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： 作业1：最小PE压缩

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 12 月 4日

**目 录**

[1实验名称 1](#_Toc8733)

[2实验目的 1](#_Toc7191)

[3实验步骤及内容 1](#_Toc17452)

[（1）删除DOS头和DOS\_STUB结构中不重要数据。 1](#_Toc12628)

[（2）修改可选文件头的数据目录 2](#_Toc856)

[（3）修改节表内容。 3](#_Toc9159)

[（4）修改代码节的内容。 4](#_Toc5687)

[（5）修改函数调用内容。 5](#_Toc15907)

[（6）只读数据段重构。 6](#_Toc6488)

[（7）补充函数代码。 7](#_Toc23738)

[（8）修改参数 8](#_Toc27078)

[4.结果展示 9](#_Toc31359)

# 1实验名称

最小PE压缩

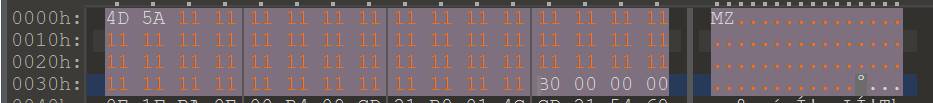
# 2实验目的

自己打造一个尽可能小的PE文件

# 3实验步骤及内容

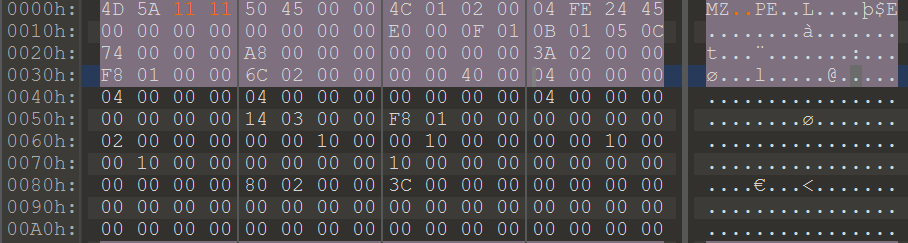
### （1）删除DOS头和DOS\_STUB结构中不重要数据。

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。

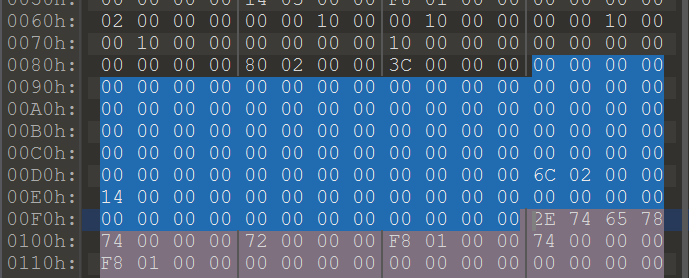
同时DOS\_STUB结构中的所有数据都是不重要的数据可以全部删除，将二者完全删除后的结果如下图所示。



