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：_____

2020 年　月　日

目 录

实验 2　PE 文件格式分析(模板)

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 PEview 观察 Hello25 的 16 进制数据，如图 1.1 所示

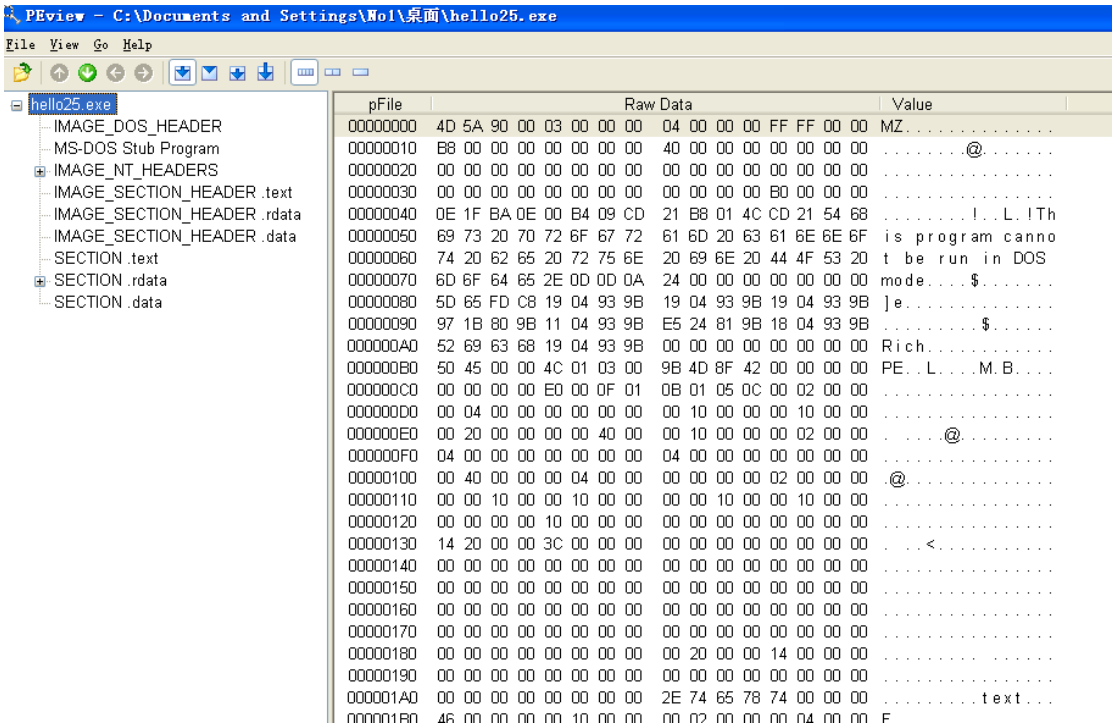


图 1.1 PEView 观察 Hello25.exe

使用 PEView 自带的分析模板，可以看到这个例子程序的结构，我们可以看到，hello25.exe 文件结构包括 MZ 文件头、Dos 桩、PE 文件头、节表、代码节、引入函数节和数据节，如图 1.2 所示。

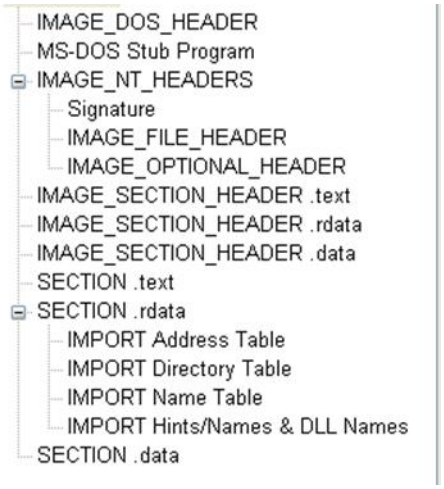


图 1.2 PE 模板分析

下面使用 010Editor 使 PE 程序只弹出第二个框，使用 Ollydbg 打开程序，可以看到：例子程序在内存中的对齐单位是 1000h，而文件中的对齐单位是 200h。随后使用 Dllydbg 自带的单步调试工具一步步执行程序，可以看到程序陆续弹出两个对话框，随后退出。如图 1.3 所示

