《软件安全》实验报告

姓名：邢清画 学号：2211999 班级：1023

**实验名称：**

格式化字符串漏洞

**实验要求：**

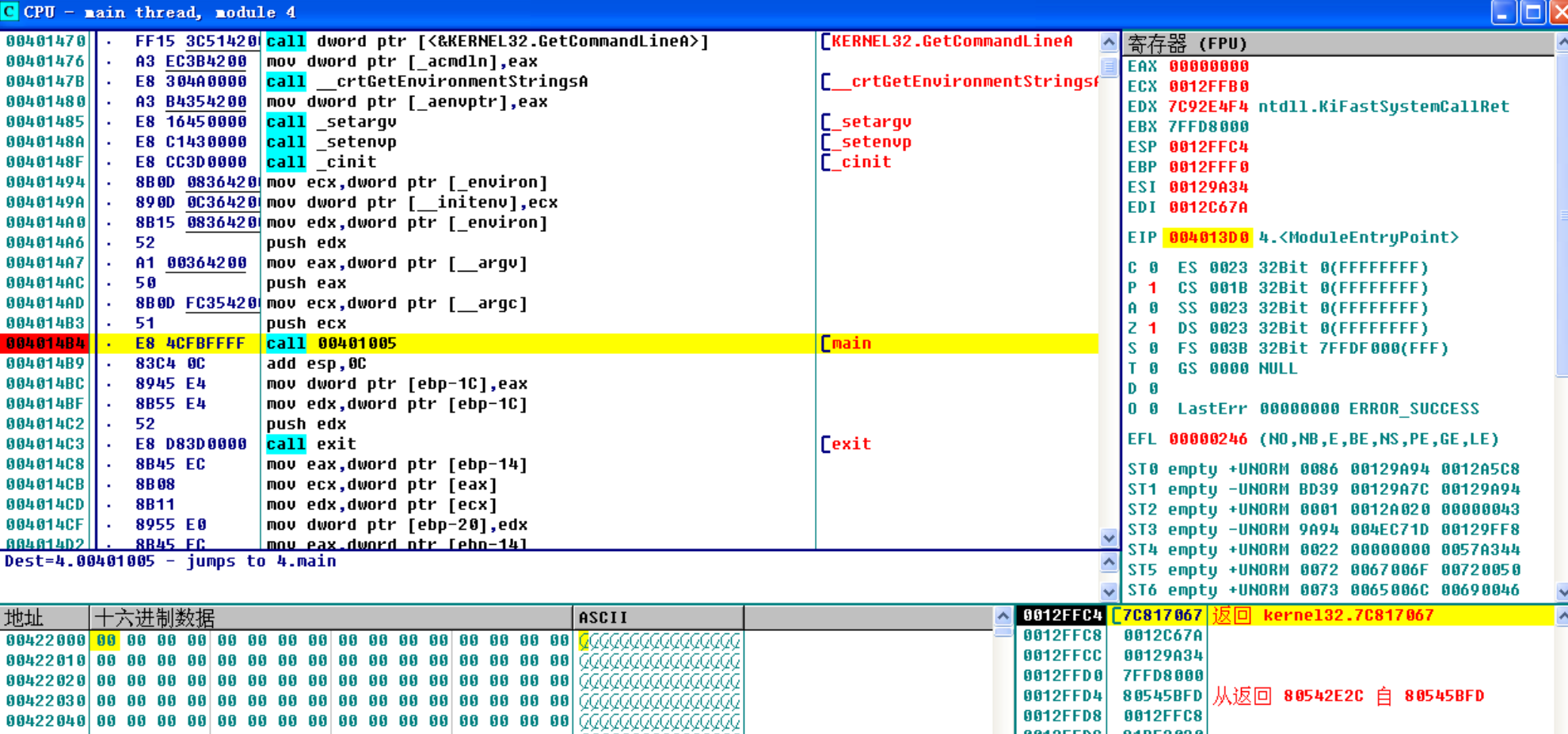
以第四章示例4-7代码，完成任意地址的数据获取，观察Release模式和Debug模式的差异，并进行总结。

