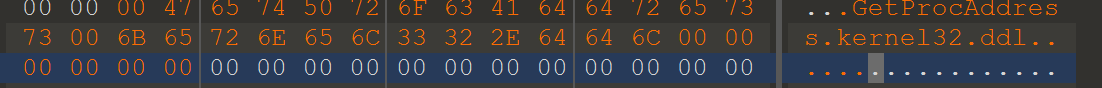
将kernel32的ExitProcess的Hint改为0



将Kernel32改为LoadLibraryA,最后用00截断



再引入GetProcAddress，最后用00截断



将kernel32放到最后，并将多余的位置清零

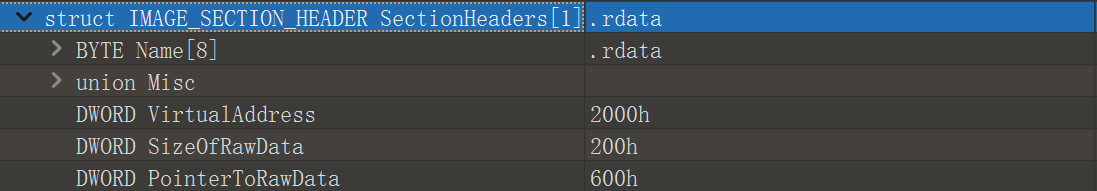
1. 修改INT表



要对FirstThunk进行修改，我们找到600的位置



由于data节中RVA=2000，文件偏移为600



因此计算得到600处的2064-2000+600 = 664，因此第一个导入函数在文件中的偏移是664