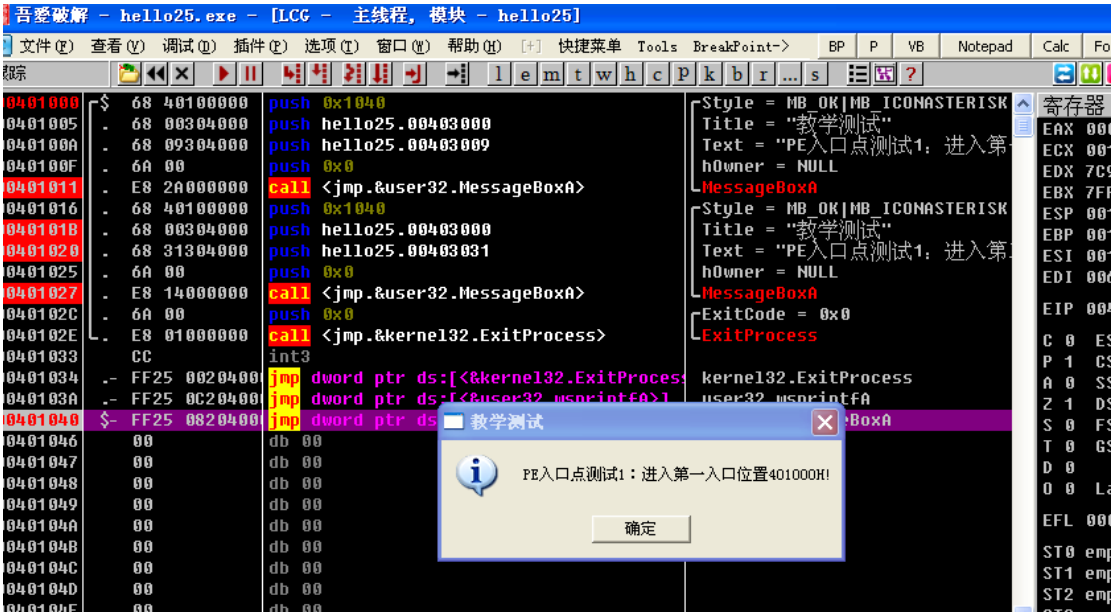


图 1.3 Ollydbg 单步调试

为了直接弹出第二个弹框，查看程序的开始地址和第二个框的地址，如图 1.4 所示

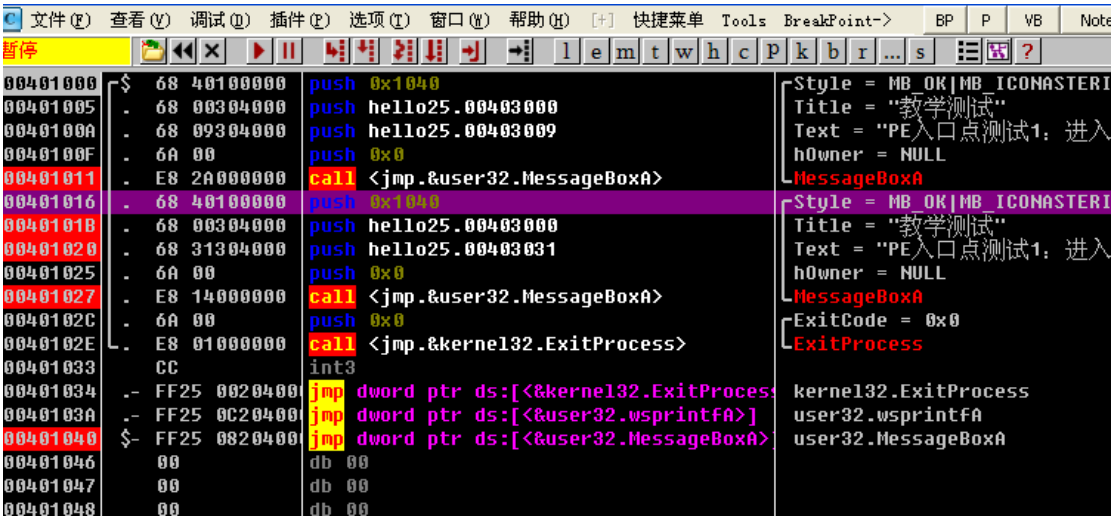


图 1.4

可以看到两个弹窗函数的入口地址，那么如果我们需要直接弹第二个窗，我们只需要将程序的入口点由 401000h 修改为 401016h 即可，使用 010Editor 修改如图 1.5 所示

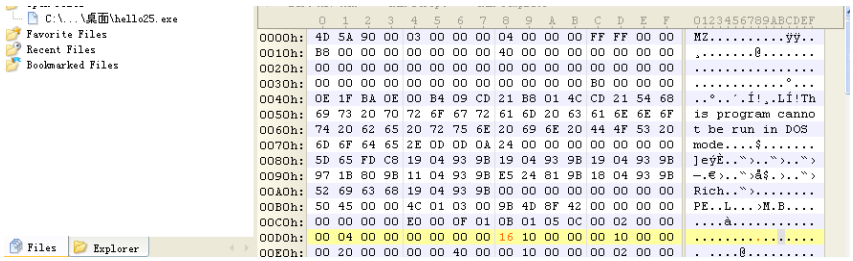


图 1.5 010Editor 修改

保存程序，点击运行，可以发现直接弹出第二个框，如图 1.6 所示，再使用 Ollydbg 查看可以发现程序的装载地址发生了变化

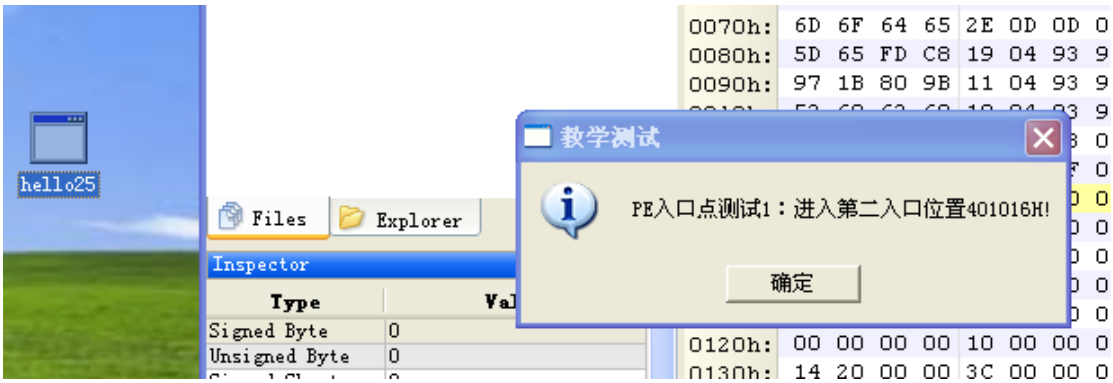


图 1.6 测试

1.4.2 函数的引入与引出机制

使用 Stud_PE 加载 hello25.exe 程序，可以看到 PE 文件头的具体信息，如图 2.1 所示

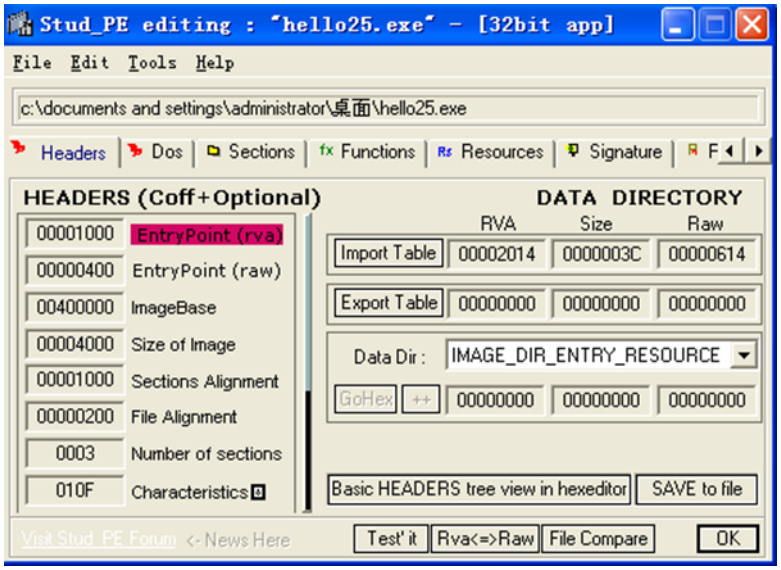


图 2.1 StudPE 分析

可以看到，程序的 RVA 为 1000h，而 RAW 为 400h，程序优先装载地址（ImageBase）为 400000h。程序在内存中的对齐粒度为 1000h，而在文件中的对齐粒度为 200h。3 个节。另外引入函数节的 RVA 为 2014h，大小为 3Ch，并且 RAW 为 614h

