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： 实验三：PE病毒分析与清除

专业(班)： 信息安全-5班

学 号： 2022302181161

姓 名： 王亚鹏

课程名称： 软件安全实验

任课教师： 赵磊

2024年 10 月 24日

**目 录**

[实验3 PE病毒分析与清除(模板) 1](#_Toc181712199)

[1.1实验名称 1](#_Toc181712200)

[1.2实验目的 1](#_Toc181712201)

[1.3实验步骤及内容 1](#_Toc181712202)

[1.4实验关键过程、数据及其分析 3](#_Toc181712203)

[1.4.1熟悉MASM32 3](#_Toc181712204)

[1.4.2熟悉病毒重定位 5](#_Toc181712205)

[1.4.3 Kernel32基地址定位及API函数地址搜索 7](#_Toc181712206)

[1.4.4分析病毒感染过程 9](#_Toc181712207)

[1.5课后习题思考 11](#_Toc181712208)

[1.6实验体会和拓展思考 14](#_Toc181712209)

## 实验3 PE病毒分析与清除(模板)

### 1.1实验名称

PE病毒分析与清除

### 1.2实验目的

1. 了解PE病毒的基本原理
2. 熟悉PE病毒中的部分关键技术
3. 学会清除PE病毒

### 1.3实验步骤及内容

**第一阶段：熟悉Masm32**

* 编写helloworld程序
  + 下载masm32v11
  + 熟悉masm32的基本环境
  + 写一个最简单的helloworld程序，并编译成功
  + 对得到的可执行文件进行反汇编，比较其反汇编代码和最初的汇编代码有哪些异同
  + 查看并理解masm32\bin下各个批处理程序，了解他们的大致功能，以及与qeditor程序Project菜单的具体对应关系（Edit-Setting-Edit Menus）。
  + 探索其他菜单功能，并列出你认为对本课程后续学习有用的功能。

**第二阶段：熟悉病毒重定位的基本思路和方法**

* 在HelloWorld.exe中添加一段代码
  + 该段代码弹出一个对话框（标题：武大网安病毒重定位，内容：姓名+学号）
  + 该段代码同时包括代码和字符串数据
  + 该段代码可以插入到.text节的任意指令之间，而不修改该段代码中的任何字节，对其可移动性进行验证（移动产生的空闲区域可用nop代替）。

**第三阶段：****Kernel32基地址定位及API函数地址搜索**

* 搜索kernel32.dll的导出API函数地址。
  + 用ollydbg打开HelloWorld.exe，获取kernel32.dll模块基地址，定位到kernel32.dll模块。
  + 从内存中的kernel32.dll模块获取函数LoadLibraryA和GetProcessAddress的函数地址，并实际检验获得的地址是否正确

**第四阶段：分析病毒感染过程**

* 分析病毒
  + 编译教材中的感染例子程序-bookexample-old.rar，使用该感染例子对HelloWorld.exe进行感染。判断该病毒在感染文件时具体做了哪些操作，该病毒如何返回HOST。最后找出该病毒程序存在的一些问题，并解决这些问题。
  + 编译教材中的感染例子程序-bookexample-new.rar，使用该感染例子对计算器calc.exe进行感染。
  + 病毒感染例子程序在64位系统中无法正常感染，请定位其原因, 并以最小范围的源码修改方案解决该问题。
* 清除病毒
  + 在被感染的程序中定位HOST程序原程序入口地址，并手工恢复被感染的calc.exe。

**课后习题思考：**

* 对于Win10来说，传统的kerenl32基地址获取方法为何失效？对Win10下的在32位及64位程序来说，在通过PEB获取kernel32基地址的方法上有何差异？
* 尝试编写一个程序，可用来搜索指定目录下的所有exe文件，用MessageBox显示每一个被搜索到的exe文件名。
* 编写课本中病毒感染程序的病毒清除程序，其可以用来恢复被感染的任何文件。
* 病毒感染例子程序在64位系统中无法正常感染，请定位其原因并给出更多的修改方案。

