电子科技大学

计算机科学与工程学院

标 准 实 验 报 告

（实验）课程名称 信息对抗综合设计实验 II

电子科技大学教务处制表

电 子 科 技 大 学

实 验 报 告

学生姓名：刘芷溢　　学 号：2020080907009

指导教师：李忻洋

一、实验项目名称： 机器码探索与病毒初见

二、实验目的：

通过调试器，自己设计实验分析机器码的格式，并设计实验修改机器码，完成实证。本实验包含对 x86 指令集中比较简单的 mov 指令的分析，并在此基础上分析在函数调用和二进制安全领域有较大作用的 call 指令。根据修改赋值语句机器码的执行，我们通过调试器实现代码补丁的方式修改程序的流程，完成病毒机制中获取执行权限的过程，在内存中模拟病毒执行过程，在主函数跳转到病毒函数接着跳回主函数实现原函数的正常执行。

三、实验原理：

1. 运用调试器，不修改源码而观察目标程序。掌握 Visual Studio

调试器的监视变量功能，内存查看功能，代码转变成反汇编形式功能。

2. 整数在电脑上的表示有大小端机之分，本实验的机器为小端机，逻辑上的低字节放在物理上的低字节。

3. MOV、JMP 指令的机器码表示，更改机器码实现指令跳转以及修改变量的值。

例如：MOV 指令实现赋值

int gi = 12;

0041138E　　C7 05 40 71 41 00 0C 00 00 00

mov dword ptr ds:[00417140h],0Ch

4. Win7 及以上系统中使用 VS 时，由于 ASLR 地址随机化机制的影响，程序加载到内存的基地址可能发生变化。每次调试变量的地址可能不相同；为避免重定位表带来的影响，禁止随机化属性。

5. 病毒执行过程特点：窃取执行权，将执行流程从正常执行的指令修改为执行病毒指令；归还执行权，病毒执行完后，恢复执行被感染对象正常的指令（潜伏）。本实验简化该过程，通过修改内存机器码模拟上述流程，忽略了执行程序文件格式的复杂因素，直指本问题的本质；正常执行流程：normalfunc，病毒执行流程：virusfunc。

5-1. 找到 normalfunc 入口的位置，利用 JMP 指令跳转到 virusfunc 入口处；

5-2. 找到 virusfunc 中的打印指令，在其后修改指令，填充内容为 normalfunc 中被第一步 JMP 指令覆盖的 3 条完整指令；

5-3. 在第 2 步填充的 3 条指令后填写 JMP 指令使其跳回第 1 步

覆盖的 3 条指令后，执行 normalfunc。

四、实验环境（设备、元器件）：

个人 PC 机

操作系统： Windows 10

应用软件： Visual Studio 2010

五、实验步骤：

任务一：修改机器码的执行

实验代码：

```
int g1,g2;
int _tmain(int argc, _TCHAR* argv[])
{
    g1 = 907009;
    return 0;
}
```