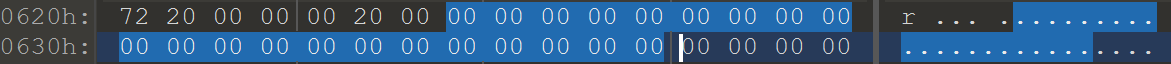
以此类推可以发现第二个trunk的值为672，第三个为681，因此修改如下



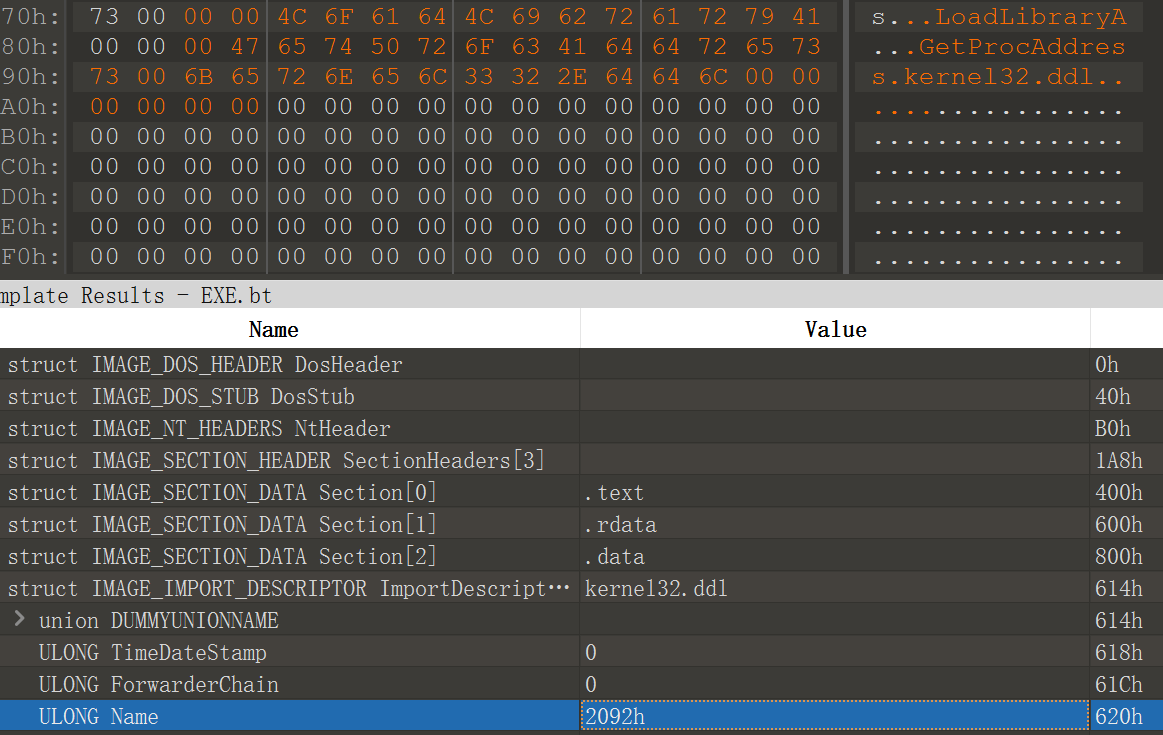
本例只引入三个函数，因此第四个字段要置零

1. 修改IDT表

由于我只引入了一个dll，因此我要将原来的usr32.dll清空



另外由于kernel32更改了位置，因此我要将name处的地址更改



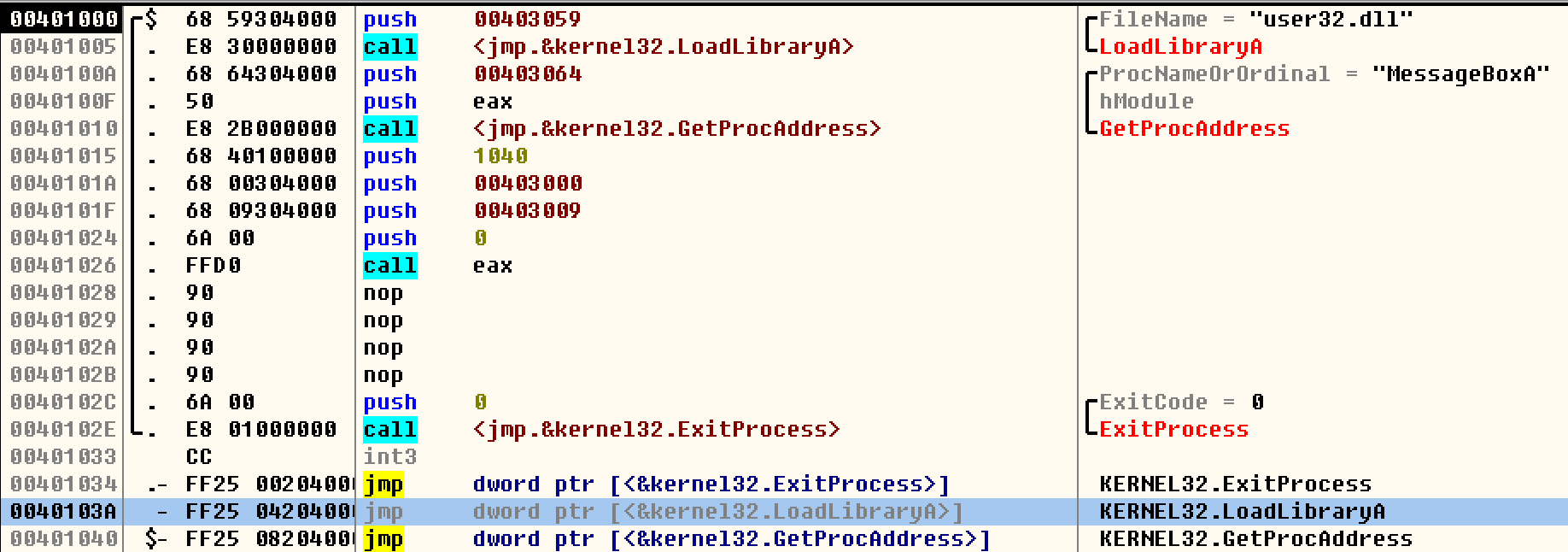
692对应2092

1. 在代码节中写入部分代码，利用这两个函数获取MesageBoxA的函数地址，使hello-2.5.exe正常