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。

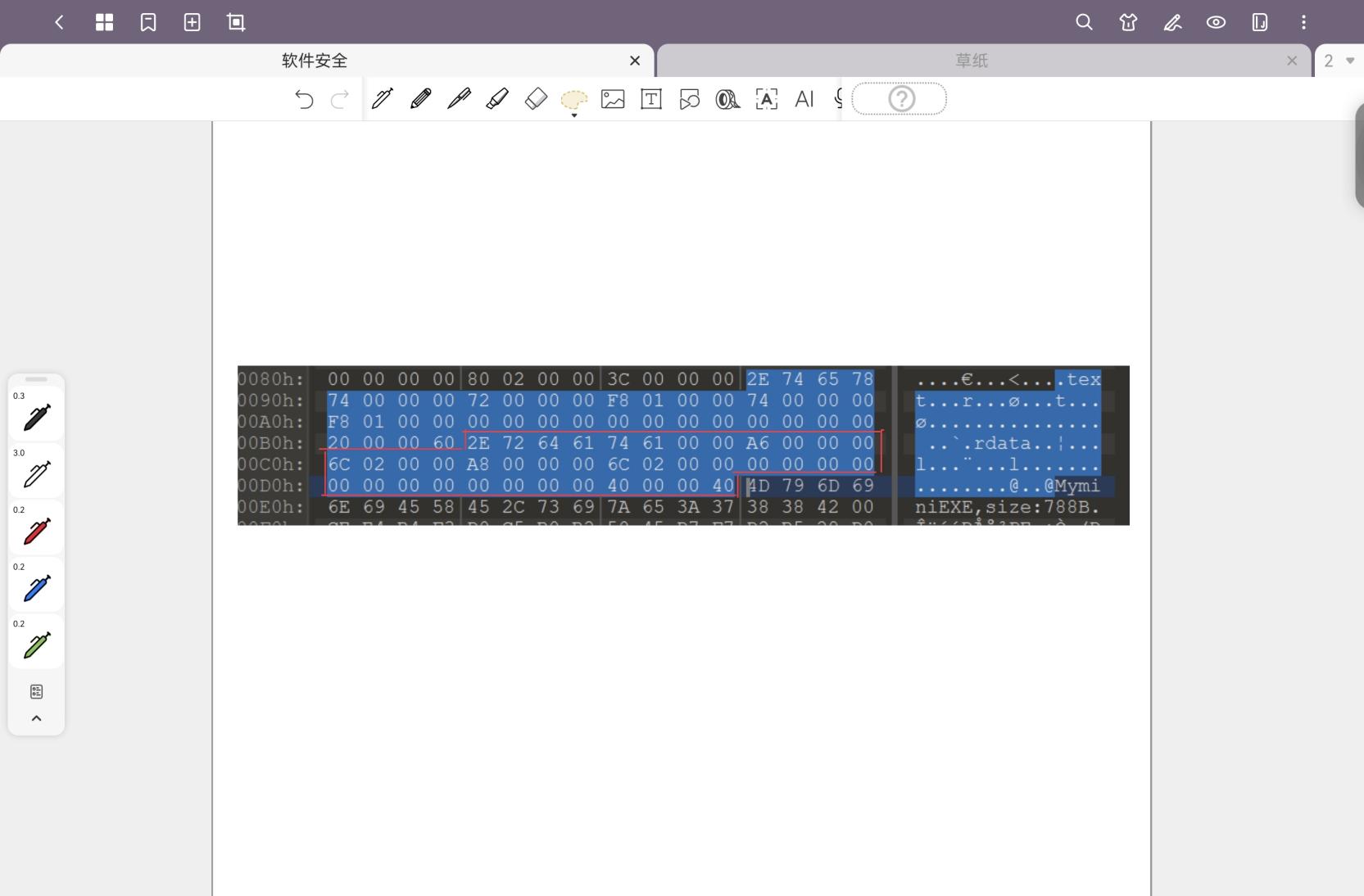


修改完成后的结果如下图所示。



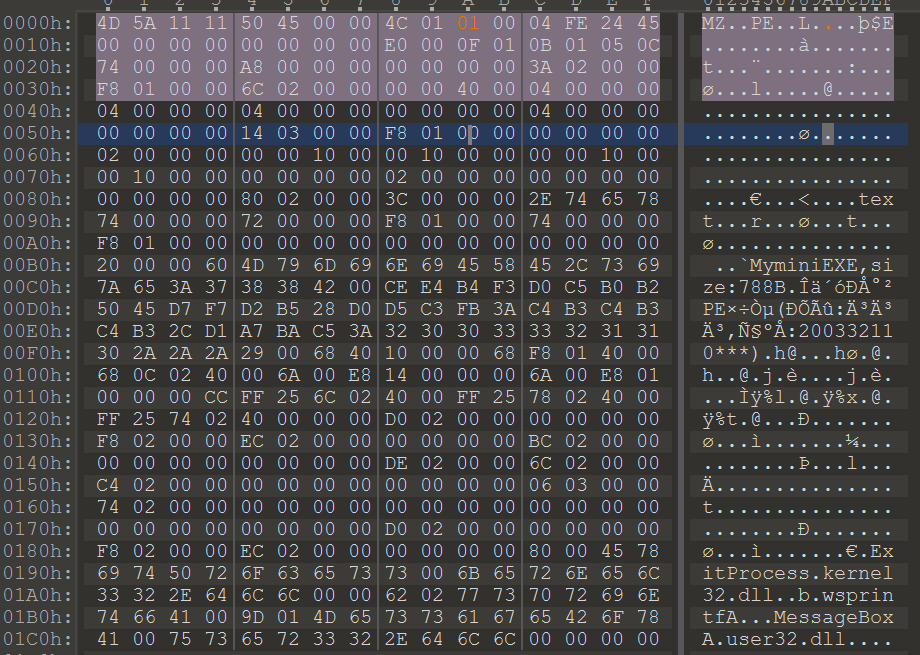
### （3）修改节表内容。

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。