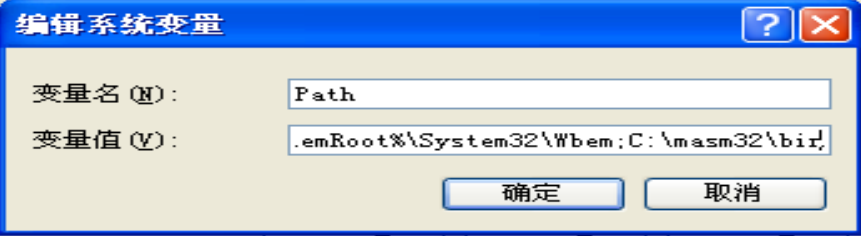
### 1.4实验关键过程、数据及其分析

#### 1.4.1熟悉MASM32

**masm32环境配置**

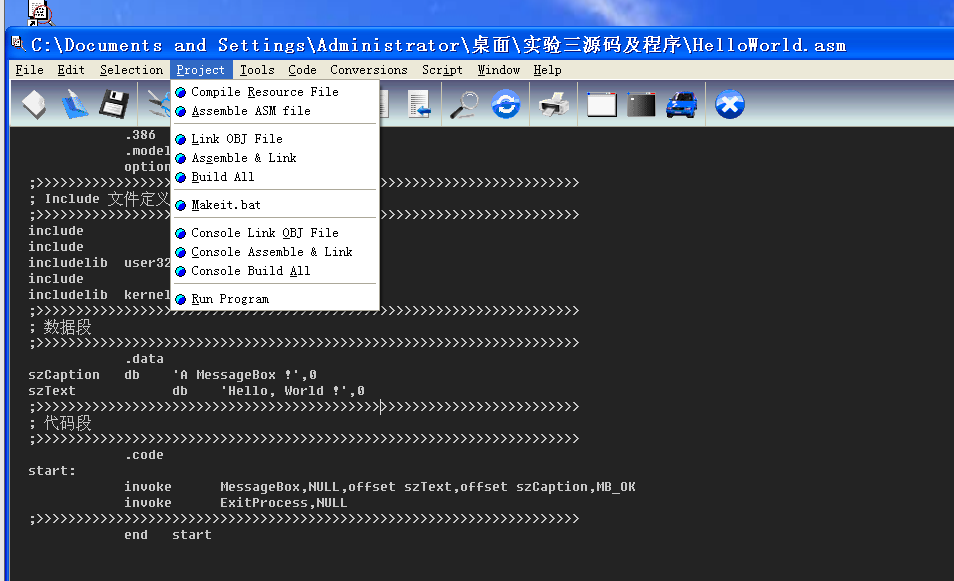
实验源码的压缩包中附带masm32的安装程序，先安装

安装完成之后打开我的电脑，找到masm32文件夹，之后需要修改环境变量



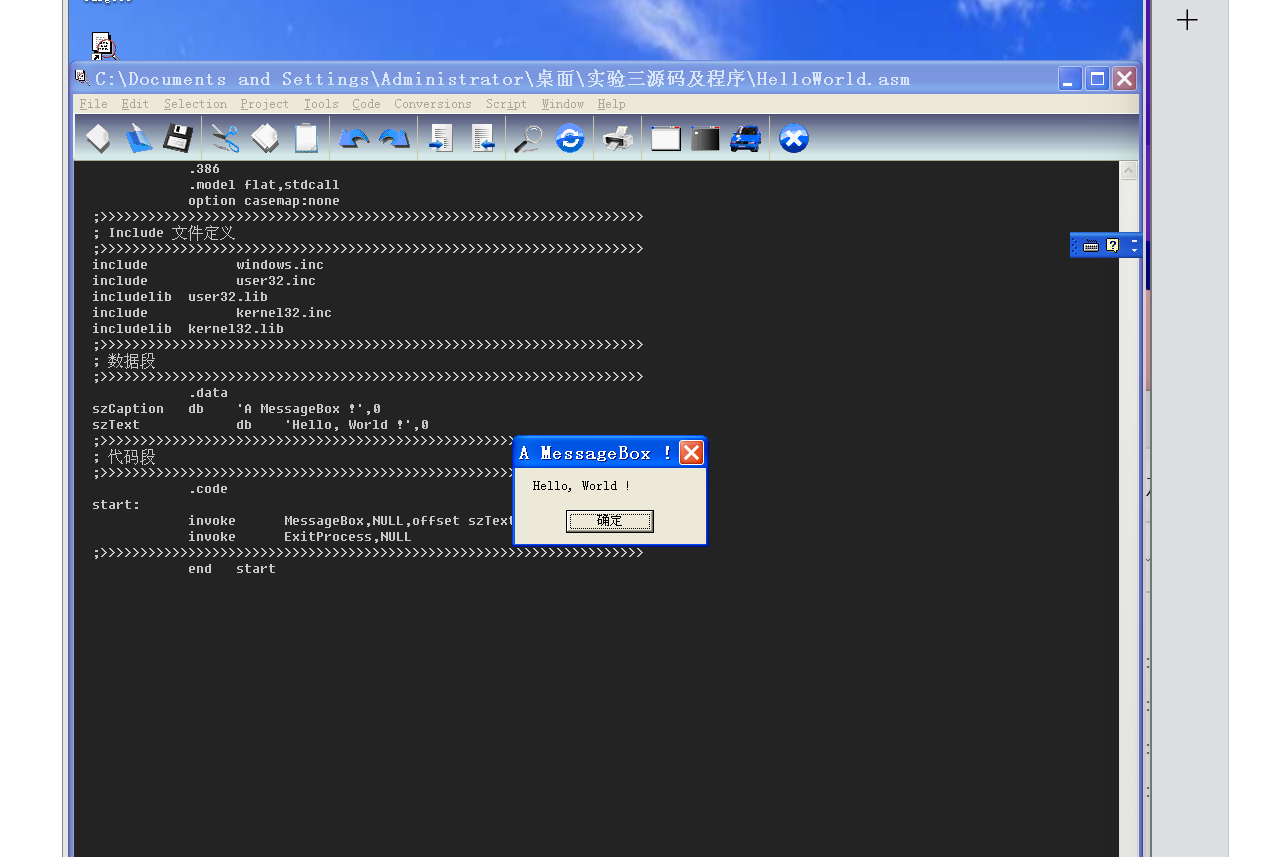
需要将bin，include，lib文件路径添加到环境变量之中

**编译链接hello.asm并进行反汇编**



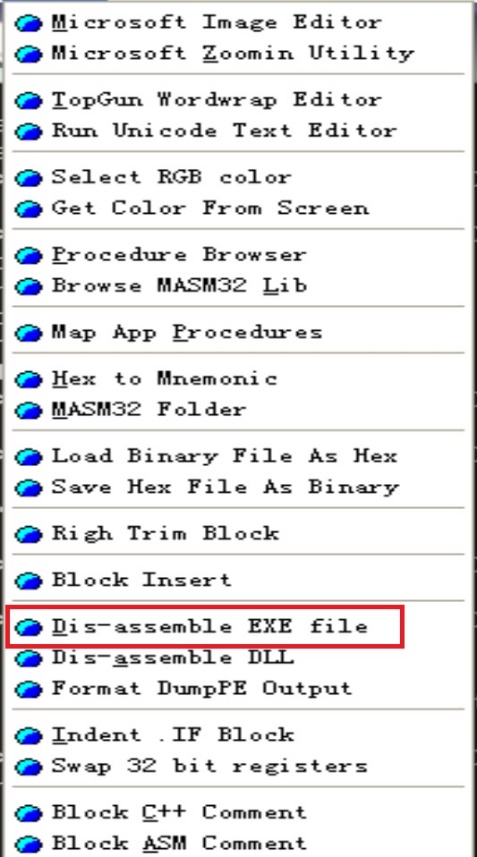