打开 Ollydbg，在内存数据区添加要新增的弹框的数据，复制到可执行文件，如图 2.2 所示

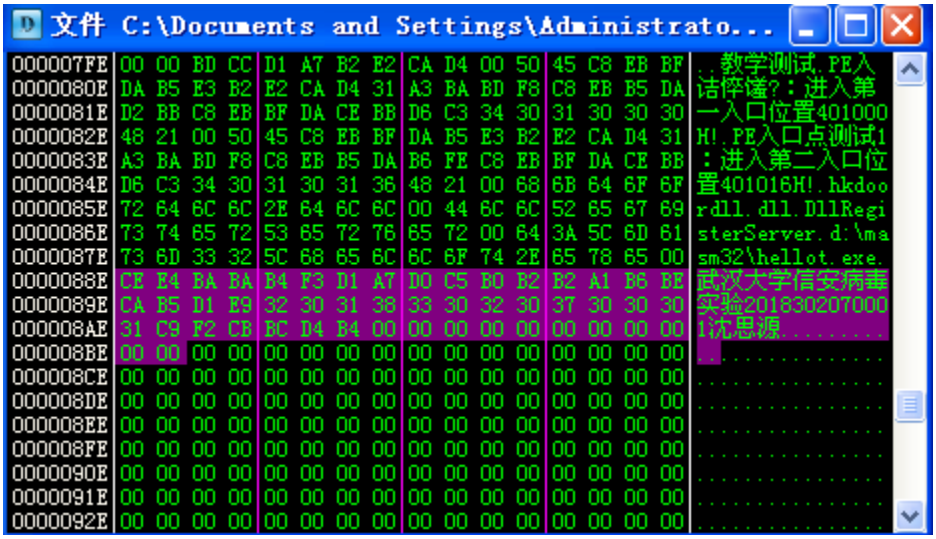


图 2.2 新增数据

使用 Ollydbg，找到程序的装载地址，来添加一个弹窗相应的代码，首先将跳转指令向后放 22 个字节（一个弹窗所需的代码为 22 字节），随后复制第一个弹窗的代码部分，如图 2.3