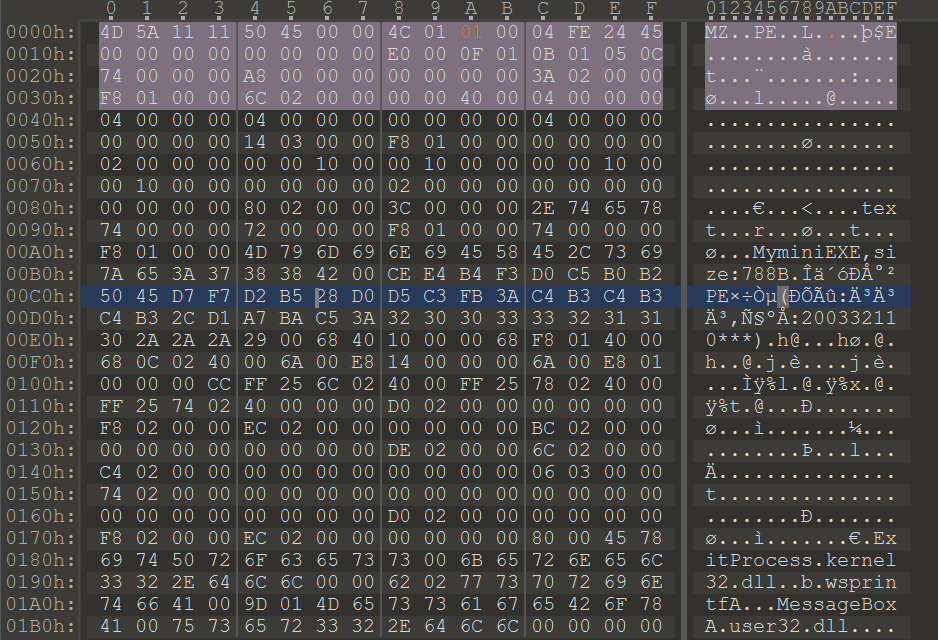


在本次实验中需要保留代码节的相关部分，将只读数据节的部分即上图中红框中的部分进行删除。

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。



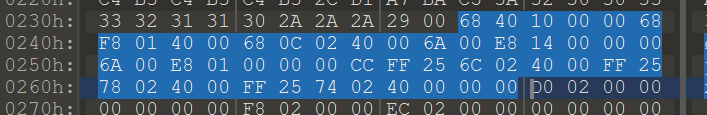
代码节中也存在有一些不重要的部分即代码节的一些属性，代码节的后16个字节。将其也删除后的结果如下图所示