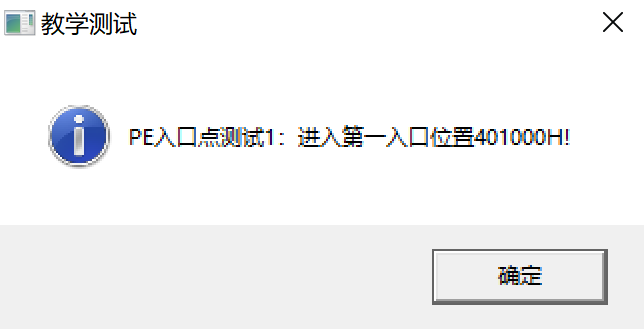
修改.data段，添加两个字符串



修改.text段



保存，打开我们保存的exe文件



#### 1.4.4 图标资源替换与软件手工汉化

资源表资源操作实践

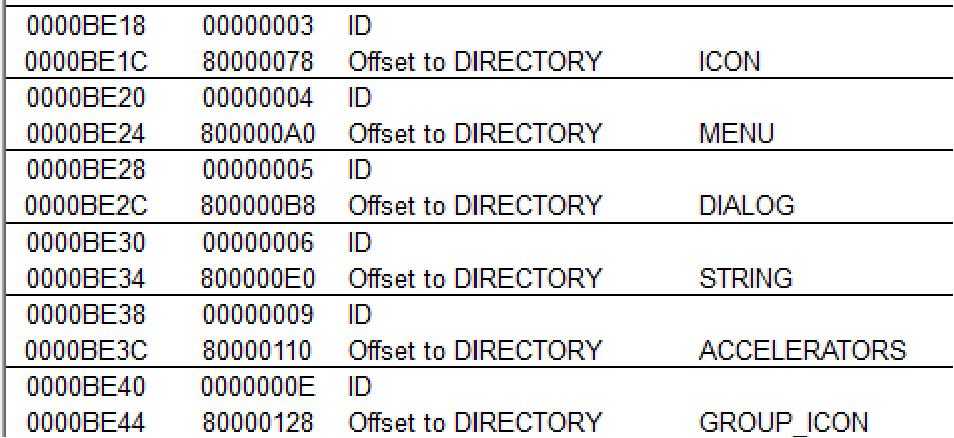
 利用PEview.exe分析PEview.exe程序

 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

资源节一般名为.rsrc，这个节放有如图标，对话框等程序要用到的资源，资源节是树形结构的，通常有三层目录（资源类型，资源标识符，资源语言ID），第4层是具体的资源