如图所示是masm32的基本功能界面，compile是编译，link是链接，assemble是汇编

打开hello.asm并编译链接后运行

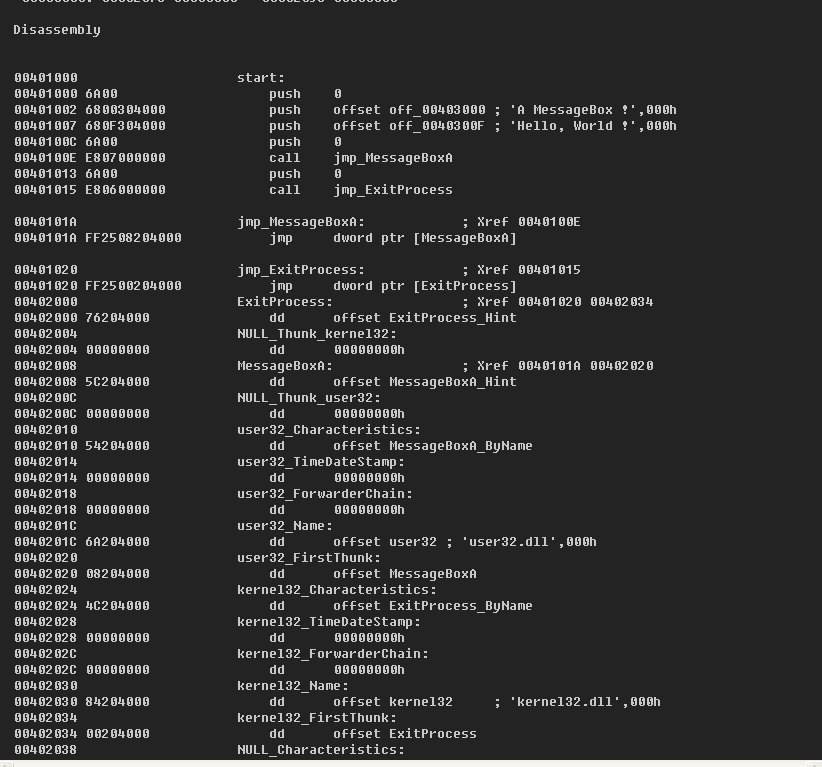


可以看到对话框正常弹出，运行正常

接下来进行反汇编操作

****

红色部分即是反汇编功能所在



如图所示反汇编的结果，我们可以看到相比于hello.asm多了很多东西，这就是在编译链接过程中重定位、导入函数的相关信息，还有其他的编译相关信息

#### 1.4.2熟悉病毒重定位

**实验原理**：

当我们call一条指令的时候，会将下一条指令压栈，然后在jmp到我们需要跳转的地址。call指令在编译的时候，编译器编译的是相对偏移，call指令调用的地址＝call指令所处偏移+5（因为call指令占五个字节）+相对偏移