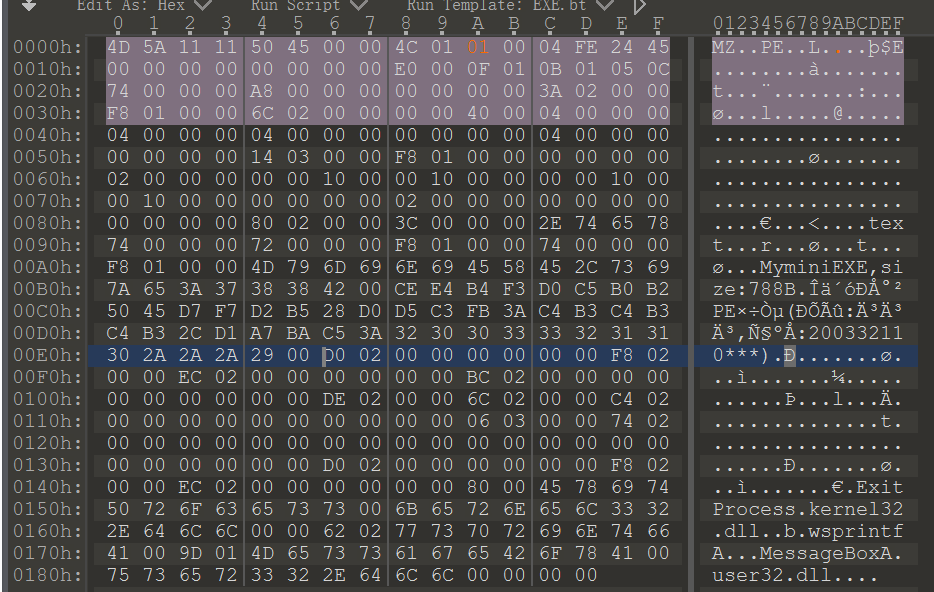
### （4）修改代码节的内容。

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。

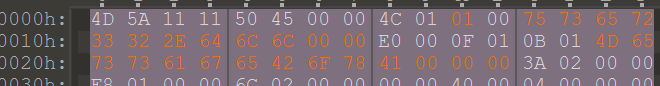


### （5）修改函数调用内容。

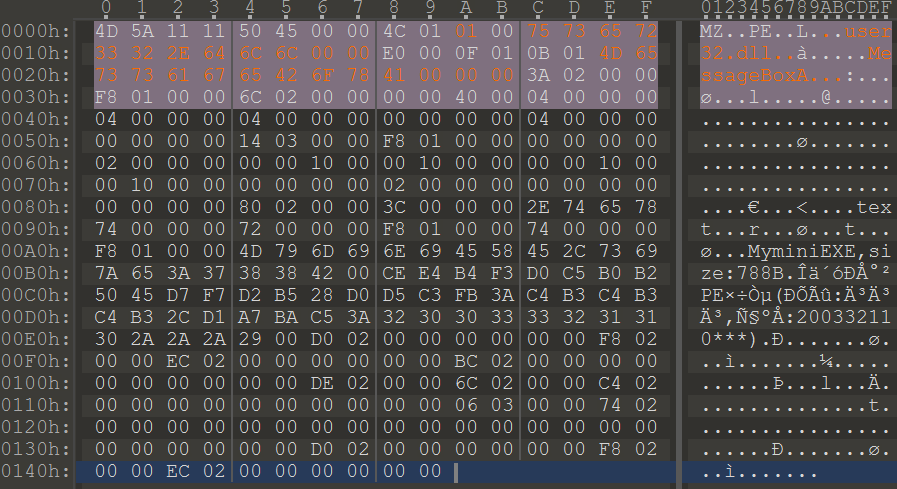
在之前的步骤中只是删掉了只读数据节的节表内容，在此要对只读数据节进行删除，首先只读数据节中保存有字符串“MessageBoxA”及其对应的dll“user32.dll”。

“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。



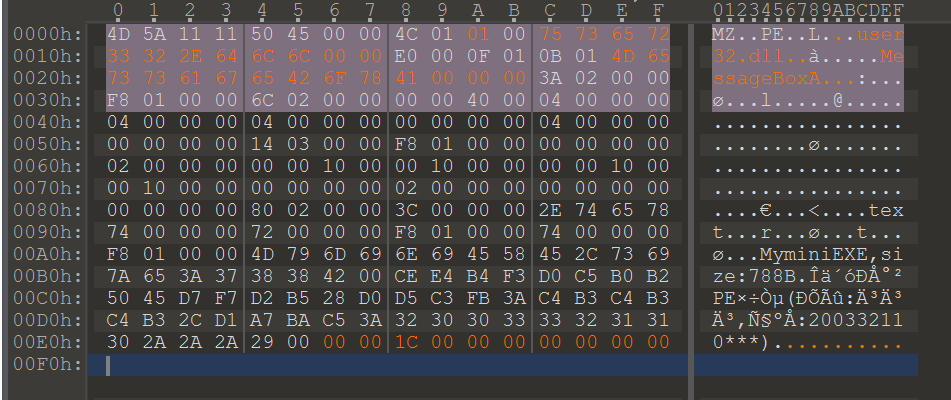
同时由于ExitProcess在函数运行过程中并不是必须的函数调用，所以在当前PE文件中将其调用的部分删除。删除后的整个PE文件如下图所示。