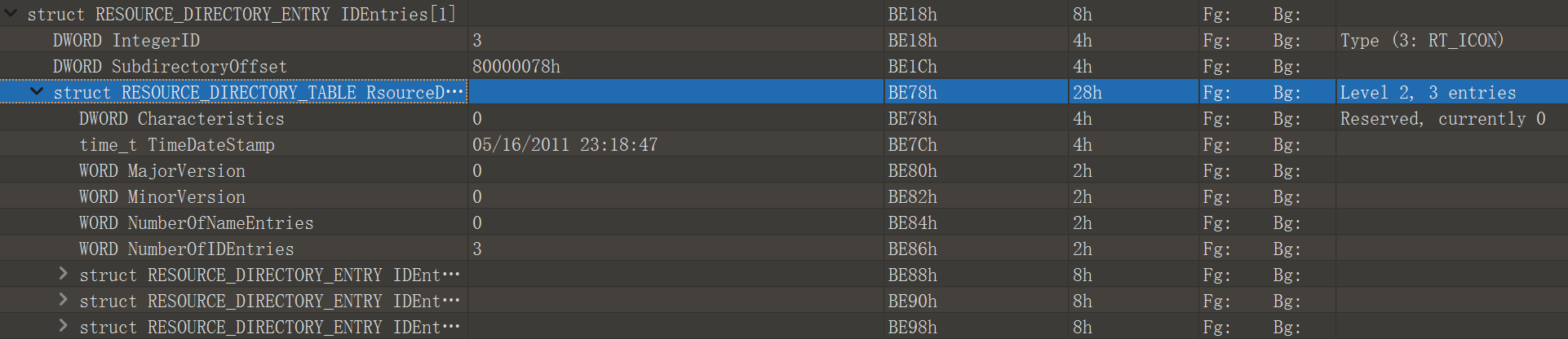
使用PEview.exe查看PEview.exe



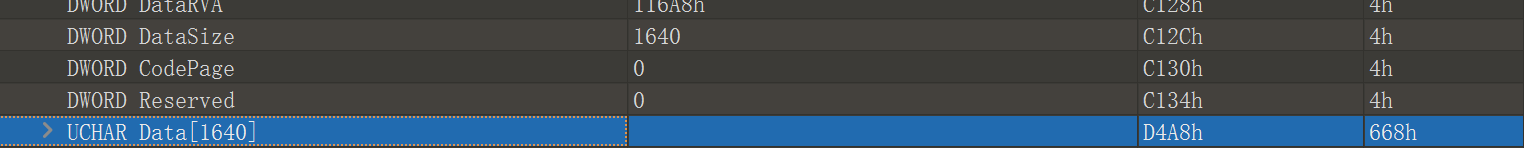
其图标在ID=3的位置处，GROUP\_ICON在ID=E处

ICON结构：图标资源的图像数据，对应ICO文件中的ICONIMAGE结构：

我们首先要定位到图标资源，地址为BE18



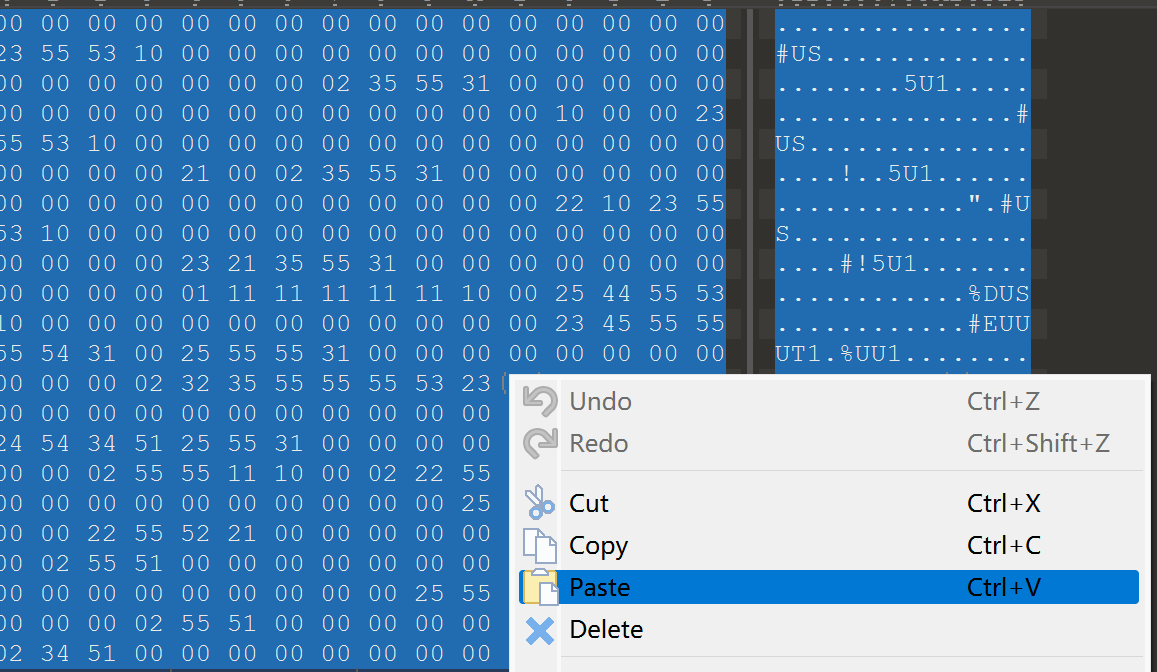