**将不需要的内容用NOP填充**

这里需要填充一个稍微大一点的区域，方便我们进行数据写入

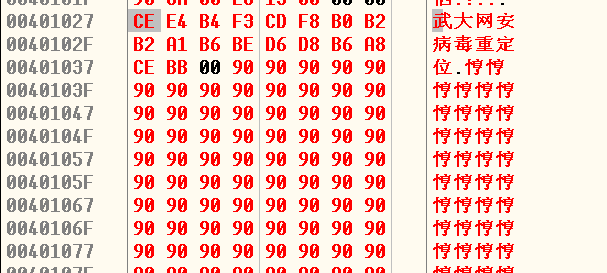
**数据填充**

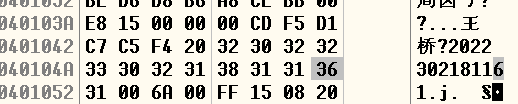
在填充的第一条nop指令处进行反汇编，加入push MessageBoxA，由此我们开始一个新的弹出窗口的编写

push的第二个参数是武大网安病毒重定位，共9个汉字，所以共18字节+0共19个字节

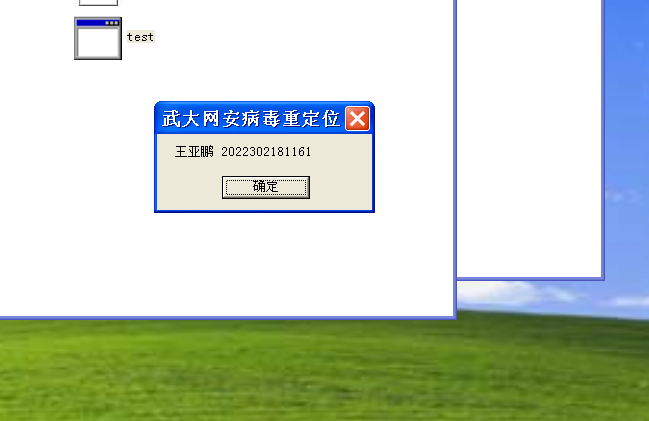
****

接下来的姓名加学号也是用这种方法填充，我的是“王亚鹏 2022302181161”一共是20个字节，还要填充末尾的0，最后结果如下：





不知道为什么名字变成了乱码，好在运行结果不变

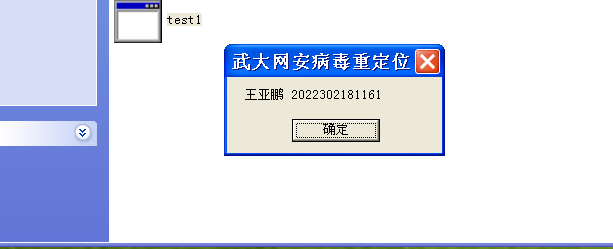


正常打印输出

**代码位置更改**

只需要将代码整块进行位移即可，观察这个新的程序能否有同样的弹窗

对于改变后的test1.exe而言，执行结果不变



使用用messagebox，如果我们使用相对偏移的话，重定位后就会失效，因此我们要使用messagebox的IAT

我们调用的弹窗都是这样的，所以可以任意移动

#### 1.4.3 Kernel32基地址定位及API函数地址搜索

**实验原理：**

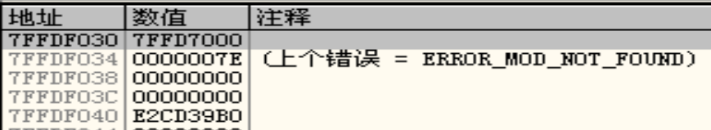
在NT内核系统中FS寄存器指向TEB结构（Thread Environment Block，线程环境块）。

TEB+0x30处指向PEB结构，PEB+0x0c处指向PEB\_LDR\_DATA结构。

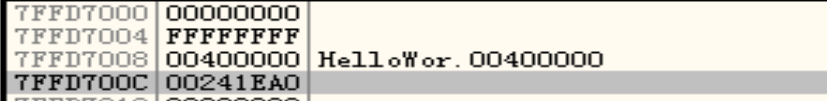
PEB\_LDR\_DATA+0x1c处是InInitialzationOrderModuleList，指向LDR\_MODULE双向链表结构，存放了一些动态链接库地址，第一个指向ntdl.dll，第二个就是kernel32.dll