### （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。

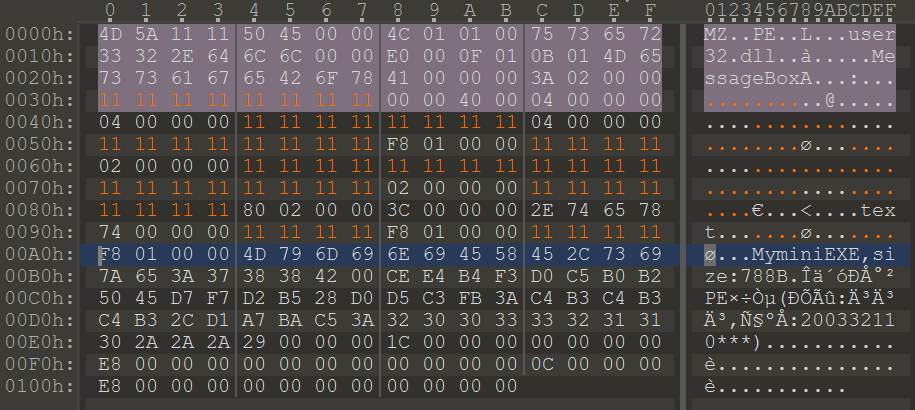
对于IAT表我们需要让其指向我们从user32.dll中调用的函数“MessageBoxA”因此我们将其设置为1C 00 00 00 00 00 00 00h。重构后的PE文件如下图所示。



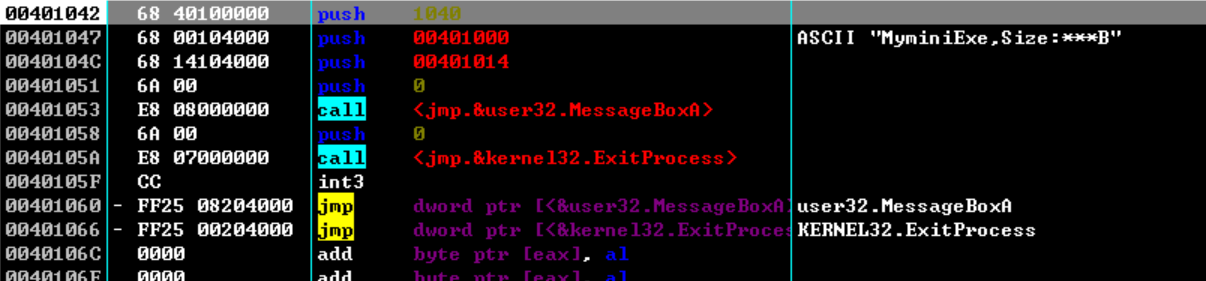
对于IDT表，我们需要让其OriginalFirstThunk变量和FirstThunk变量指向IAT的位置，即为E8 00 00 00h。将变量ForwarderChain变量指向第一个API传递器链表）即为0C 00 00 00H。最终将PE文件修改结果如下图所示。

### （7）补充函数代码。

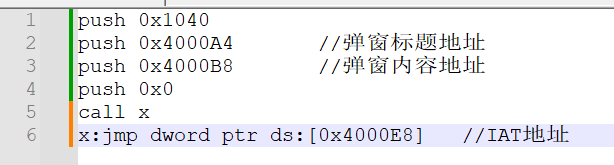
由于我们在第四步中将程序的代码删除，所以在此需要重新写程序的代码部分。首先我们先把当前PE文件中能够放置数据的不重要的位置标出来如下图所示。



