《漏洞利用及渗透测试基础》实验报告

姓名：赵悦蛟 学号：2313650 班级： 1071

**实验名称：**

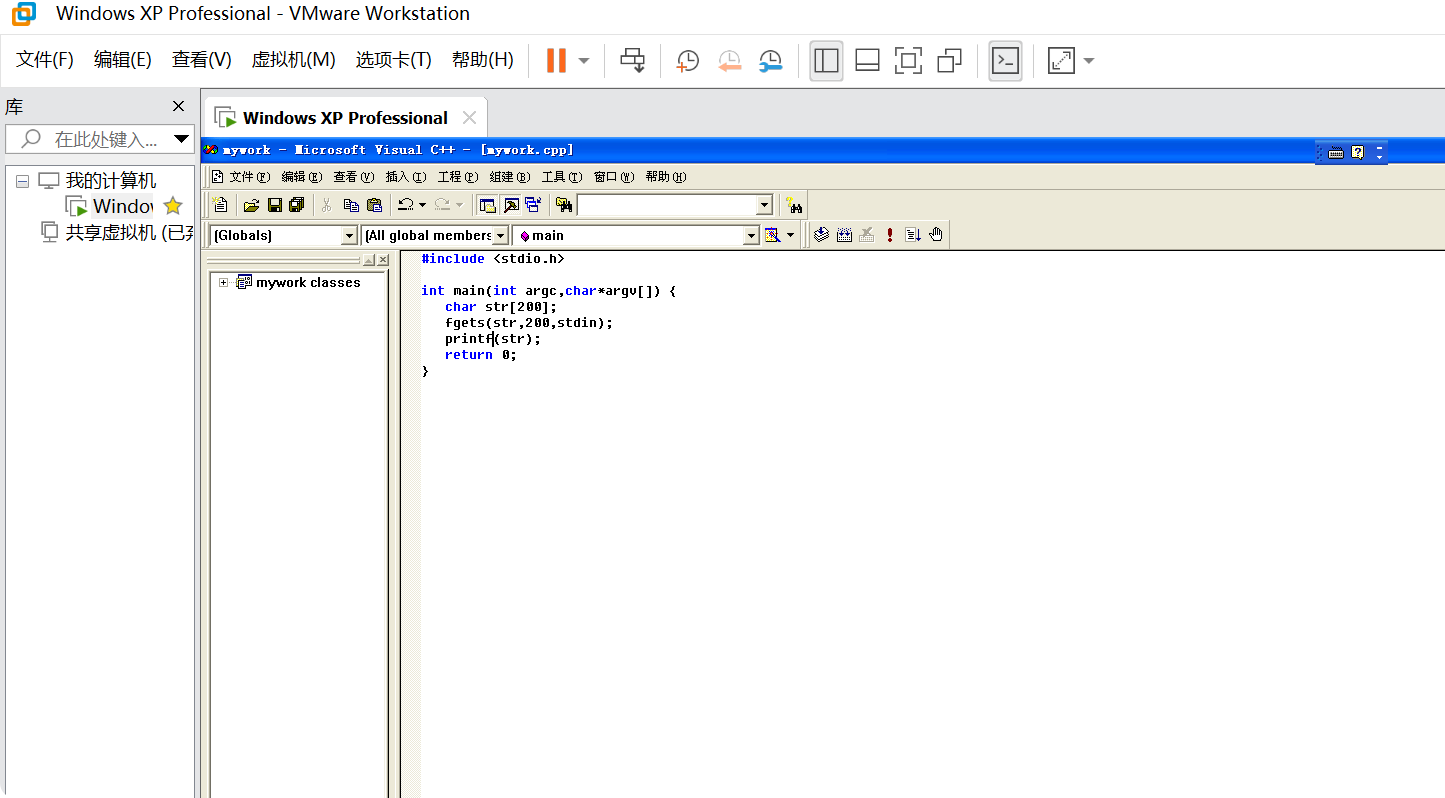
格式化字符串漏洞