**首先找到FS寄存器所在的位置**

之后找到其0x30的偏移量

****

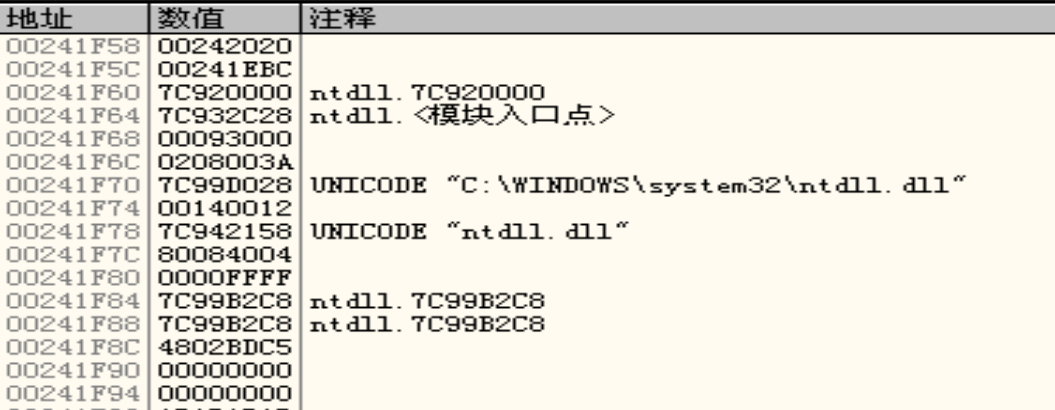
跟随，找到PEB结构，我们再找其0x0c的偏移量

****

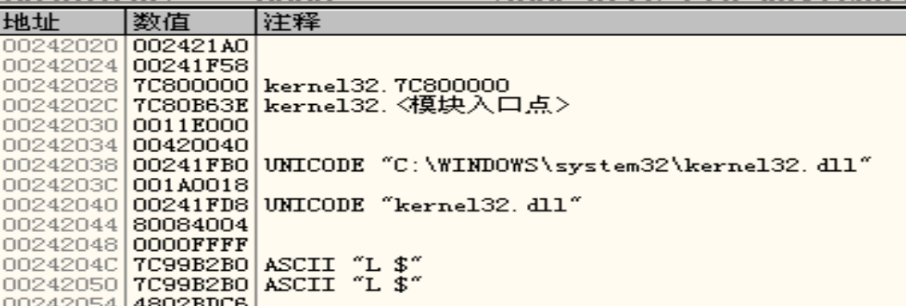
跟随，找到PEB\_LDR\_DATA结构，我们再找其0x1c的偏移量

****

跟随，找到InInitialzationOrderModuleList，指向LDR\_MODULE双向链表结构，存放了一些动态链接库地址。

****

整个双向链表的第一个连接模块是ntdll，指向7C92000，然后我们点击第一个值，去找第二个指针指向的kernel32dll

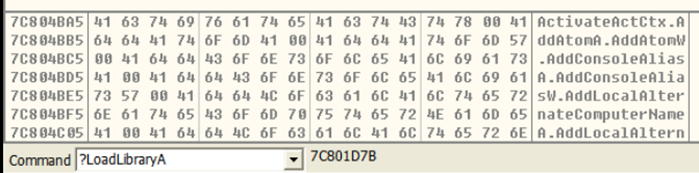
****

这里就找到了kernel32.dll

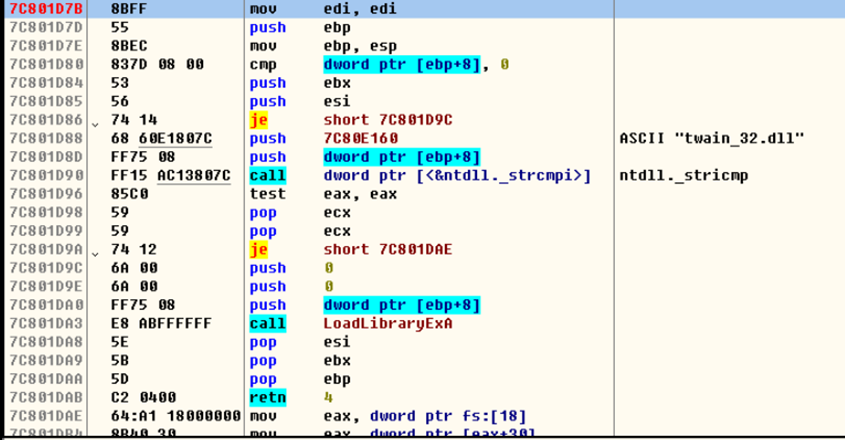
之后要进入导出表的偏移量为20的地方

****

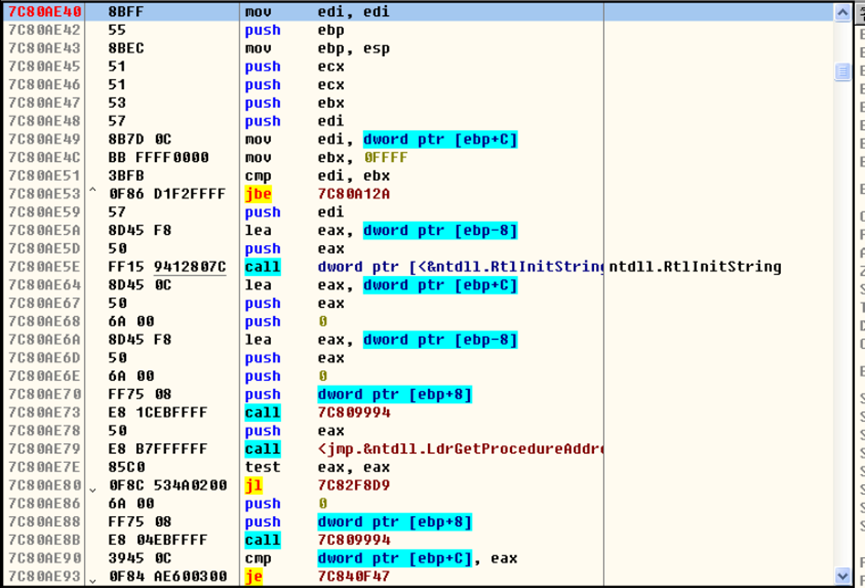
找到后还要跳转去验证，这里跳转后先进行搜索

****

再次进行验证，可以看到函数LoadLibraryA的内容

****

接着寻找GetProcAddress的函数地址

****

至此完成任务