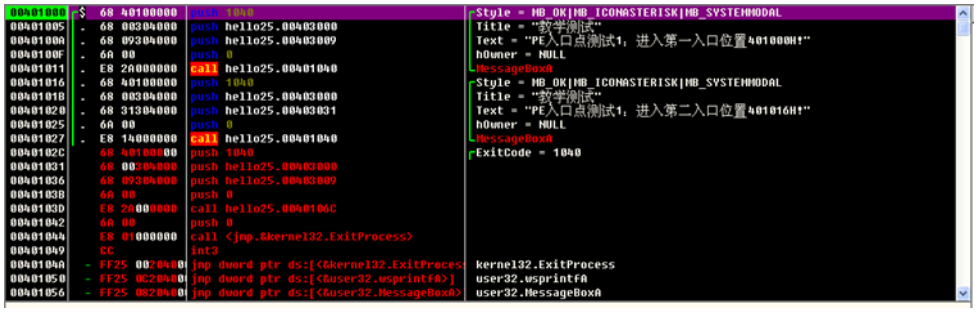


图 2.3 新增代码

修改 push 参数和 call 指令的地址，让其能够找到被移动后的 MessageBoxA 的位置，如图 2.4 所示

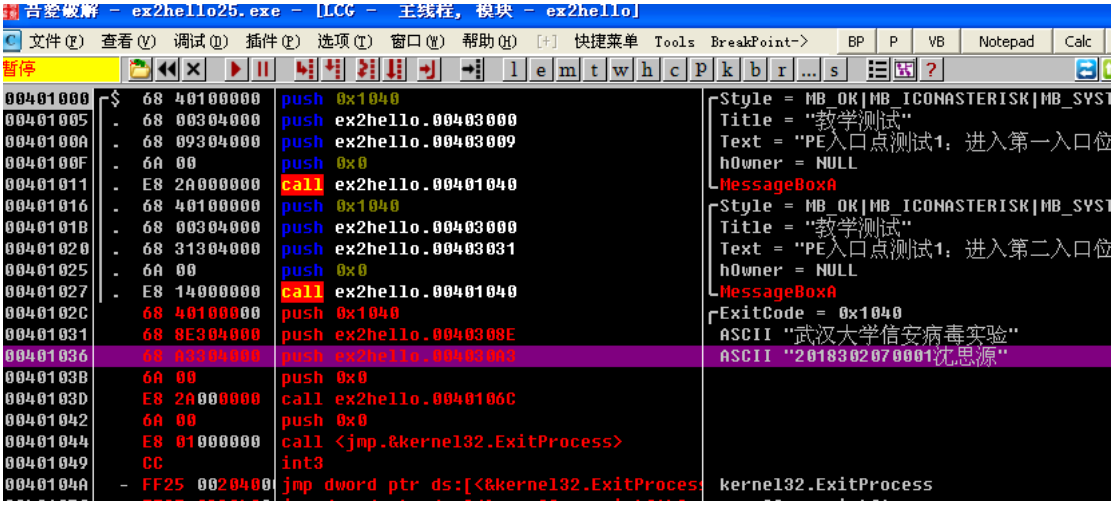


图 2.4

将其保存到可执行文件中，执行，第三个框的效果如图 2.5 所示

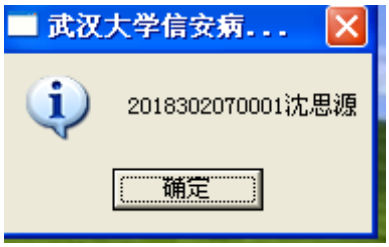


图 2.5 第三个框

使用 OllyDbg 打开 hello25.exe 程序可以看到，MessageBoxA 函数加载的地址为 77D5050B，如图 2.6 所示

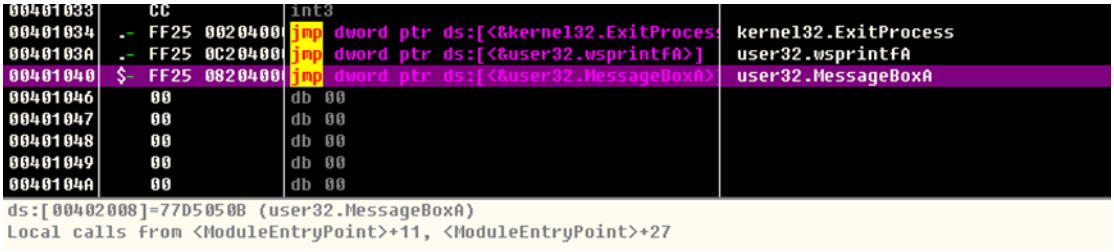


图 2.6 内存中的 MessageBoxA

再通过 PEView 打开 user32.dll 文件，打开导出函数的名字表，找到 MessageBoxA 函数，我们可以看到此时 MessageBoxA 的 RVA 为 4BC0h，通过计算可得 MessageBoxA 在名字表中的序号=（4BC0-4450）/4=1DCh，如图 2.6 所示

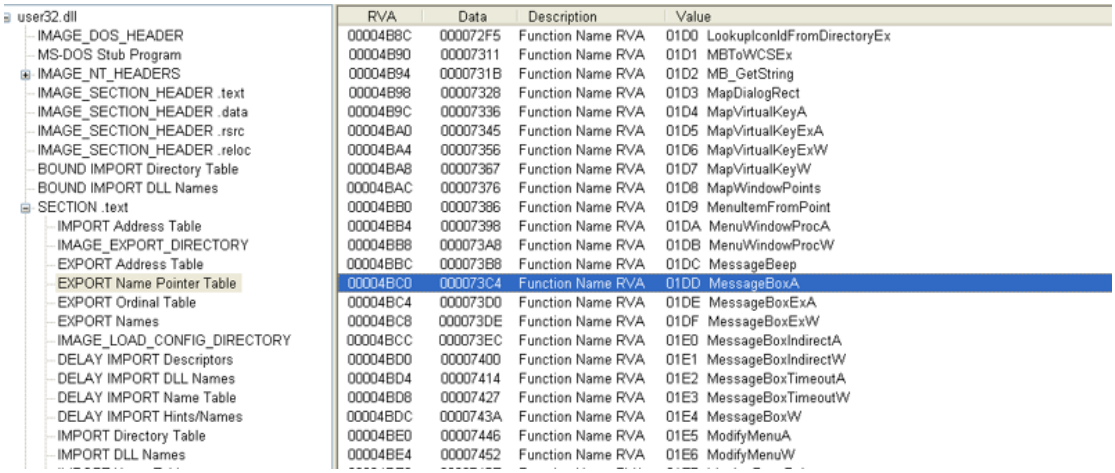
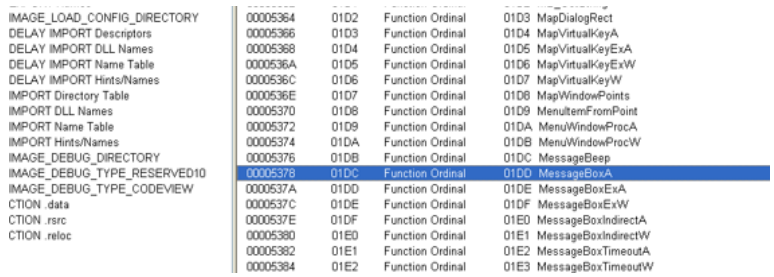


