《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：

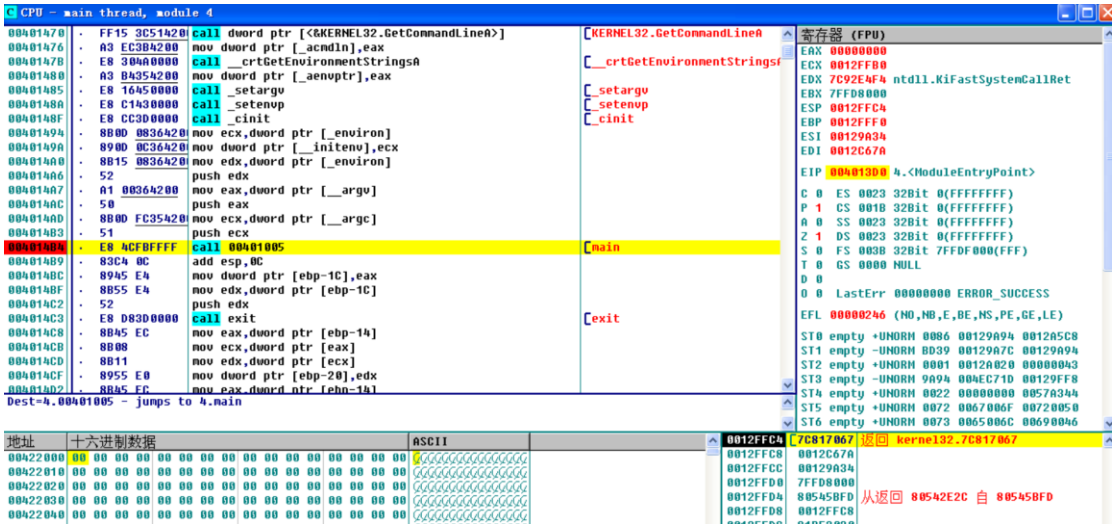
以第四章示例 4-7 代码，完成任意地址的数据获取，观察 Release 模式和 Debug 模式的差异，并进行总结。

实验过程：

1. **测试在 Debug 模式下输入"AAAA%x%x%x%x"。**

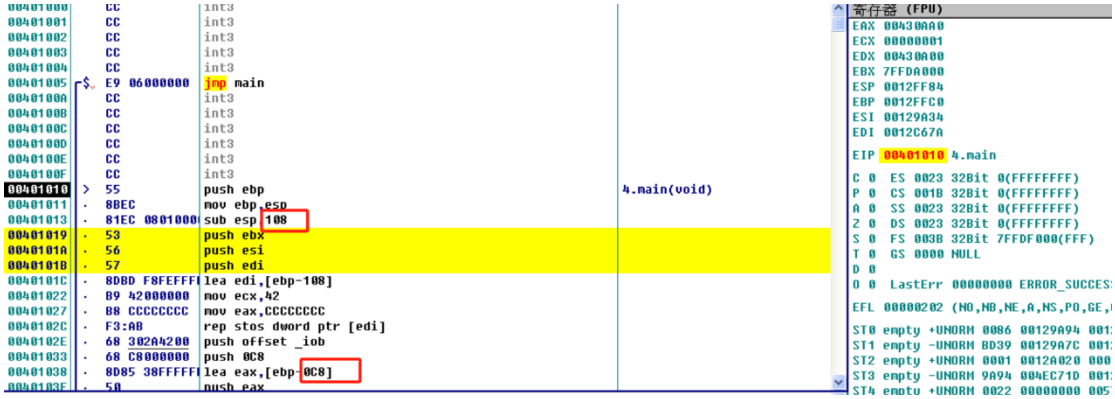
1.1 输入代码后，在 DEBUG 模式下编译生成 exe 文件，在 ollgdbg 中打开文件。

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）

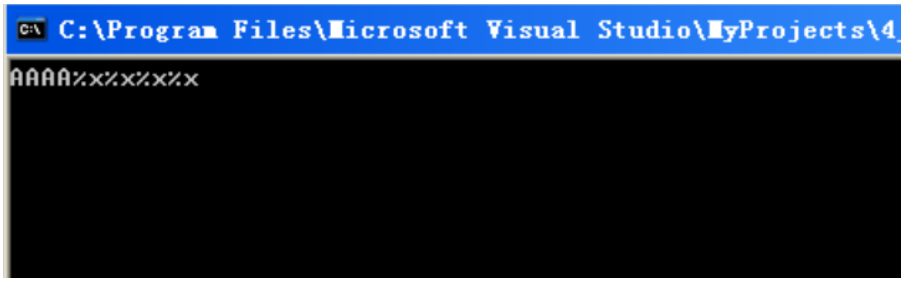


push ebx，push esi，push edi 连续执行了三个入栈，保存了这些寄存器的状态。

```
00401016  .  57           push edi
0040101C  .  8DBD F8FEFFF lea edi,[ebp-108]
00401022  .  B9 42000000  mov ecx,42
00401027  .  B8 CCCCCCCC  mov eax,CCCCCCCC
0040102C  .  F3:AB        rep stos dword ptr [edi]
0040102E  .  68 30204200  push offset 4_1._iob
```

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.