**实验过程：**

**1. 测试在Debug模式下输入”AAAA%x%x%x%x”。**

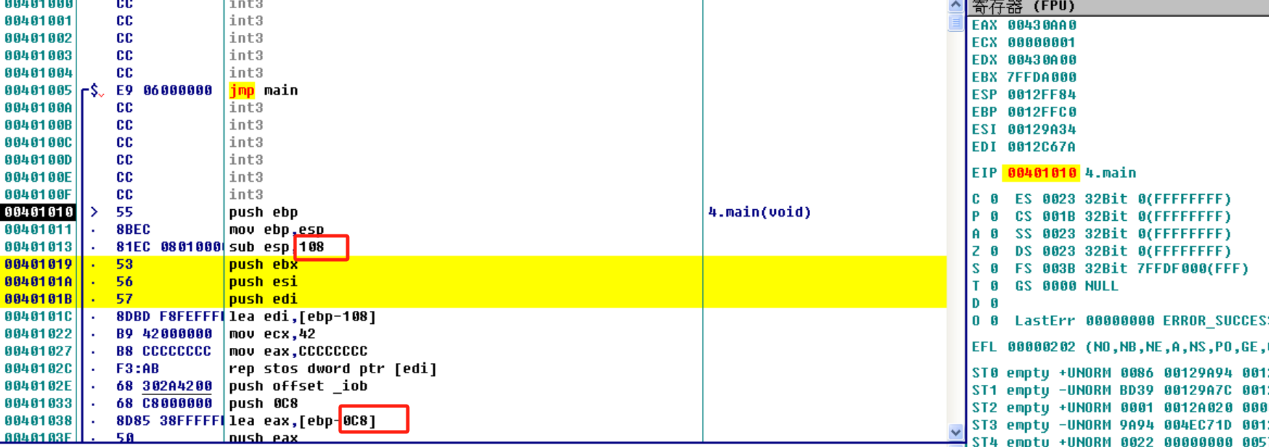
1.1输入代码后，在DEBUG模式下编译生成exe文件，在ollgdbg中打开文件。

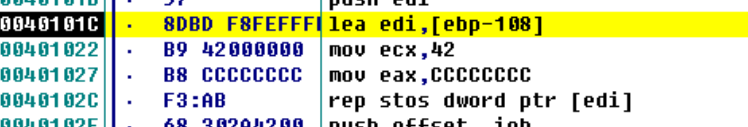
向下查找到main函数的地址：



1.2 F7单步调试进入函数内部.

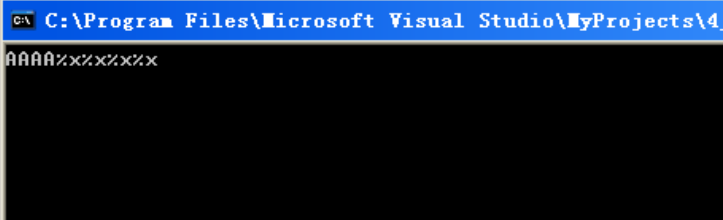
发现栈内初始化了很大一块区域，且sub esp,0x108为局部变量赋予的空间远大于数组str所需的空间（str所需的空间为0xc8）

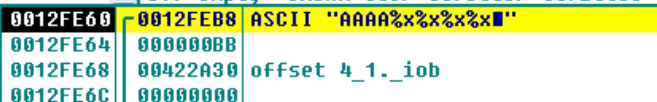


push ebx，push esi，push edi连续执行了三个入栈，保存了这些寄存器的状态。

Mov ecx,Mov eax对栈区进行初始化赋值为CCCCCCCC，然后通过rep stos对大小为108的栈区进行循环赋值。

在调用fgets之前进行三个push，对应三个参数。在终端输入AAAA%x%x%x%x,此时0012FEB8处的值为输入的字符串。且在DEBUG模式下，三个push对应一个add.