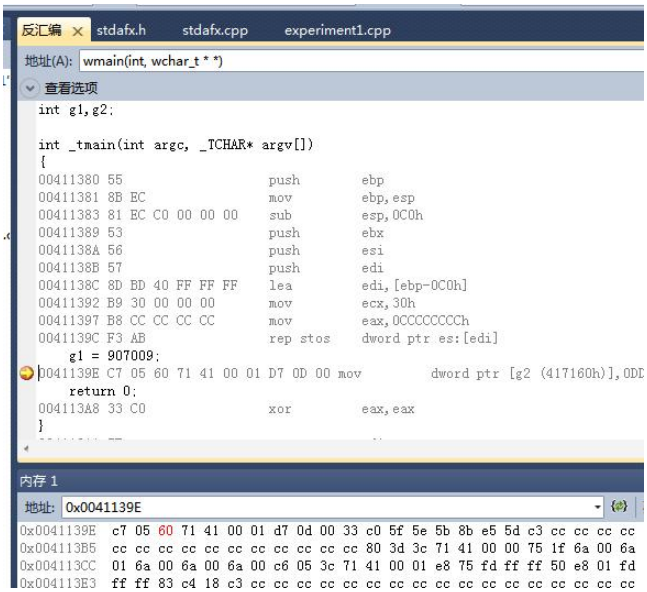
1. 定义两个局部变量 g1，g2；并对 g1 赋值成 907009

2. watch 窗口查看 g1，g2 的内存地址

3. 查看赋值语句对应的 MOV 指令机器码的指令地址，是否与 g1 的内存地址相符；

4. 修改 MOV 指令目的操作数为 g2 的地址（0x417160）；观察 g2 的

内容是否发生变化。



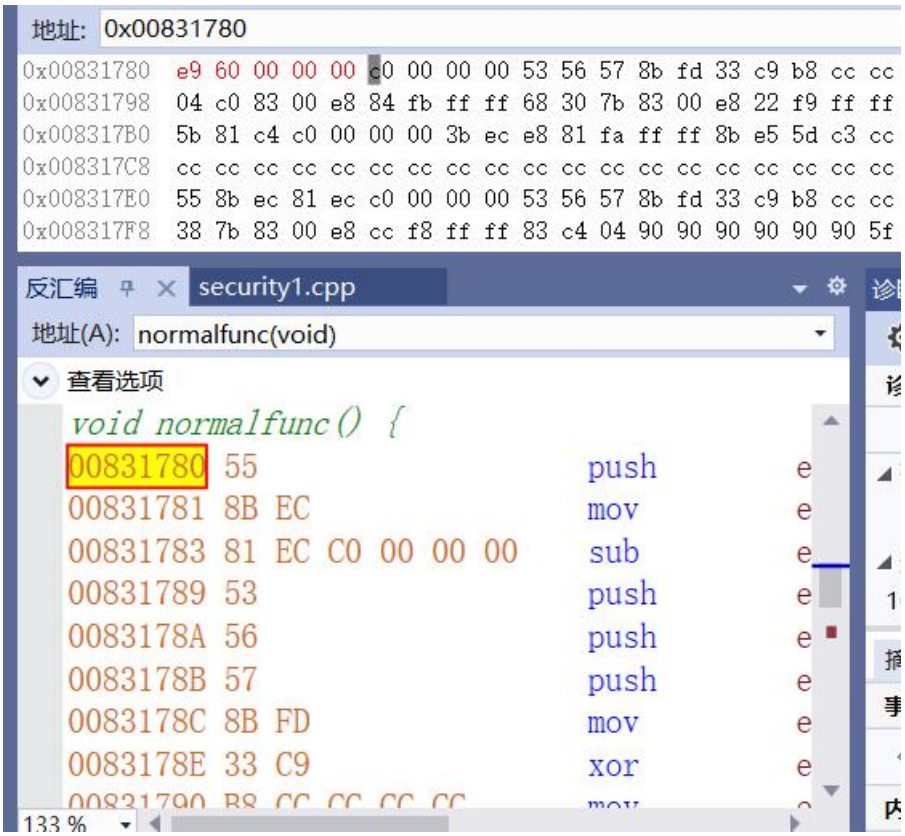
任务二：在内存和硬盘中实现病毒的基本流程

实验代码：

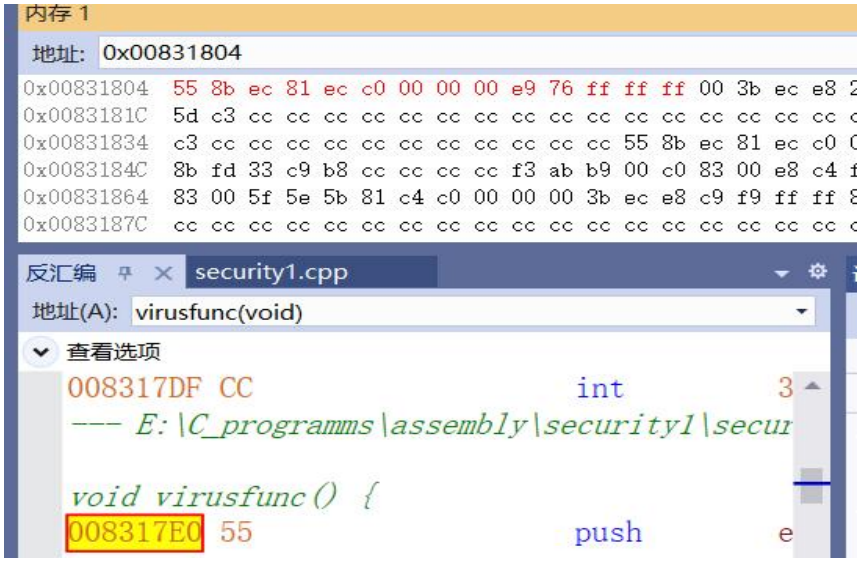
```
void normalfunc(){
    printf("Hello\n");
}
void virusfunc(){
    printf("virus:907009\n");
    _asm
    {
        nop;
        nop;
        nop;
        nop;
        nop;
```