#### 1.4.4分析病毒感染过程

**分析病毒**

由PPT内容所言，病毒感染程序的步骤如下：

1. 先定位kernel32基址。

2. 定位相关API函数地址。

3. 打开感染的文件，获得文件大小，创建映射文件，得到映射对象句柄。

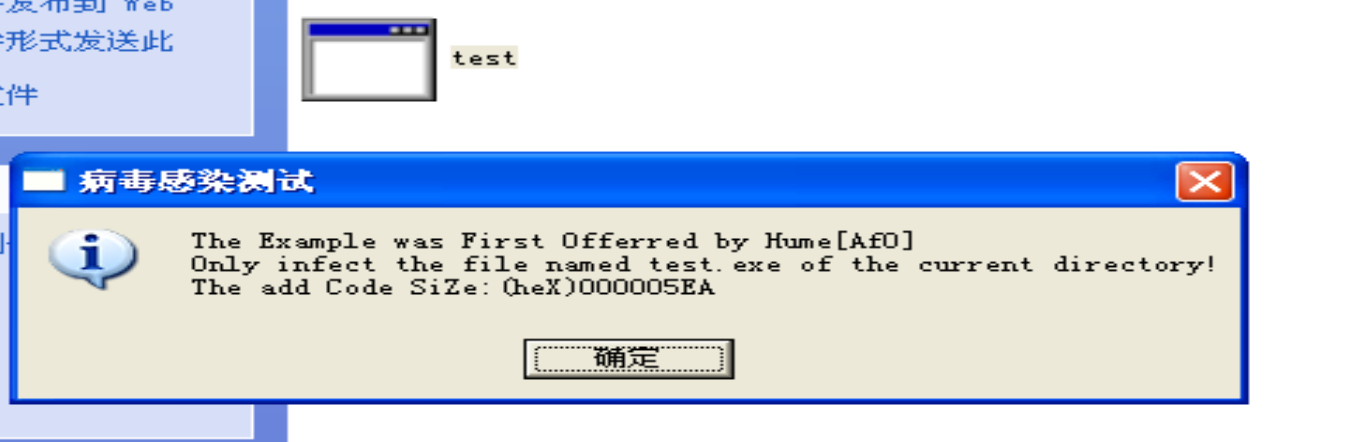
4. 判断MZ PE标志位确认为PE文件，判断感染标志dark，若已被感染则跳出执行HOST程序，否则继续。

5. 获得Directory的个数，定位section-header的起始位置，在最后一个节区头末尾新添加.hum节区头。

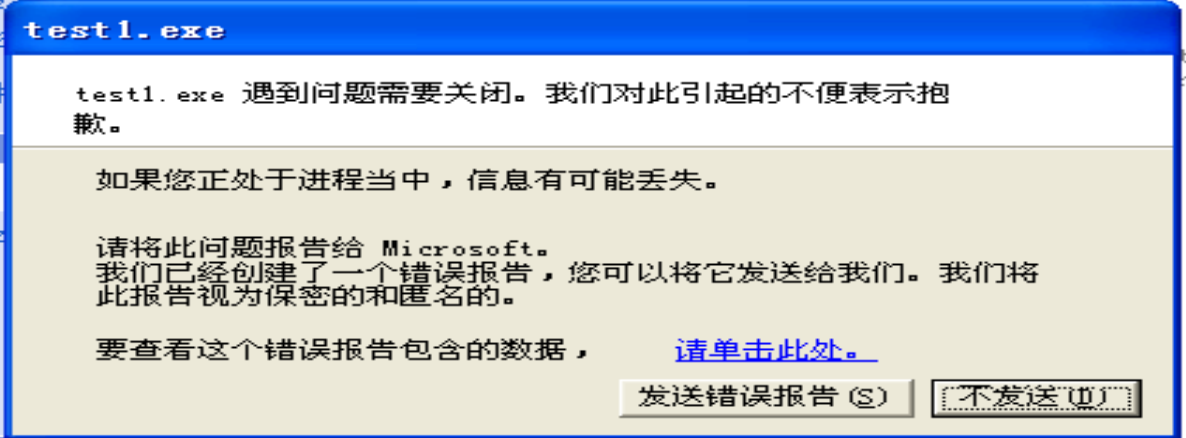
6. 保存新的入口点（返回HOST），修改入口点指向病毒代码起始位置。

7. 写入感染标志，写入病毒代码到.hum节。

旧病毒感染程序bookexample-old

****

要执行的时候会出现以下错误：

****

老程序用一个变量EIP保存原来的入口点，但是当病毒需要感染第三个程序的时候，call时保存的不是host的入口点，而是计算器（即被感染程序）的入口点，所以无法返回。

eip应该保存该程序的原入口点地址，对HelloWorld程序时1000H，但是此时保存的值为12475H，而12475H正好是calc的入口地址，说明入口地址发生了修改。