在010中查看ID为3的部分，可以看到它有三个图标资源（最底下）



找到要替换图标的地址，可以看到其大小为668h，大于我们要进行替换的目标图标大小

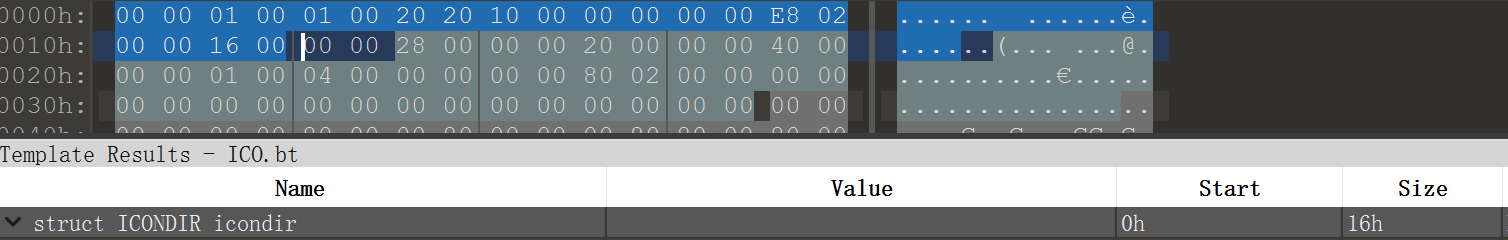


因此我们直接复制2E8大小的图标部分，粘贴到PEview中

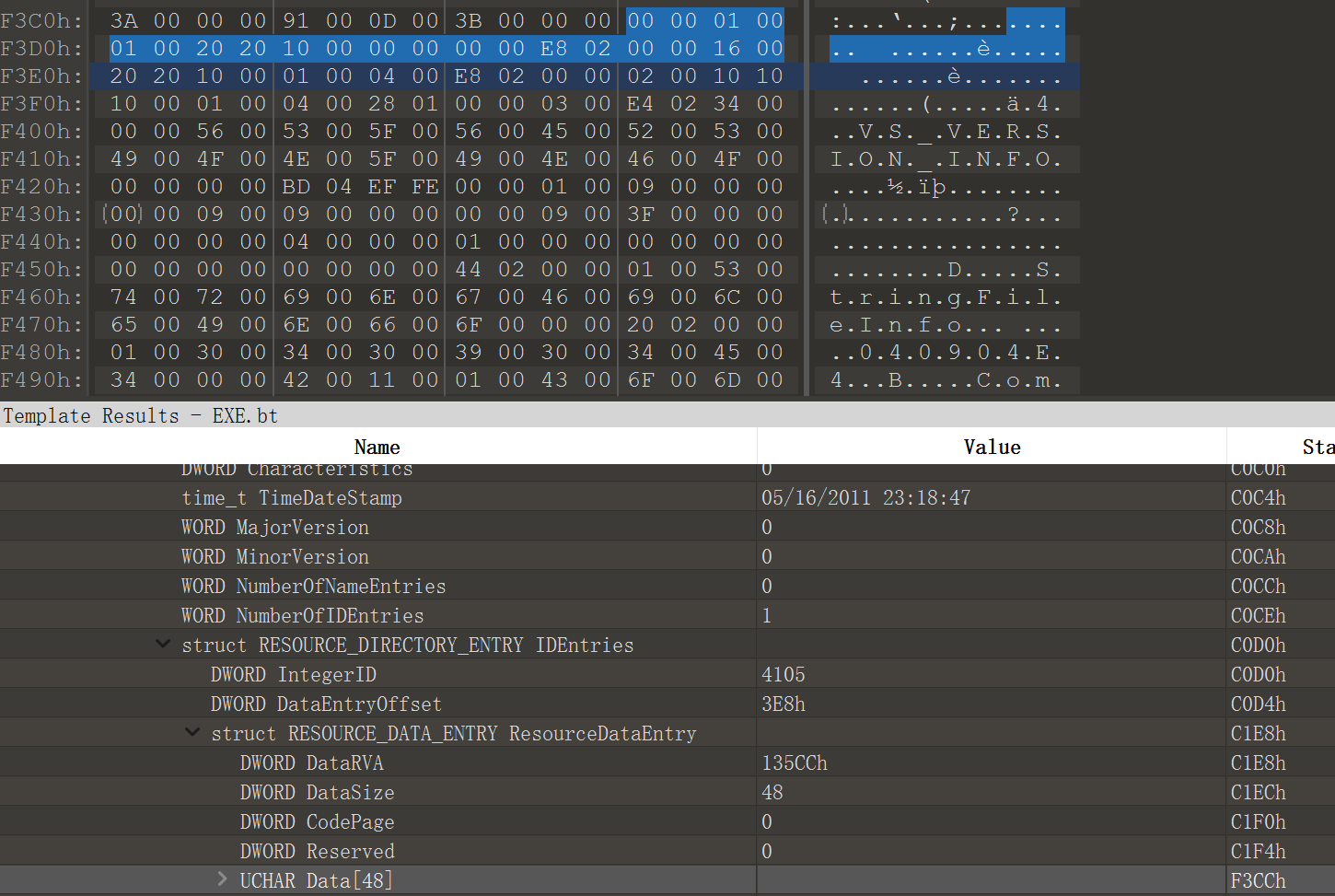


GROUP\_ICON:与ICO文件的ICONDIR结构相同，除了ICONIRENTRY最后一个word字段，ICONDIR结构最后是一个双字DWORD