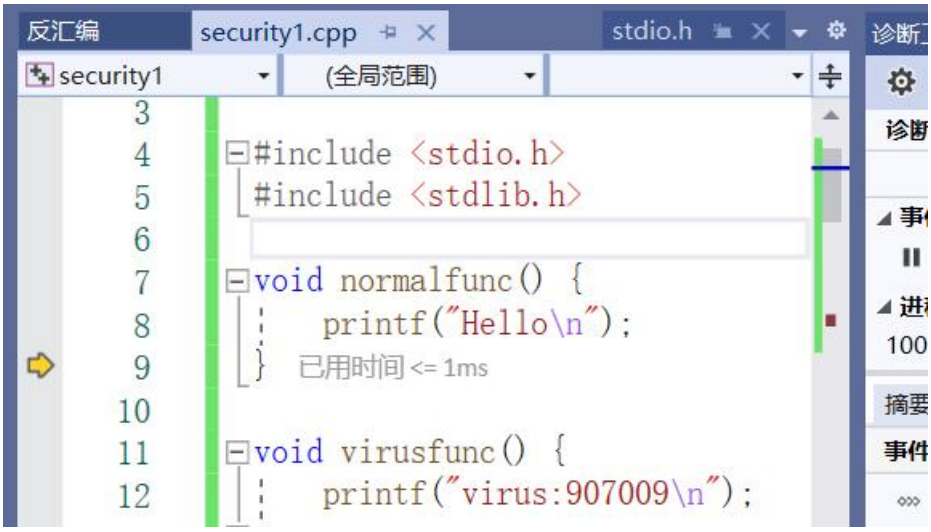
```
        nop;
    }
}
int _tmain(int argc, _TCHAR* argv[])
{
    normalfunc();
    system("pause");
    return 0;
}
```

1. 修改 normalfunc 的前三条指令，变成 JMP 指令，跳转到 virusfunc 函数开始处；跳转偏移量：0x8317E0 - 0x9E1780 = 60，小端机内存表示 60 00 00 00，最终 JMP 指令为 e9 60 00 00 00；



2. virusfunc 的 nop 指令处，进行对被覆盖字节的恢复，并跳回 normalfunc，需要恢复的字节码是：55 8b ec 81 ec c0 00 00；跳转会 normalfunc 的偏移量就是 0x831789 – 0x83180E- 5 = ffffff76，小端机内存表示 5e ff ff ff, 最终指令为 e9 76 ff ff ff；如果成功进行会打印两个字符串。