**实验要求：**

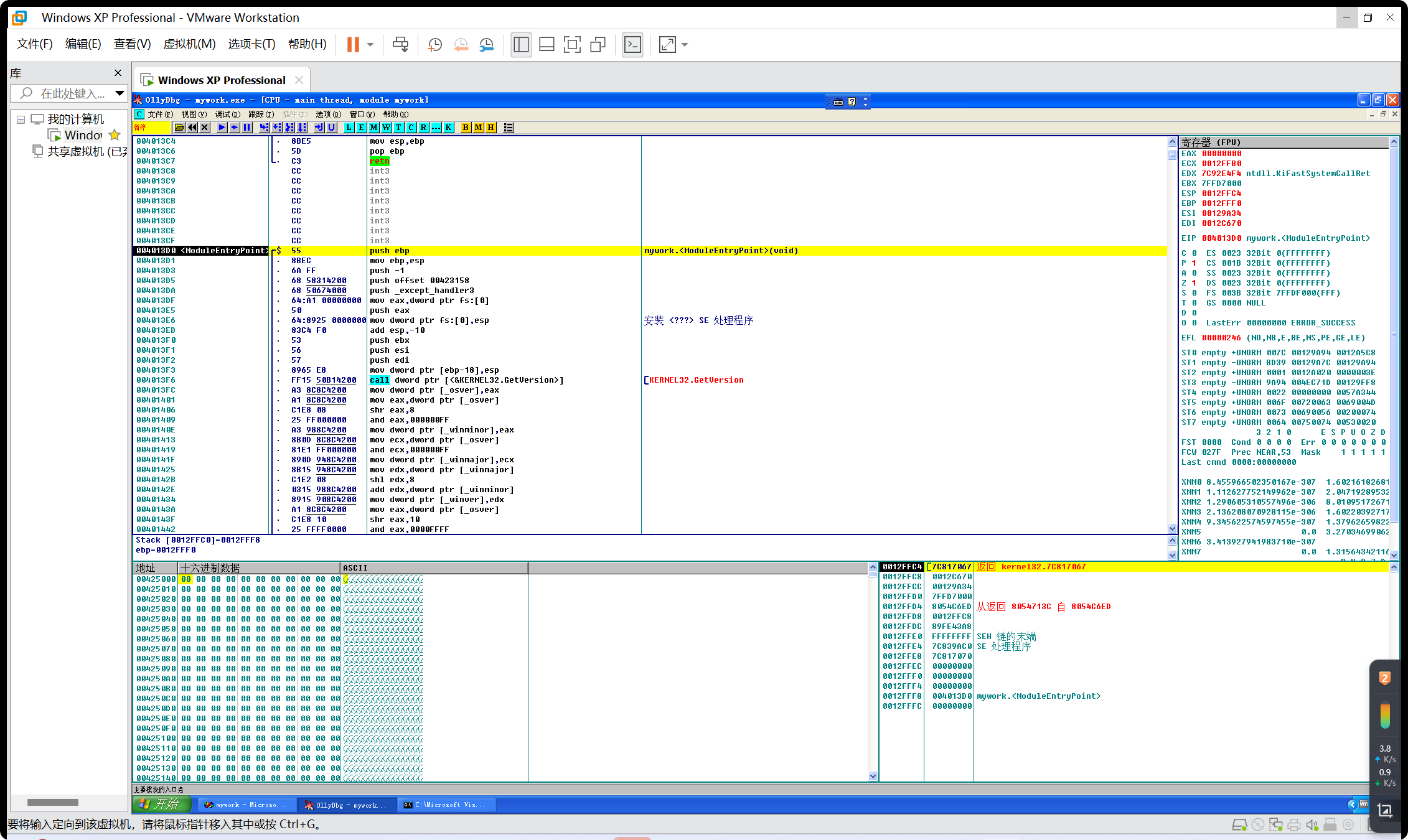
以第四章示例4-7代码，完成任意地址的数据获取，观察Release模式和Debug模式的差异，并进行总结。