dwlmageOffset，GROUP\_ICON最后一个字段为word nID

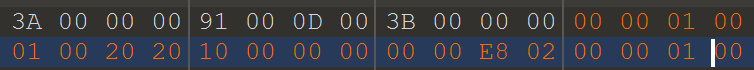
同时我们还要修改GROUP\_ICON：



复制ICONDIR，由于上面说的ICONDIR最后结构是一个双字，因此我们要少复制一个word过去



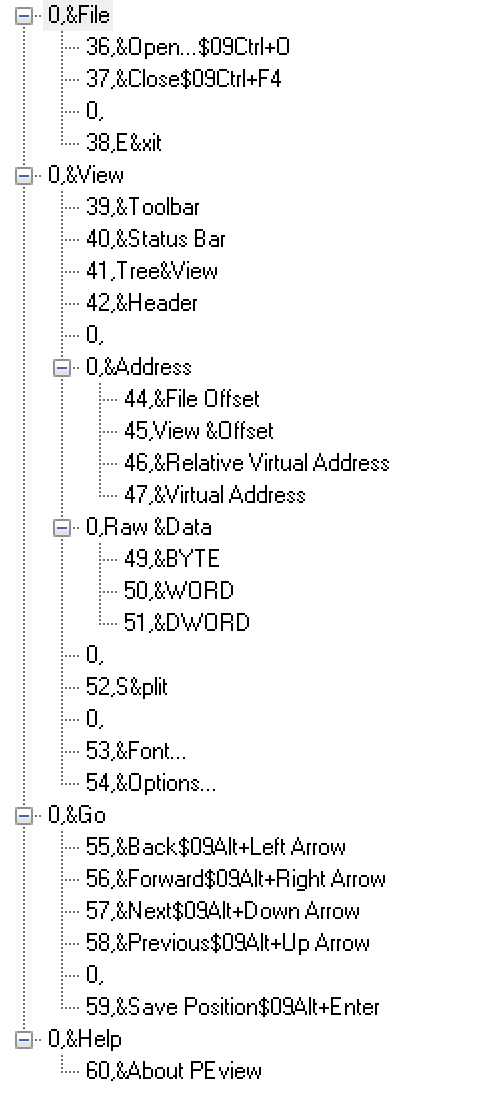
粘贴到GROUP\_ICON处，并且要修改最后一个word为对应的ICON ID