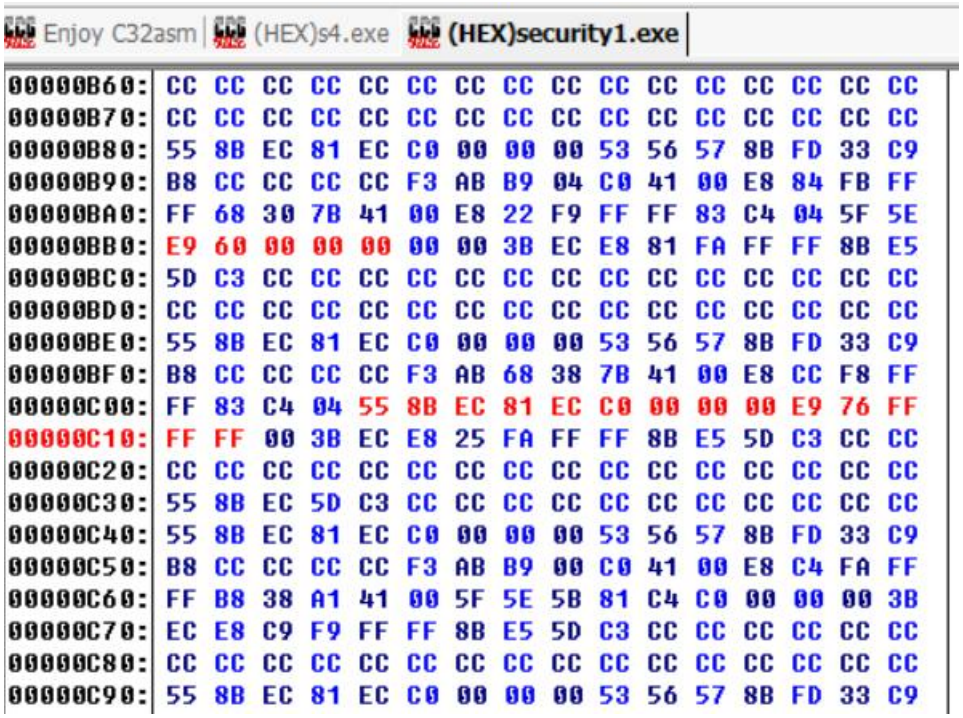


在文件中进行指令的修改：

1. 使用 C32ASM 工具以 16 进制编辑方式打开目标 exe；

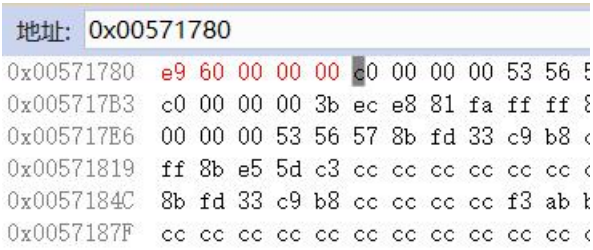
2. Ctrl+F，以 16 进制方式搜索 E8 22 F9 FF FF；搜索到后，修改为 JMP 指令 e9 60 00 00 00，然后保存；

3. 继续搜索 virusfunc 的 90 90 90 90 90 90，在后面恢复被覆盖指令 55 8b ec 81 ec c0 00 00 00；加上跳转到 normalfunc 的指令 e9 76 ff ff ff。

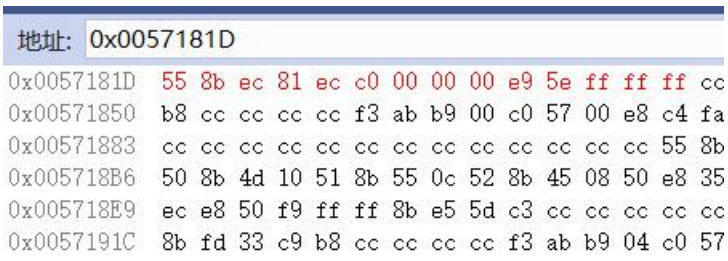


任务三：在内存中模拟病毒时，取消 nop，当篡改 normalFunc 跳转到 virusFunc 时，跳转到 virusFunc 的第一条指令然后在 virusFunc 最后一条 ret 语句处，进行对被覆盖字节的恢复，并跳回 normalFunc，看是否能顺利完成病毒。