图 2.6

打开序号表找到第 1DCh 项，可以看到正好是 MessageBoxA 函数，并且它在序号表中的序号也是 1DCh：



打开引出地址表，我们找到第 1DCh 项可以看到，MessageBoxA 函数的 RVA 为 4050Bh,如图 2.7 所示

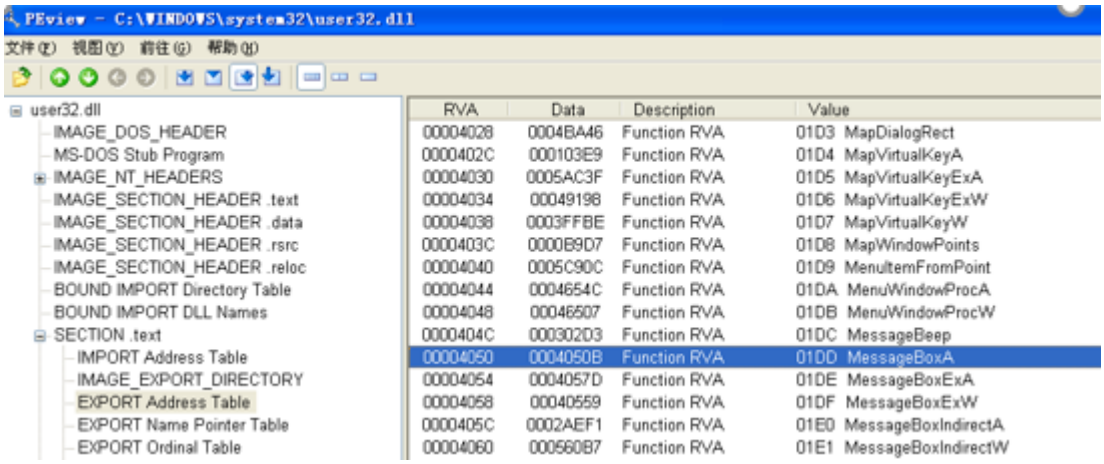


图 2.7　引出地址表

再通过 PE 可选文件头查找 user32.dll 文件的 ImageBase，可以看到该值为 77D1000h，如图 2.8 所示

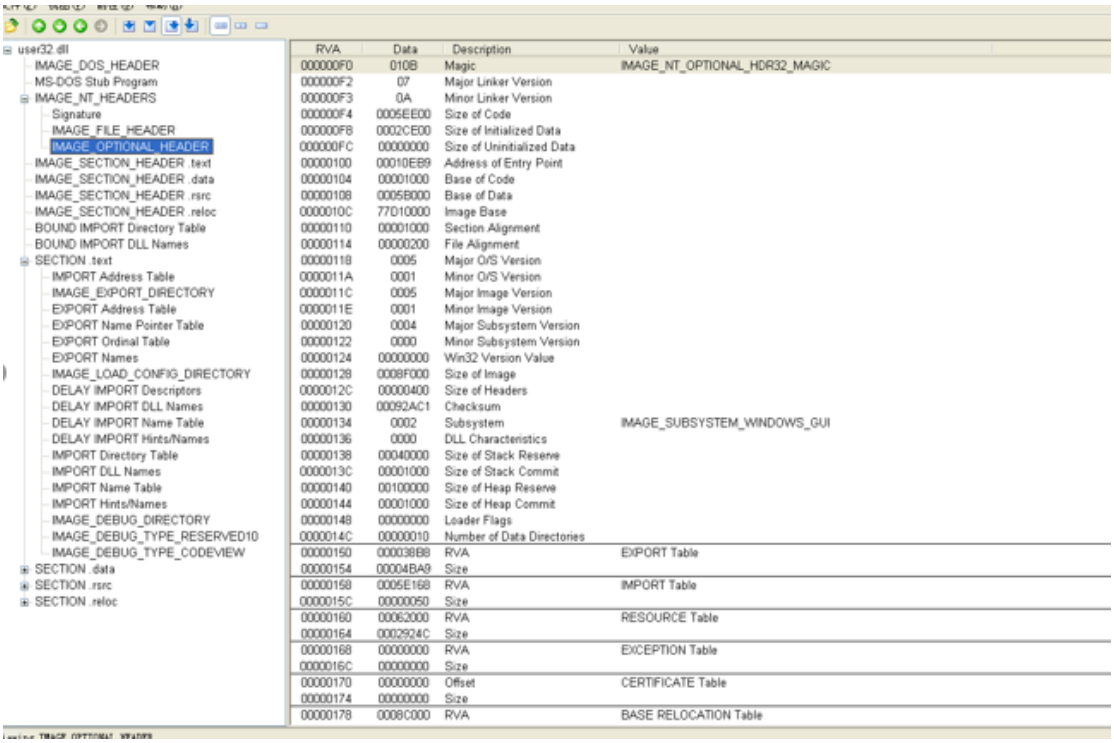


图 2.8　ImageBase

得到 MessageBoxA 函数在内存中加载的地址为 77D10000+4050B=77D5050Bh
与图 2.6 进行比对，发现正确，成功找到了 MessageBoxA 的地址

1.4.3 在目标程序中新增代码

使用010Editor打开hello25.exe，将所有需要引入函数的函数名字符串加入到引入函数节中，随后将 Hints 字段全部改为 0，由于不从 user32.dll 中引入函数，将 IDT 第二个部分直接清 0，最后修改 IAT 的相应字段，使其能够找到更新的各个函数和 dll 文件，如图 3.1 所示