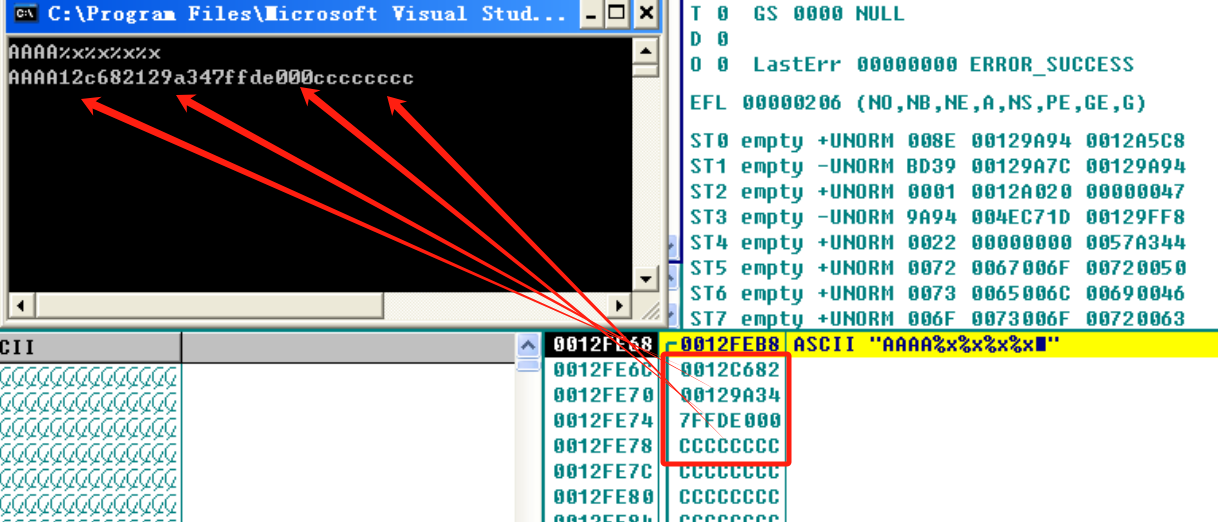




1.3 不断执行到printf处AAAA%x%x%x%x字符串入栈.

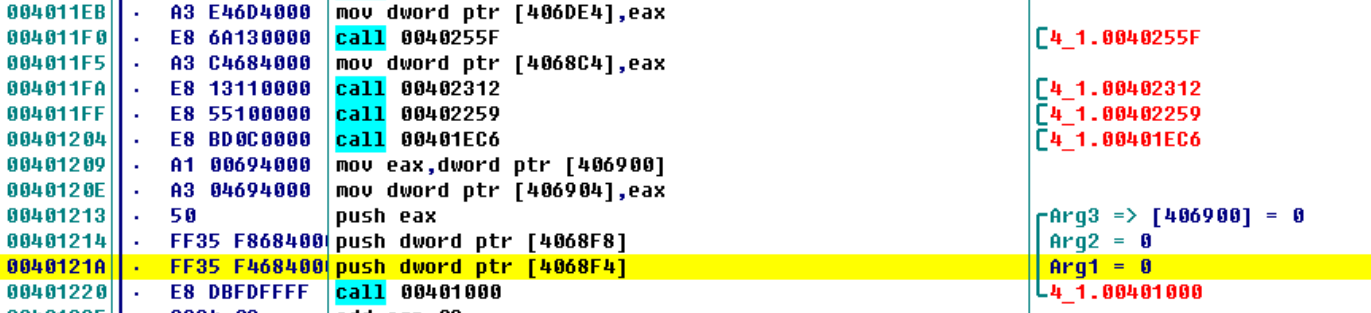
调用格式化输出函数的时候，由于输入的字符串中包含格式化字符，因此函数将会按照格式化字符去寻找参数，即首先会输出”AAAA”，然后到AAAA对应的0012FEB8后面紧跟的四个地址处寻找。



F8执行printf，得到最终的输出结果。栈中的第一个位置是输入的字符串的首地址，之后是主函数栈帧中保存的edi，esi和ebx。去依此寻找四个对应的参数，由于我们并没有提供对应的参数，函数将会在栈中寻找，第一个数据为”0012C682”，按照16进制输出为” 0012C682”，然后找第二个数据”00129A34”，按照16进制输出仍为” 00129A34”，然后寻找第三个数据”7FFDE000”，按照16进制输出为”7FFDE000”，最后寻找第四个数据”CCCCCCCC”，按照16进制输出为”CCCCCCCC”。可以发现，发生了内存的泄漏。