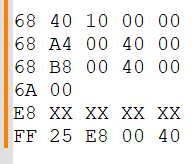
将原PE文件使用OllyICE打开后如下图所示。



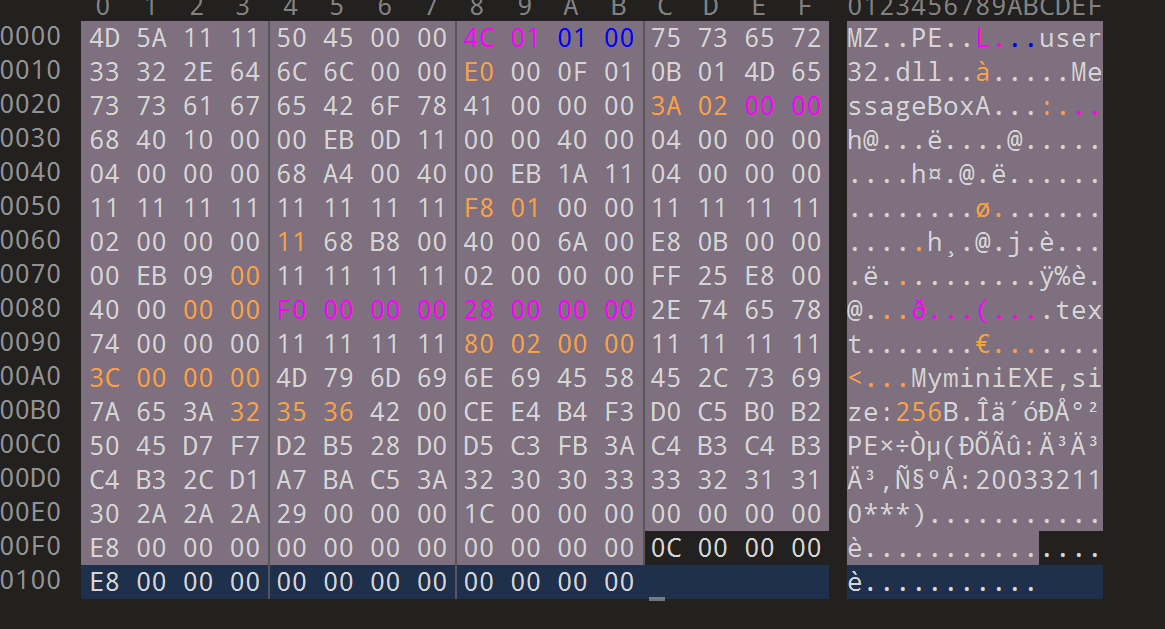
仿照原PE文件的汇编指令编写一段汇编指令如下图所示。



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。



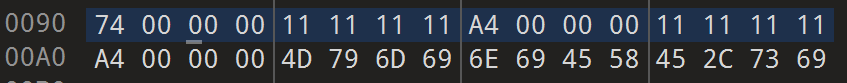
然后修改变量SizeOfHeaders指向第一个节开始的位置，在这里让其指向程序开始地址。



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。

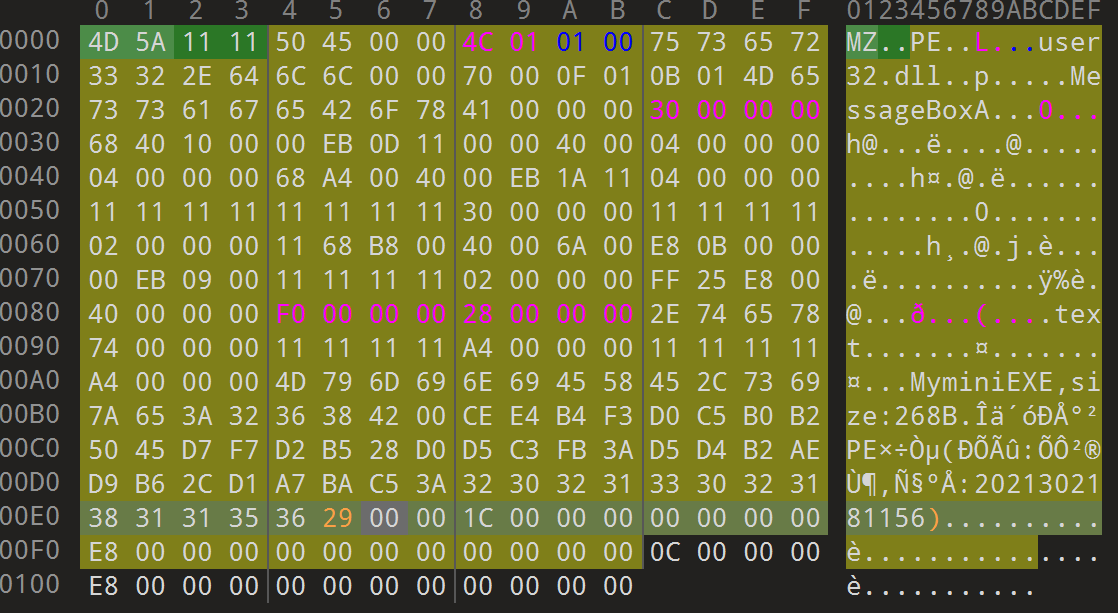


最后修改节表的VirtualAddress和PointerToRawData两个变量的值使其指向A4H位置。



# 4.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