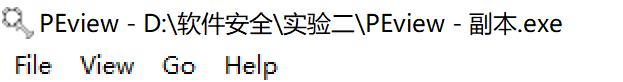
保存，就可以看到修改后的图标



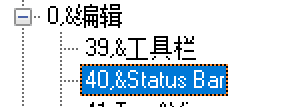
我们使用exeScope查看PEview



查看menu如上所示，跟我们正常打开PEView的菜单栏是一致的



因此我们就可以对菜单栏进行汉化



汉化完毕后如图所示



打开我们汉化后的PEview



可以看到被成功汉化

#### 1.4.5 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/>

在理论课第一个实验中实现了，不在此赘述

* 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。

详细的实验过程可以查看上面图标替换的实验，具体过程详见如下：

1. 打开PE文件：

- 使用CreateFile()函数打开PE文件，获取文件句柄。

- 使用CreateFileMapping()函数创建文件映射，将文件映射到内存中。

- 使用MapViewOfFile()函数将文件映射视图映射到当前进程的地址空间，获取指向文件的指针。

2. 定位资源节：

- 使用ImageNtHeader()函数获取PE文件的NT头。

- 通过NT头中的数据定位到资源表的位置。

3. 提取资源：

- 使用FindResource()函数定位到要提取的资源。

- 使用LoadResource()函数加载资源。

- 使用LockResource()函数锁定资源，并获取资源的指针。

- 可以将资源的指针转换为十六进制数据，并进行相应的处理和保存。

4. 替换资源：

- 使用BeginUpdateResource()函数开始资源更新过程。

- 使用UpdateResource()函数将新的资源添加到资源表中。

- 可以使用LoadResource()函数加载要替换的资源。

- 可以使用LockResource()函数锁定资源，并获取资源的指针。

- 可以将新的资源数据复制到资源的指针所指向的位置。

- 注意对资源表中的group字段进行相应的修改。

- 使用EndUpdateResource()函数结束资源更新过程。

5. 关闭PE文件：

- 使用UnmapViewOfFile()函数取消文件映射视图。

- 使用CloseHandle()函数关闭文件句柄。

* 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？

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资源节中的一个ICON资源项。资源节中的ICON资源项包含了不同尺寸和颜色深度的图标资源。每个ICON资源项由一个或多个图像资源组成，每个图像资源对应一个特定尺寸和颜色深度的图标。

图标替换策略应该根据不同的操作系统版本来进行调整。在资源节中，每个ICON资源项都有一个与之对应的ID或名称。可以通过读取资源节中的ICON资源项的ID或名称来判断每个图标资源对应的操作系统版本。

根据不同的操作系统版本，可以选择相应的图标资源进行替换。然后，根据目标程序的操作系统版本，选择合适的替代图标资源进行替换。

可能的具体的替换策略如下：

1. 读取目标程序的操作系统版本信息。

2. 根据操作系统版本信息，确定要替换的ICON资源项的ID或名称。

3. 使用BeginUpdateResource()函数开始资源更新过程。

4. 使用UpdateResource()函数将新的图标资源添加到资源表中，替换对应的ICON资源项。

5. 结合资源表中的group字段，确保替换后的图标资源与原图标资源在资源组中的位置保持一致。

6. 使用EndUpdateResource()函数结束资源更新过程。

* 资源节与恶意代码有何关联？

资源节与恶意代码之间存在密切的关联。资源节是PE文件中的一个节区，用于存储程序所需的各种资源，如图标、位图、字符串等。恶意代码可以利用资源节来进行各种攻击和伪装。

以下是资源节与恶意代码之间的关联：

1. 图标替换攻击：恶意代码可以修改资源节中的图标资源，将原始图标替换为恶意图标。这种攻击方式常见于病毒或恶意软件，通过替换图标，使用户误认为文件或程序是安全的，从而诱导用户执行恶意代码。