Bookexample-new对此进行了修改，使得感染能够正常进行

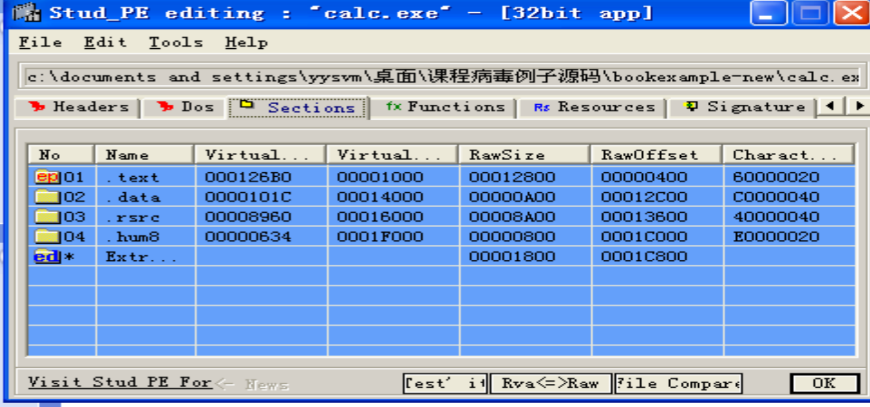
新增了一个变量来存放入口地址，感染结果如下（需要将被感染文件先改名为test



**清除病毒**

清理病毒即把入口地址改为原入口地址12475即可，这样我们就不会运行时先运行病毒弹窗

我们还要清理原有的病毒代码：

****

病毒写入了hum8，将这一部分清除

此时病毒被清理干净，程序可以正常执行



不会再出现第一个病毒感染弹窗

计算器清除病毒后也恢复原样

****

### 1.5课后习题思考

* 对于Win10来说，传统的kerenl32基地址获取方法为何失效？对Win10下的在32位及64位程序来说，在通过PEB获取kernel32基地址的方法上有何差异？

（1）地址空间布局随机化（ASLR）：自Vista开始，Windows引入了ASLR技术。ASLR会随机化包括kernel32.dll在内的系统库的基地址，因此，直接依赖于固定基地址的方法失效。在Win10中，kernel32的基地址在每次启动时都是随机的，无法使用传统方式进行获取

（2）系统保护机制的提升：Windows 10加强了对内存和进程的保护，使得直接读取PEB中的地址信息变得更加困难，某些操作甚至会被阻止

* 尝试编写一个程序，可用来搜索指定目录下的所有exe文件，用MessageBox显示每一个被搜索到的exe文件名

思路：通过Windows API函数FindFirstFileA来开始搜索.exe文件，并使用FindNextFileA继续查找目录中的下一个.exe文件，直到搜索完毕。每次找到一个文件，它的文件名会被保存到findFileData.cFileName

代码如下：

#include <windows.h>

#include <stdio.h>

void SearchAndDisplayExeFiles(const char\* directoryPath)

{

    WIN32\_FIND\_DATAA findFileData;

    char searchPath[MAX\_PATH];

    snprintf(searchPath, MAX\_PATH, "%s\\\*.exe", directoryPath);  // 构建搜索路径

    HANDLE hFind = FindFirstFileA(searchPath, &findFileData);

    if (hFind == INVALID\_HANDLE\_VALUE)

    {

        MessageBoxA(NULL, "No exe files found!", "Search Result", MB\_OK);

        return;

    }

    else

    {

        // 初始化一个字符串用于存储所有exe文件的名称

        char allExeFiles[4096] = "";  // 假设4096字节足够存储所有文件名

        do

        {

            // 将找到的exe文件名添加到allExeFiles中，并换行

            strcat(allExeFiles, findFileData.cFileName);

            strcat(allExeFiles, "\n");

        } while (FindNextFileA(hFind, &findFileData) != 0);

        FindClose(hFind);

        // 显示所有文件名

        MessageBoxA(NULL, allExeFiles, "Found exe files", MB\_OK);

    }

}

int main()

{

    // 设置要搜索的目录路径为 E:\.vscode\vscode\code

    const char\* directory = "E:\\.vscode\\vscode\\code";

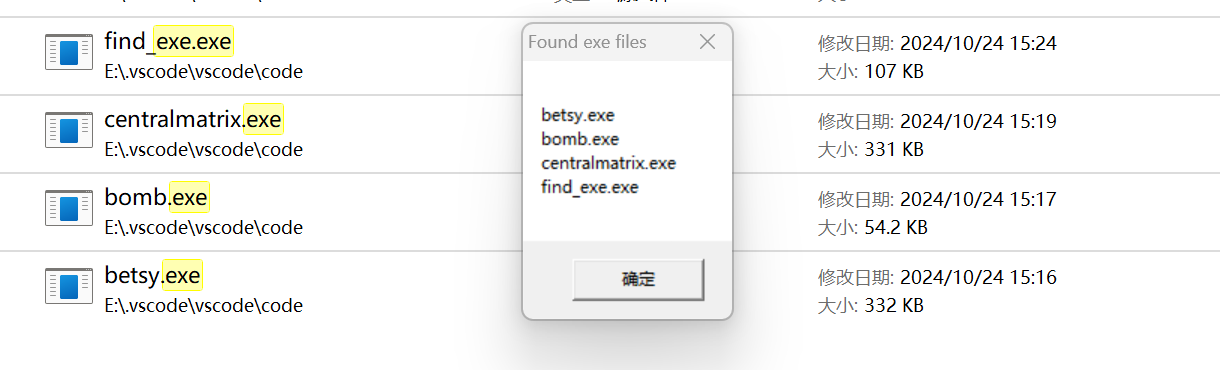
    SearchAndDisplayExeFiles(directory);

    system("pause");

    return 0;