**实验过程：**

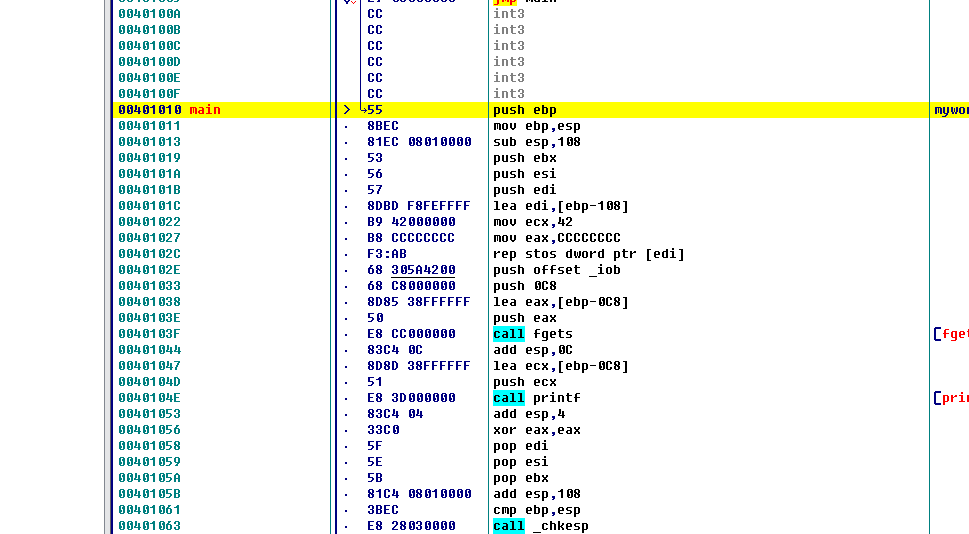
# 1.按照例4-7在xp虚拟机里用vc6编写好代码



2在Debug模式下编译生成exe文件之后进入ollydbg观察程序的执行情况。



1. 进行单步调试



当运行main函数时，sub esp,108语句在栈上分配了 264 字节（0x108） 的局部变量空间，接下来push exb,push esi ,push edi是按照cdecl调用约定对寄存器进行压栈备份，保存寄存器的状态。

接下来进行初始化栈空间，

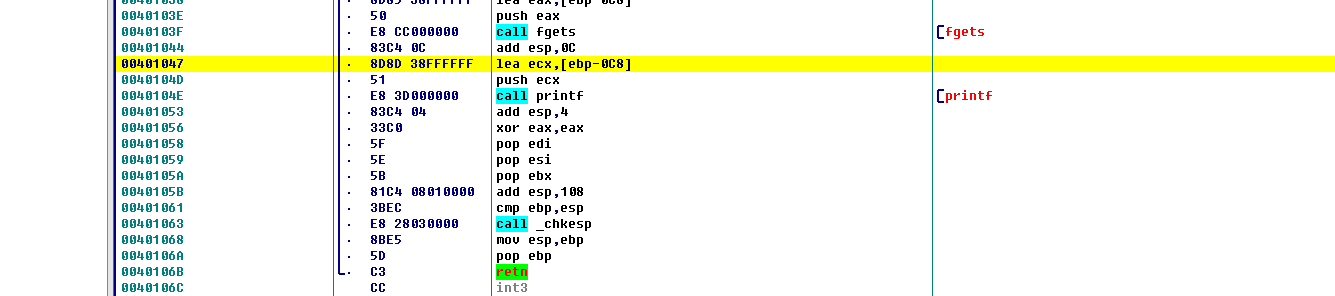
lea edi, [ebp-108]：edi 指向栈空间起始地址（ebp-108h）

mov ecx, 42h：循环计数 66 次，每次处理 4 字节，总共填充了 264 字节。

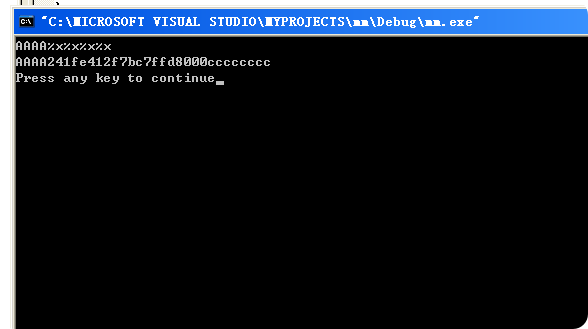
mov eax, CCCCCCCCh：填充值 0xCC

rep stosd：用 0xCC 填充栈空间

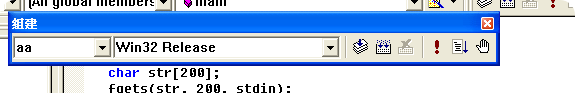
再call fgets，调用fgets读取输入，输入AAAA%x%x%x%x，之后调用printf函数输出