2. 图标伪装攻击：恶意代码可以将自身的图标（exe文件图标）更改为某个常见的应用或文件的图标，如QQ、txt文本文件等。这种伪装可以欺骗用户，让用户误以为恶意程序是无害的常见文件，从而诱导用户点击运行恶意程序。

3. 驱动或程序嵌入：恶意代码可以将嵌入的程序或驱动放在资源节中，程序运行时会加载这些资源并执行其中的恶意代码。通过将恶意代码嵌入资源节，攻击者可以隐藏恶意行为，增加检测的难度。

4. 恶意代码伪装：某些恶意代码本身就存在于资源节中进行伪装。攻击者可以将恶意代码以资源的形式存储在资源节中，通过修改资源节的属性和标识，使其看起来像是合法的或无害的资源。这种伪装可以绕过某些安全检测机制，增加恶意代码的存活时间。

总之，资源节提供了恶意代码进行伪装、攻击和隐藏的机会。攻击者可以利用资源节来替换图标、伪装为常见文件、嵌入恶意代码或直接将恶意代码存储在资源节中，以实现各种恶意行为。因此，在分析和处理恶意代码时，资源节是一个重要的部分，需要仔细检查和分析其中的内容。

* 什么是HOOK？其与本章学习有何关系？

HOOK（钩子）是一种特殊的消息处理机制，用于监视系统或进程中的各种事件消息，并截获这些消息进行处理。通过钩子机制，我们可以自定义钩子来监视系统中特定事件的发生，并完成特定功能，如屏幕取词、监视日志、截获键盘和鼠标输入等。

在本章学习中，我们涉及了恶意代码的分析和检测。钩子机制在恶意代码分析中发挥着重要的作用，与本章学习的内容有以下关系：

1. 监视系统事件：钩子可以用于监视系统中特定事件的发生，例如截获键盘和鼠标输入。在恶意代码分析中，了解恶意代码对用户输入的截获和监视行为是很重要的，可以帮助我们理解恶意代码的功能和行为。

2. 拦截系统调用：钩子可以拦截系统函数的调用，截获相关的消息，并对其进行自定义处理。这与本章学习中分析恶意代码的行为和功能密切相关。恶意代码可能会尝试调用系统函数来执行恶意操作，通过自定义钩子，我们可以拦截这些调用、阻止恶意操作的执行或修改调用函数的信息，以达到分析和检测的目的。

3. 动态链接库（DLL）：钩子函数必须放置在独立的动态链接库（DLL）中。在本章学习中，我们也提到了在分析恶意代码时，常常需要对样本进行动态分析，使用动态链接库是一种常见的方法。通过将钩子函数放置在DLL中，可以方便地加载和注入到目标进程中，实现对恶意代码的监视和拦截。

综上所述，HOOK是一种Windows消息的拦截机制，可以拦截单个进程或所有进程的消息，并对拦截的消息进行自定义的处理。在本章学习中，我们关注恶意代码的分析和检测，钩子机制在恶意代码分析中起到重要的作用，帮助我们理解恶意代码的行为和功能，并进行相应的分析和检测工作。

### 1.5实验体会和拓展思考

在本次实验中，我通过学习和实践PE文件的编辑与查看，深入了解了PE文件的结构和各个部分的作用。我通过使用二进制编辑工具和Ollydbg对例子程序hello-2.5.exe进行了初步的观察和调试，了解了该程序的结构和功能。在对PE文件头部、引入表的结构进行分析，理解了函数的引入与引出机制，我进行了手工修改hello-2.5.exe程序，使其能够弹出目标对话框。最后我通过利用PEview.exe和eXeScope工具对PE文件进行资源操作实践。成功修改了PEview.exe的图标为csWhu.ico，并使用eXeScope工具对PEview.exe进行了汉化操作。这个部分还是蛮有趣的，感觉更侧重于学习一些工具的使用，对后续的学习还是很有帮助的。