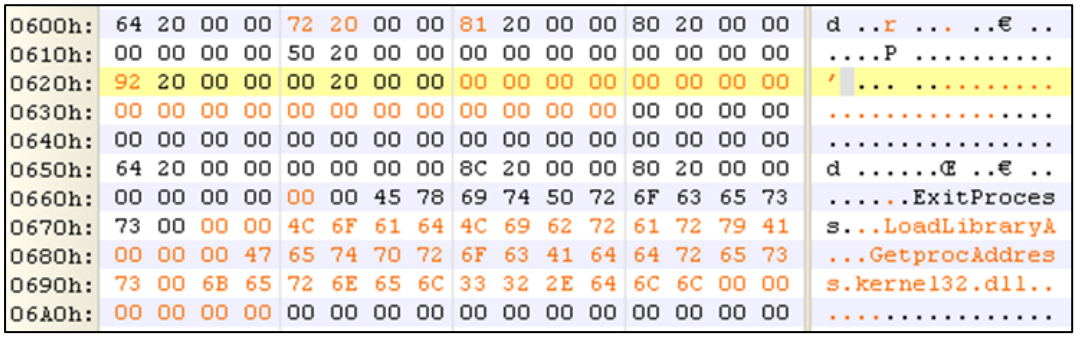


图 3.1　引入函数

尝试运行 PE 文件，发现不能运行，但是这是个好现象，说明 PE 整体结构是没有问题的。如图 3.2 所示

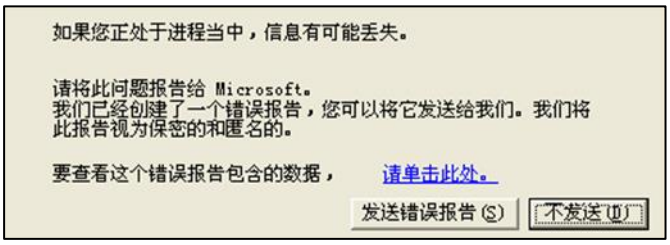


图 3.2　无法运行

通过 OllyDbg 打开修改的程序，在数据区填充 user32.dll 和 MessageBoxA 字段，并且修改代码段，如图 3.3 所示

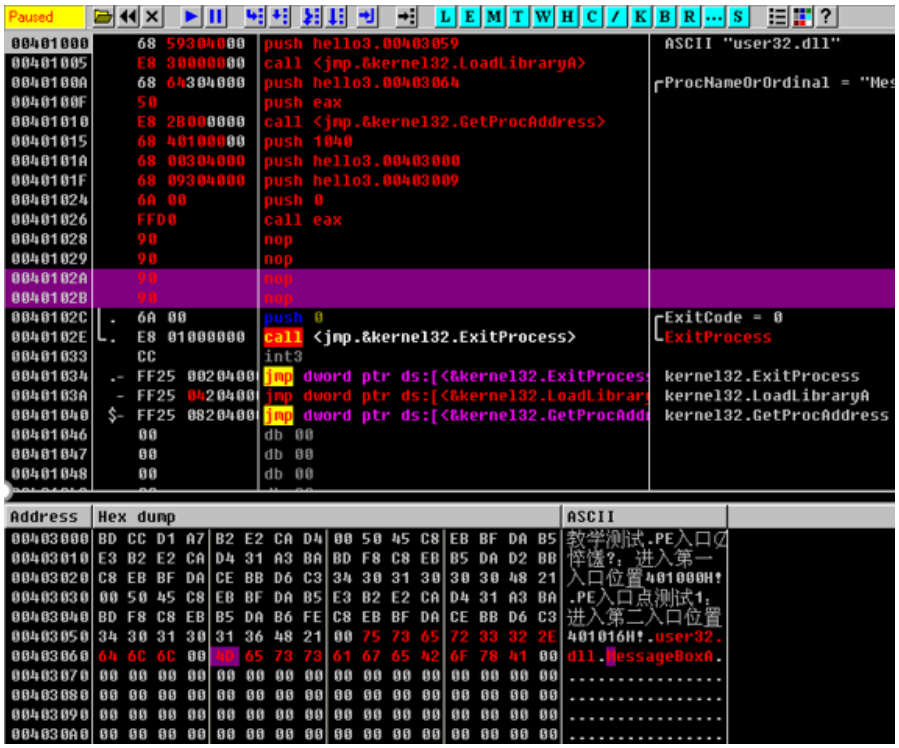


图 3.3　修改程序

保存，启动程序进行单步调试，可以发现程序又可以进行弹框，恢复了原有的功能。如图 3.4 所示

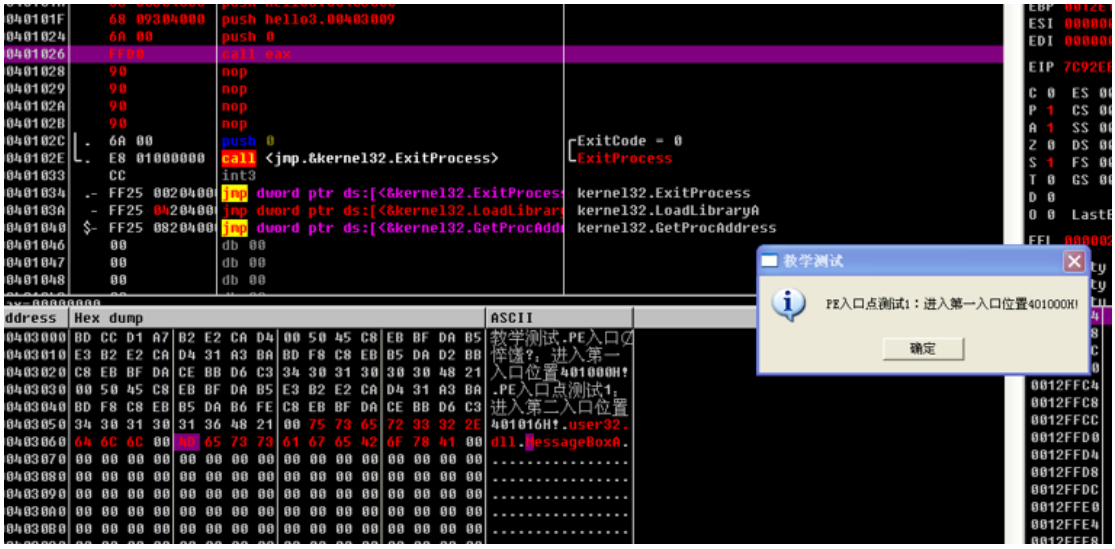


图 **3.4**　测试

1.4.4 图标资源替换与手工汉化

通过 PEview 打开另一个要被汉化的 PEView 复件，找到 ICON 0001，我们可以看到复件 PEView 的图标大小为 2E8h,如图 4.1 所示