1. 第一条跳转指令的偏移不变；

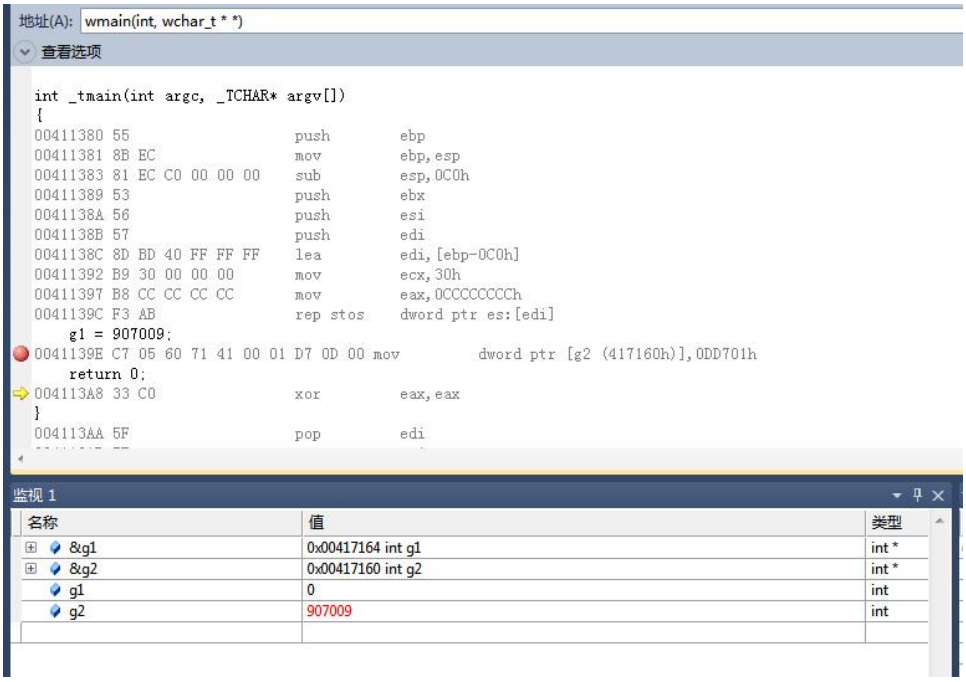


2. 最后的跳转指令偏移变为 0x831789 – 0x831826 – 5 = ffffff5e



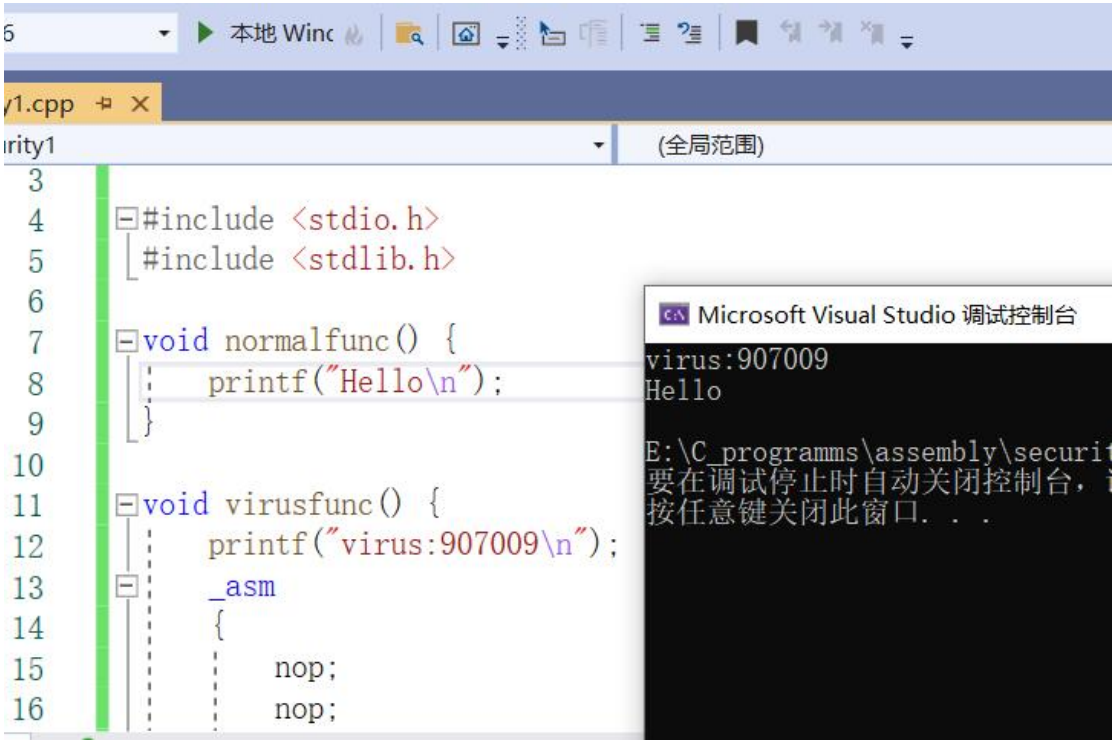
六、实验结论：

任务一：经过实验发现，修改 MOV 指令字节码，改变目的操作数地址为 g2 的地址，监视窗口 g2 的值会变成 907009，而 g1 变成 0。

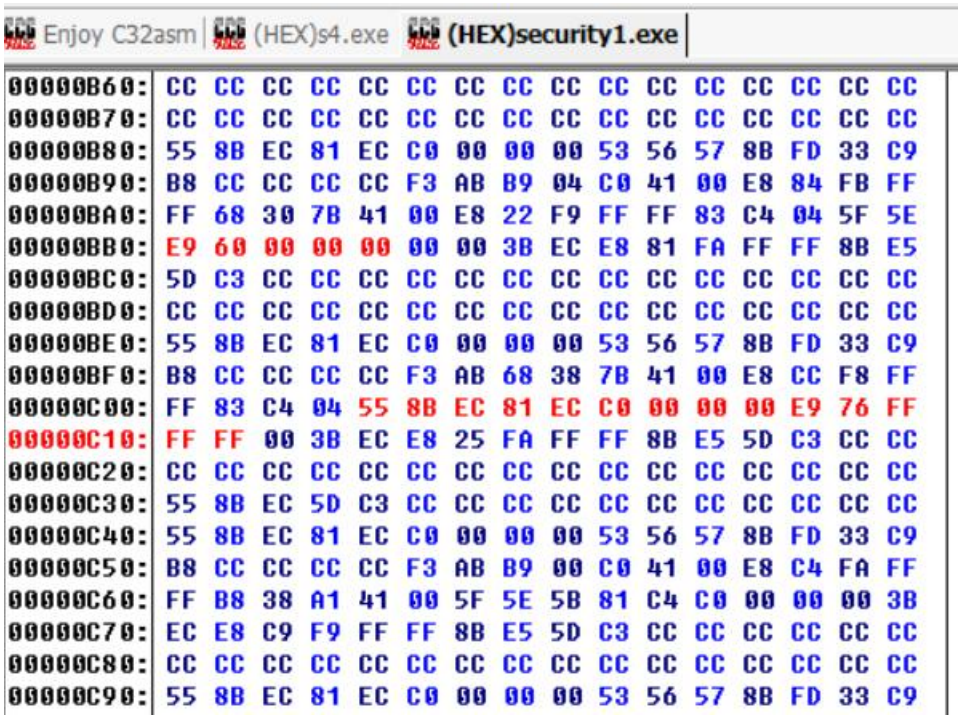


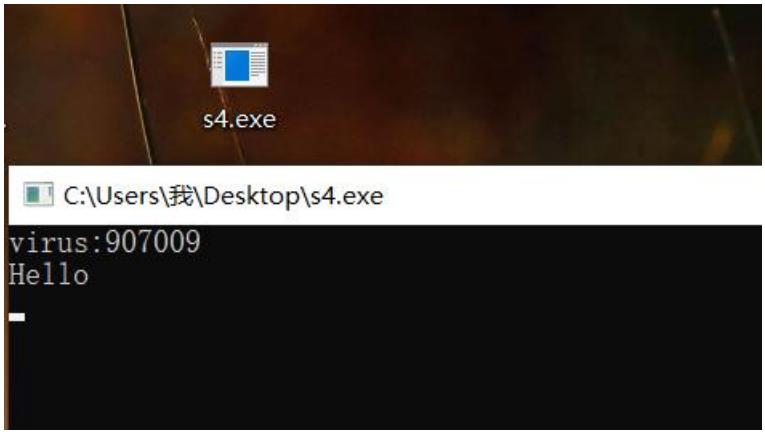