****

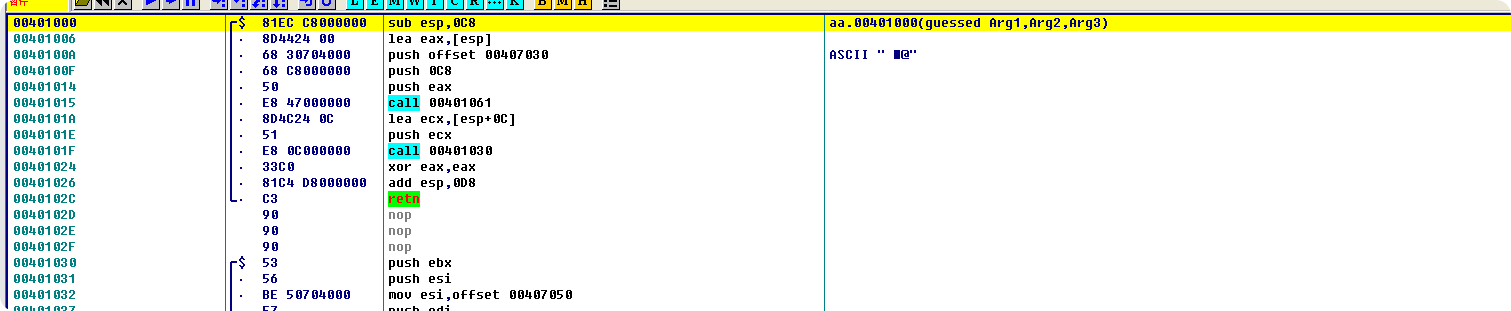
执行到 call printf 时，栈顶应为 str 的地址，由于 printf 无格式字符串，直接传递 AAAA%x%x%x%x 会触发格式化字符串漏洞，此时printf 会解析 %x，打印栈上的数据：



1. release模式



将代码文件转换成release模式，再次进入ollydbg

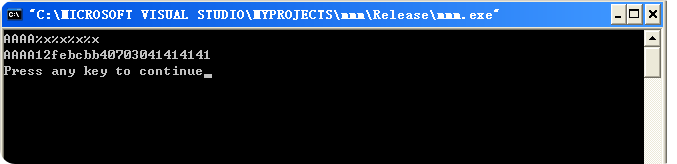


可以看到release模式时main函数的反汇编代码。与debug模式相比，release模式没有push ebp,mov ebp,esp的语句，这是因为在release模式编译器省略了传统栈帧构建步骤而改为直接操作ESP寄存器进行栈空间分配，此时 sub esp,0xc8 刚好为 str 数组的大小，没有空间的多余，而 DEBUG 模式Q为 sub esp,0x108，申请了大量的栈空间。

并且release 模式也没有保存寄存器状态，没有利用 rep stos 语句来对栈区进行初始化。接下来用3个push和一个call对应了调用函数fgets(str,200,stdin)。

在release 模式中，栈内第一行为printf()的参数str，接下来为fgets() 的三个参数，和输入的数据AAAA%X%X%X%X，因此，逻辑上跟debug模式一 样，printf执行时会往栈内高地址继续读取四个参数依次输出，但因为栈的内容不同，因此最终输出的结果不同。

此时的输出结果为：



**心得体会：**

通过本次实验，我动手编写了存在漏洞的代码，并使用 OllyDbg 进行动态调试，观察了程序在内存中的实际行为，让我对漏洞的利用方式有了更直观的认识。通过构造特定输入可以触发格式化字符串漏洞。

实验过程中，我对比了 Debug 和 Release 模式的反汇编代码，学习了解了两种模式的区别：Debug 模式为了便于调试，会分配额外的栈空间，并用 0xCC 填充，同时保存寄存器状态。这些操作虽然增加了安全性检查，但也使得栈布局更容易预测。而Release 模式 则进行了高度优化，仅分配必要的栈空间，且去掉了冗余的初始化代码。这种优化虽然提升了效率，但也可能掩盖某些漏洞的表现形式。

因此在分析漏洞时，还要考虑编译环境的影响，不能仅依赖 Debug 模式的结论。同时我还掌握了如何在vc6中切换这两种模式。

此外通过在ollydbg中进行单步跟踪与调试，我对ollydbg平台也更加熟悉。