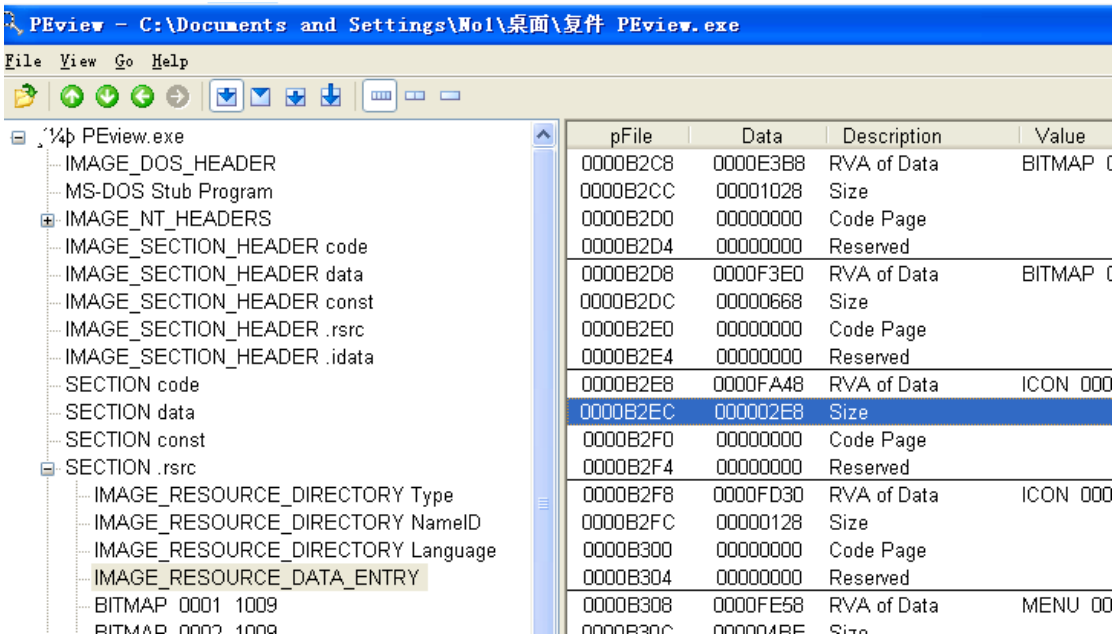


图 **4.1**

接着通过 PEView 打开图标资源的 16 进制数据，如图 4.2 所示

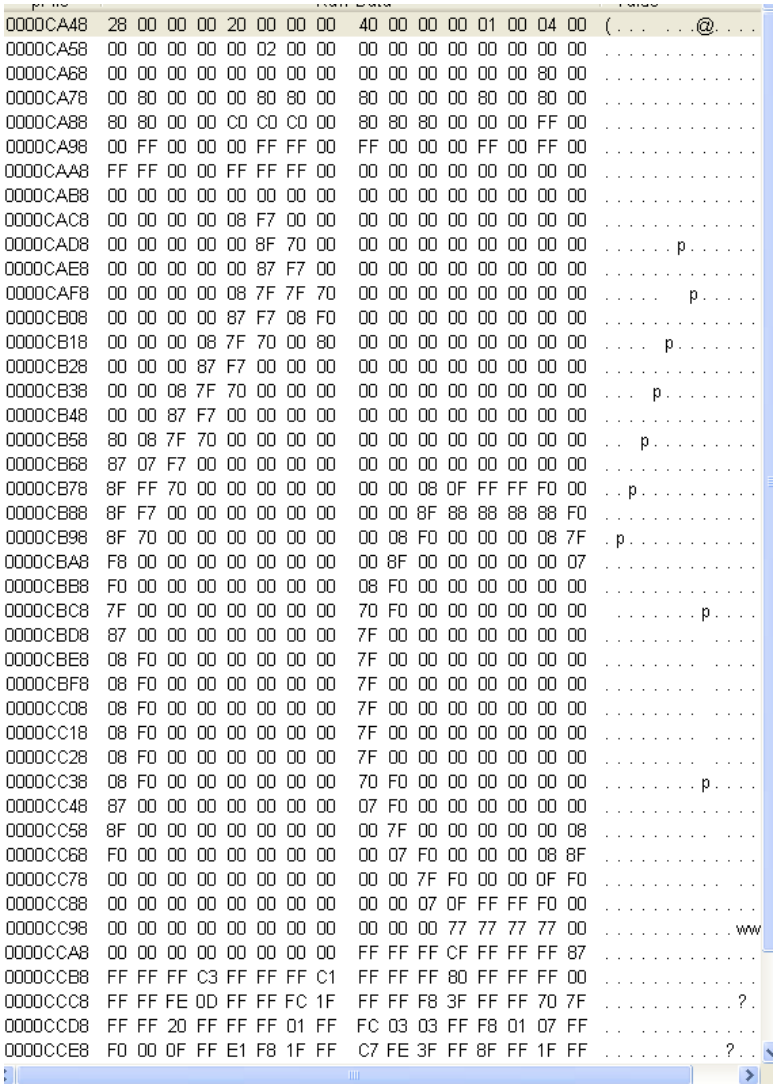


图 **4.2**

修改要替换的图标大小到 0x2e8h，用 ultraEdit 打开，复制到 PEView 图标资源对应部分

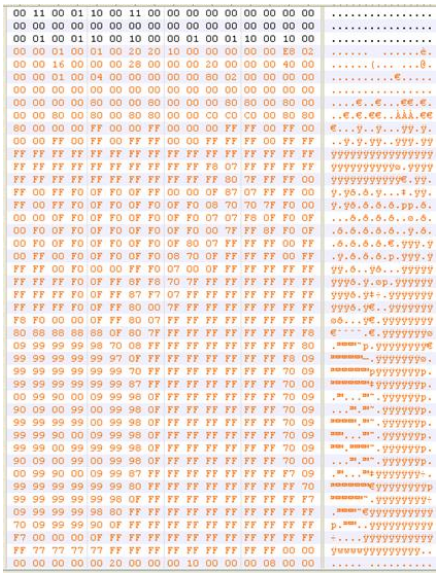


图 **4.3**

接着对于 GROUP_ICON 的字段，复制 icon 图标的前 20 字节，并把第 19 字节修改为 01：

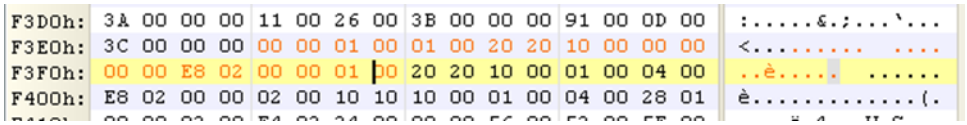


图 **4.4**

修改之后另存为文件 PEviewwhu，可以看见程序图标已经被修改

图 **4.5**

使用 eXeScope 对 Peview 进行汉化，如图 4.5 所示

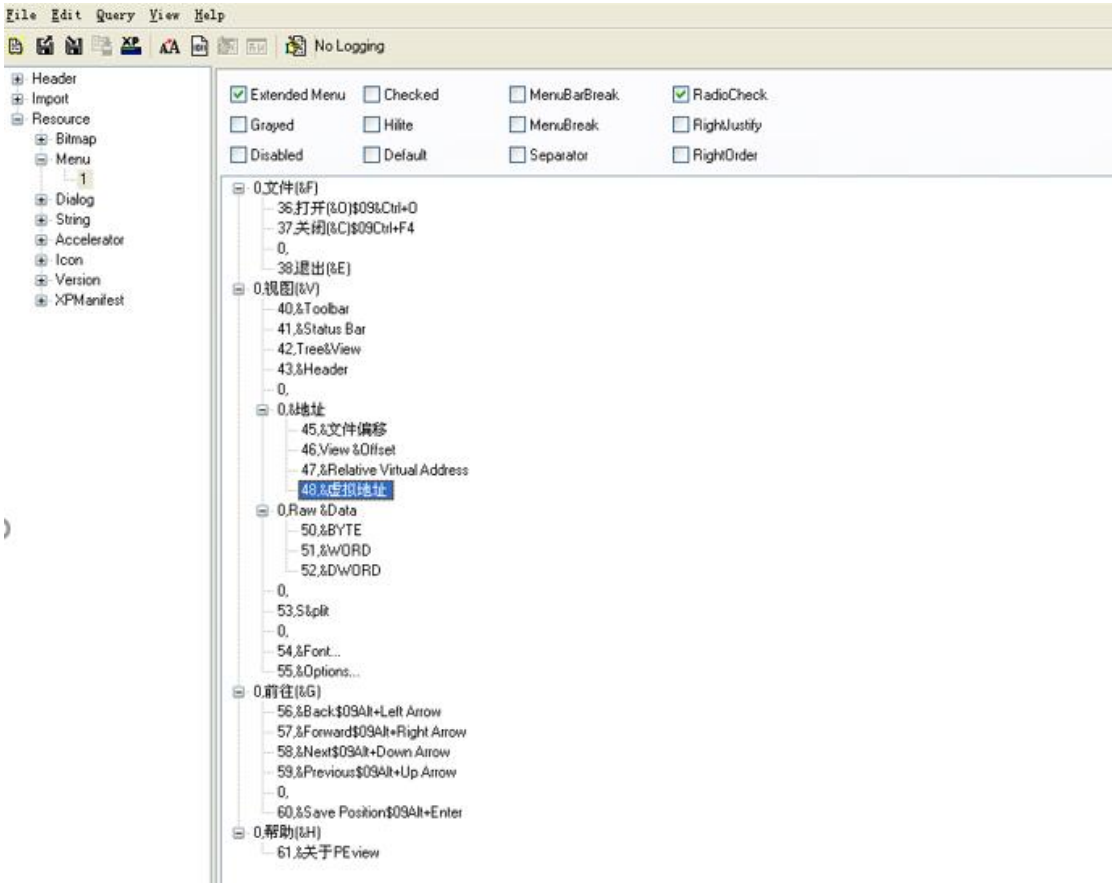


图 **4.5 程序汉化**

汉化效果

1.4.5 课后习题思考

打造最小 PE 文件的方法：

1、 合并 MZ 文件头和 PE 文件头
2、 删除 dos 桩和节表
3、 缩小文件的对齐粒度
4、 将代码节和数据节进行合并
5、 将 IAT 和 INT 进行合并