}

运行结果如下：



可以看到运行结果正确无误

* 编写课本中病毒感染程序的病毒清除程序，其可以用来恢复被感染的任何文件

参见实验步骤中的相关内容，已经清除了病毒，如果要写成程序自动化执行病毒清除任务，可以按照以下代码（需要自定义一些内容）

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// 假设函数来查找病毒签名并清除病毒代码

void clean\_virus(FILE \*file) {

    // 定位病毒签名

    fseek(file, 0, SEEK\_SET); // 从文件头开始扫描

    unsigned char buffer[4096]; // 假设文件足够小

    fread(buffer, sizeof(buffer), 1, file);

    // 查找病毒签名，并清除代码

    for (int i = 0; i < sizeof(buffer); i++) {

        if (memcmp(buffer + i, "HUM8\_SIG", 8) == 0) { // 假设病毒签名为 "HUM8\_SIG"

            memset(buffer + i, 0, 8); // 清除病毒代码

            fseek(file, i, SEEK\_SET);

            fwrite(buffer + i, 8, 1, file); // 写回去

            break;

        }

    }

}

// 改回入口地址的函数

void reset\_entry\_point(FILE \*file) {

    fseek(file, 0x18, SEEK\_SET); // 假设入口地址在文件偏移0x18

    unsigned int entry\_address = 12457;

    fwrite(&entry\_address, sizeof(entry\_address), 1, file);

}

int main(int argc, char \*argv[]) {

    if (argc != 2) {

        printf("用法: %s <文件路径>\n", argv[0]);

        return 1;

    }

    FILE \*file = fopen(argv[1], "r+b");

    if (!file) {

        perror("无法打开文件");

        return 1;

    }

    clean\_virus(file);

    reset\_entry\_point(file);

    fclose(file);

    printf("病毒清除完毕,入口地址已恢复到12457\n");

    return 0;

}

* 病毒感染例子程序在64位系统中无法正常感染，请定位其原因并给出更多的修改方案

**首先分析无法正常运行的原因：**

* **地址空间的差异**：64位系统中的地址空间远大于32位系统，这会导致一些硬编码的地址或大小在64位系统中失效。
* **API调用的差异**：病毒程序可能使用了一些32位API，而在64位系统中这些API的调用方式或返回结果可能不同。
* **汇编指令的差异**：病毒程序中使用的汇编代码可能依赖于32位指令集，在64位系统中这些指令无法正常工作，特别是涉及寄存器的操作时。

**修改方案：**

1. **修改汇编代码**：确保病毒程序中的汇编代码适用于64位环境，调整寄存器和数据指令的宽度。例如，将32位寄存器（如EAX）更改为64位寄存器（如RAX）。
2. **调整API调用**：在64位系统上，确保使用64位版本的API。如果使用的是内联汇编或调用栈，需要确保参数传递方式适应64位标准。
3. **内存布局的动态适应**：由于ASLR的存在，确保代码中没有硬编码的内存地址，使用动态方式获取函数地址和基地址

### 1.6实验体会和拓展思考

在本次PE病毒分析与清除的实验中，通过深入学习和实践，逐步掌握了PE病毒的基本原理与其关键技术，尤其是在病毒重定位与内存操作方面

**1. PE病毒重定位的理解**

在实践中，病毒的重定位思想让我对PE结构和病毒如何在程序中嵌入恶意代码有了深刻的认识。通过对HelloWorld.exe的实验，理解了如何在PE文件的.text段中插入额外的代码片段，而不影响原有程序的执行。这段插入代码的移动性和不破坏性验证了重定位技术的有效性。

**2. Kernel32基地址定位及API函数地址搜索**

在Kernel32基地址定位阶段，我使用了Ollydbg工具打开并分析HelloWorld.exe。通过工具，我成功获取了内存中kernel32.dll的基地址，并进一步定位了API函数如LoadLibraryA和GetProcAddress的具体地址。通过这个过程，我对Windows程序中API函数的内存加载机制和地址定位有了更深入的理解，也熟悉了如何从内存中提取这些关键信息。这部分实验强化了我对Windows内存管理和分析工具（如Ollydbg）的使用技能。

**3. 病毒感染过程的分析**

在实验的病毒感染过程分析部分，编译并使用了实验提供的感染例子程序，成功感染了HelloWorld.exe和系统计算器程序calc.exe。通过详细分析感染过程，我逐步理解了病毒在感染文件时是如何修改文件头、插入恶意代码，并在运行时返回HOST（宿主程序）的整个操作流程。这让我认识到病毒通过修改PE文件结构来操控程序执行的原理，并让我掌握了如何识别并分析这些感染过程中的关键环节。

**4. 课下深入理解与挑战**

在64位系统上的病毒感染实验中，我采用了一种较为粗暴的方式来修改代码以适配64位系统。例如，在寻找函数地址的过程中，我通过调试一步步发现地址的大致位置，并手动调整代码来加快程序定位这些地址的过程。这部分的工作还可以进一步优化，尤其是在如何更高效地找到API函数的地址上