任务二：先在内存中修改 nornalfunc 的前三条指令为跳转指令 e9 7b 00 00 00，发现可以跳转到病毒并执行；

在 virusfunc 的 nop 指令处，恢复之前覆盖的 3 条指令，加入 JMP 指令跳转 normalfunc，指令为 e9 76 ff ff ff；结果显示两个字串：”virus:907009”，”Hello”，说明程序正确执行。

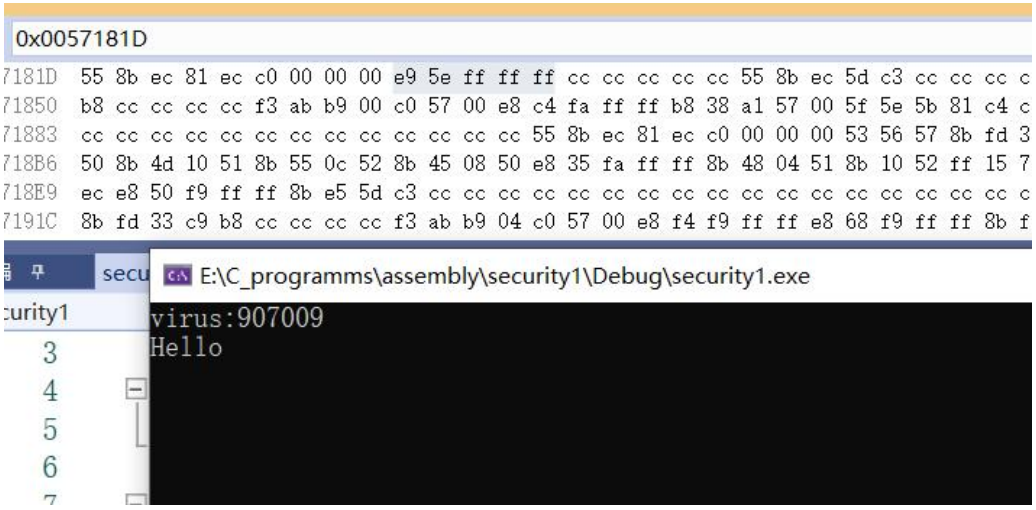


将上述在内存进行的修改在文件上进行一遍；使用 C32Asm 打开该可执行文件；修改指令，发现新的可执行文件可以执行病毒和正常程序。





任务三：在内存中修改指令，最后的恢复指令和跳转指令在 virusfunc 的 ret 指令处填写；发现病毒程序仍可正确运行。



七、总结及心得体会：

通过这次实验，我掌握了汇编指令与字节码的对应关系；能够更加熟练地运用 Visual Studio 进行程序调试；同时了解了一般的病毒如何在不干扰正常程序运行的情况下执行自身；对跳转指令认识更加深刻，理解了大端机与小端机的区别。经过实验后，我发现需要重视存在经常在不同内存中跳转的程序，可能是病毒。

八、对本实验过程及方法、手段的改进建议：

希望可以增加关于高级语言进行的病毒实验；直接打开可执行文件修改机器码，现实世界中很难实现。

报告评分：

指导教师签字：