关于文章 tinyPE http://www.phreedom.org/research/tinype/

文章开头指出最小的 PE 文件为 97 个字节，Windows 2000 上可能的最小 PE 文件为 133 字节。二进制文件的很大一部分是由 C 运行时库组成的。如果我们用 NODEFAULTLIB 选项链接相同的程序，我们将得到一个更小的输出文件。我们还将通过将子系统设置为 Win32 GUI 从程序中删除控制台窗口。同时 O1 编译器选项还将根据大小优化代码节。

减少文件对齐粒度，如果查看 1024 字节文件的 dumpbin 输出，将看到文件对齐设置为 512 字节。代码节部分的内容从文件中的偏移 0x200 开始。头和代码节部分之间的空间用 0 填充。官方的 PE 规范规定最小 im 文件对齐为 512，但是 Microsoft linker 可以生成更小对齐的 PE 文件。Windows 加载程序将忽略无效的对齐方式，并能够执行文件。

简化汇编代码并删除 DOS 桩。为了进一步缩小文件，我们需要能够编辑 PE 头中的所有字段。我们将对 468 字节的 C 程序进行反汇编，并将其转换为可以用 NASM 进行汇编的汇编源。我们将使用以下命令来构建我们的 PE 文件:我们将做的惟一更改是删除 DOS 桩，该存根将打印出此程序不能在 DOS 模式下运行的消息。PE 文件仍然需要一个 MZ 头，但是只使用了两个字段:e_magic 和 e_lfanew。我们可以用 0 填充 MZ 头的其余部分。类似地，PE 标头中还有许多其他未使用的字段，可以在不破坏程序的情况下修改它们。

折叠 MZ 头，MZ 头中的 e_lfanew 字段包含从文件开头开始的 PE 头的偏移量。通常 PE 标头在 MZ 标头和 DOS 存根之后开始，但是如果我们将 e_lfanew 设置为比 0x40 小的值，PE 标头将从 MZ 标头内部开始。这允许我们合并一些 MZ 和 PE 头的数据，并产生一个更小的文件。PE 标头不能从偏移量 0 开始，因为我们需要文件的前两个字节为“MZ”。根据 PE 规范，PE 标头必须在 8 字节边界上对齐，但是 Windows 加载程序只需要 4 字节对齐。这意味着 e_lfanew 的最小值是 4。

另外文章中还做了删除 Data Directory、折叠引入函数表等等操作。

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

利用 UltraEdit 打开需要操作的 PE 文件，并找到 PE 文件资源节对应的数据，如果是对资源进行替换的话，需要把资源拷贝到相应的位置，并对剩余的地方补零，并对 group 字段进行相应的修改，如果是提取的话只需要将相应的十六进制即可。

涉及到的关键 API 函数主要有：BeginUpdateResource(), UpdateResource(), EndUpdate Resource()。

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？

每个图标在 PE 文件资源节都对应则一个 ICON 资源，图标资源有多个是因为保证程序可以兼容不同的操作系统，每一个图标都对应一种或者一个版本的操作系统。可以进行全局替换或针对当前的操作系统进行替换

资源节与恶意代码有何关联

恶意代码可以对目标程序进行图标替换，但是对于病毒的隐蔽性来说，一般是不希望进行图标替换的，进行图标替换的例如熊猫烧香；另外还可以进行图标伪装，例如将 EXE 恶意程序的图标更改为文件夹图标或者 pdf 文档图标，在鱼叉攻击中比较常见。

除此之外，恶意代码，偶尔会把一个嵌入的程序或驱动放在资源节，在程序运行之前将嵌入可执行文件或者驱动提取出来。

什么是 HOOK？其与本章学习有何关系

Hook 技术又叫做钩子技术，在系统没有调用该函数之前，钩子程序就先捕获该消息，钩子函数先得到控制权，这时钩子函数既可以加工处理（改变）该函数的执行行为，还可以强制结束消息的传递。简单来说，就是把系统的程序拉出来变成我们自己执行代码片段。要实现钩子函数，有两个步骤：

1.利用系统内部提供的接口，通过实现该接口，然后注入进系统（特定场景下使用）

2.动态代理（使用所有场景）

Hook 技术可用于动态编写 HOOK 函数，并通过 HOOK 函数在系统调用后，根据系统、硬件、文件等的属性，来决定系统调用函数的用途。

1.5 实验体会和拓展思考

通过对 PE 文件结构的学习，使我理解到了 Windows 平台下 exe 文件的基本形式，对程序的功能结构以及在内存中的布局有一个更加清晰的认识。在实验中，对于 PE 文件内容的修改使我进一步掌握了各种二进制编辑工具的使用，而通过修改引入函数节的实验，使我认识到了 PE 文件装载函数和执行的过程，认识到了 DLL 动态连接库的本质和用途。在 OllyDbg 中修改代码也让我对代码节的相关内容有了更清晰的认识。最后的手工汉化和更换图标实验趣味性和实用性很强，平时有一些软件只有英文界面可以通过这个实验学到的知识尝试进行自我汉化，而修改文件图标则让我明白了熊猫烧香病毒的替换机制，也使我进一步了解了 PE 文件的资源构造。

在实验中，我遇到过不少问题，其中修改引入函数节，仅从 kernel32.dll 中引入 LoadLibrary 和 GetProcAddress 并弹框的实验难度比较高，我反复观看了 MOOC 视频教程并请教了同学最终完成实验。而汉化实验刚开始对于工具掌握的不熟练导致汉化的 PE 文件所有的按钮都消失了，仔细比对才发现自己输入的汉字位置不对。

总之，通过实验使我理解了比书本上介绍的更加丰富和深刻的知识，加深了自己对于 PE 文件的理解。也使得我为接下来学习病毒和文件感染打下了一定的准备知识基础