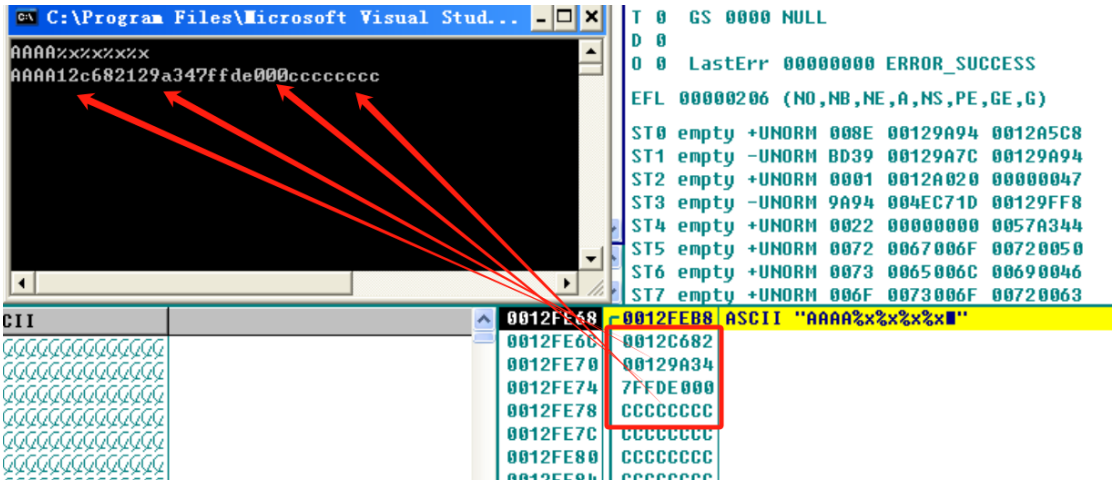

```
0012FE60  0012FEB8  ASCII "AAAA%x%x%x%x"
0012FE64  000000BB
0012FE68  00422A30  offset 4_1._iob
0012FE6C  00000000
```

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。

```
call fgets                    [fgets
add esp,0C
lea ecx,[ebp-0C8]
push ecx
call printf                   [printf
```

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。

```
004011EB   .  A3 E46D4000   mov dword ptr [406DE4],eax
004011F0   .  E8 6A130000   call 0040255F                          [4_1.0040255F
004011F5   .  A3 C4684000   mov dword ptr [4068C4],eax
004011FA   .  E8 13110000   call 00402312                          [4_1.00402312
004011FF   .  E8 55100000   call 00402259                          [4_1.00402259
00401204   .  E8 BD0C0000   call 00401EC6                          [4_1.00401EC6
00401209   .  A1 00694000   mov eax,dword ptr [406900]
0040120E   .  A3 04694000   mov dword ptr [406904],eax
00401213   .  50            push eax                               [Arg3 => [406900] = 0
00401214   .  FF35 F8684000 push dword ptr [4068F8]                 Arg2 = 0
0040121A   .  FF35 F4684000 push dword ptr [4068F4]                 Arg1 = 0
00401220   .  E8 DBFDFFFF   call 00401000                          [4_1.00401000
```

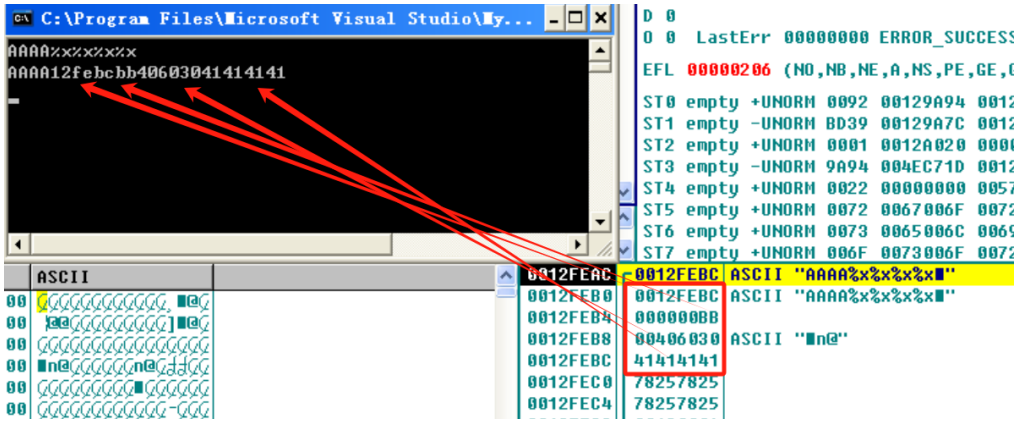
到 call 00401000 按 F7 直接跳转到主函数地址，没有再一次发生 jump，返回地址入栈。

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。

```
.  68 30604000   push offset 00406030
.  68 C8000000   push 0C8
.  50           push eax
.  E8 47000000   call 00401061                    输入字符串
.  8D4C24 0C     lea ecx,[esp+0C]
.  51           push ecx
.  E8 0C000000   call 00401030
.  33C0         xor eax,eax
.  81C4 D8000000 add esp,0D8
.  C3           retn
```

运行到 call 00401030，观察对应关系。



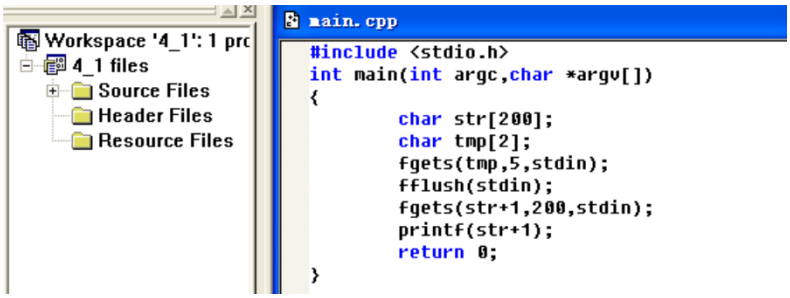
输出结果与 DEBUG 模式不同，结果为：AAAA12febcbb40603041414141

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。