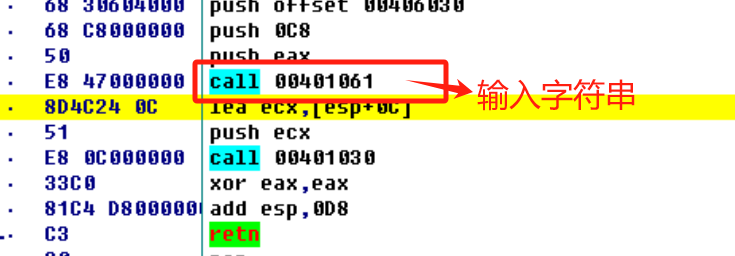
**2. 测试在Release模式下输入”AAAA%x%x%x%x”**

进入release模式，观察差异。找到主程序的入口。

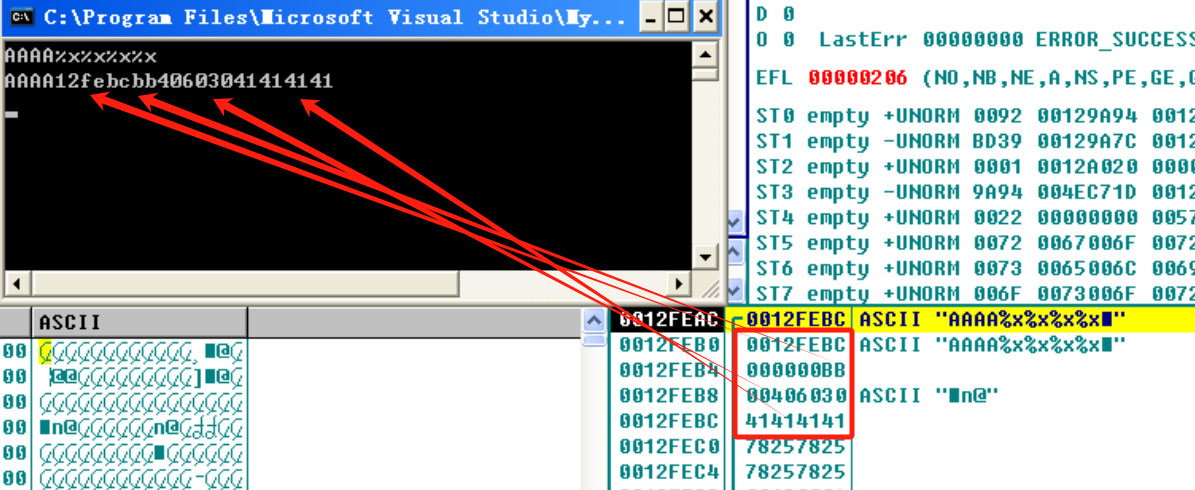


到call 00401000按F7直接跳转到主函数地址，没有再一次发生jump，返回地址入栈。

观察主函数对栈帧的切换有所不同：没有ebp入栈，sub esp,0xc8说明栈顶并没有抬高之前（DEBUG 模式下为 sub esp,0x108）那么多，仅抬高了0xC8即200字节，来为局部变量分配空间。且DEBUG模式下很多push寄存器的值的操作也没有出现（代码更加简洁，提高运行效率）。

输入AAAA%x%x%x%x，结束后，在DEBUG模式下会有一步add eip的操作，但Release模式没有。

运行到call 00401030，观察对应关系。

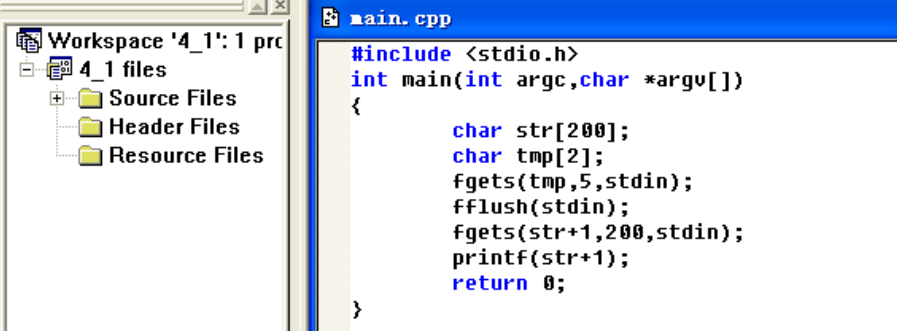


输出结果与DEBUG模式不同，结果为：AAAA12febcbb40603041414141

相应的，如果输入的字符串为AAAA%x%x%x%s,则最后会是41414141地址里面特定的数据。

**3. 测试在debug和Release模式下输入”AAAA%x%x%x%s**

3.1在debug模式下输入字符串不能得到预期结果，因为对地址CCCCCCCC的访问是非法的。

3.2在release模式下输入字符串，先修改源码,构造合理的地址进行访问，选取合适的地址进行实验验证。

**4. Release和Debug模式的差异**

Debug模式包含更多的调试信息，并且会扩大提供给局部变量的空间，全部初始化为0xCC，且在栈中会保留更多的寄存器（EDI,ESI,EDX,EBP等）的数据。

Release模式为局部变量分配空间不会超过需要的空间；调试语句减少，提高效率；push寄存器的操作减少。

**心得体会：**

1. 了解了格式化漏洞的原理，明白了漏洞发生时内部的机制。

2. 使用OllyDbg在release和debug两种模式下编译出的exe文件中，对printf函数调用前的栈进行观察分析，以及对输入的带有格式化字符的字符串的输出结果进行分析，比较并总结了两者的差异。