

《软件安全》实验报告

姓名：曹瑜 学号：2212794 班级：密码科学与技术

实验名称：

格式化字符串漏洞

实验要求：

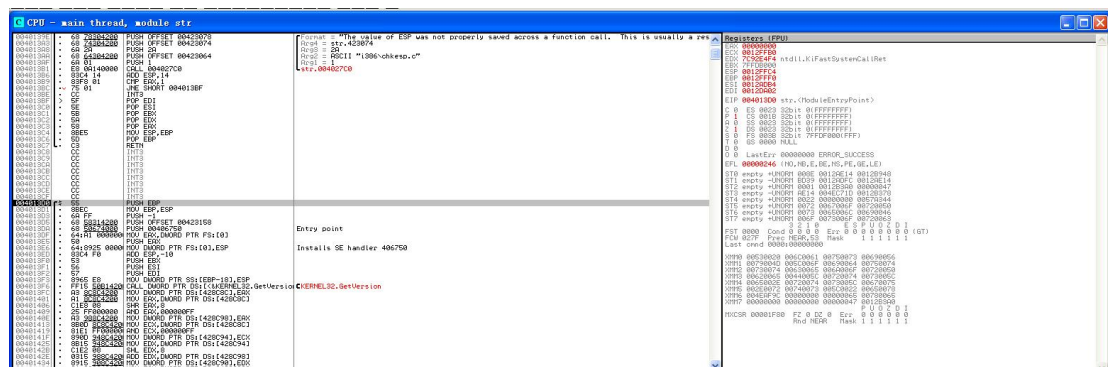
以第四章示例 4-7 代码，完成任意地址的数据获取，观察 Release 模式和 Debug 模式的差异，并进行总结

实验过程：

1. 进入 VC 反汇编

```
members | main
#include <stdio.h>
int main(int argc, char *argv[])
{
    char str[200];
    fgets(str,200,stdin);
    printf(str);
    return 0;
}
```

2. DEBUG 模式下编译后到 o1lyDBG 中打开：



找到 Main 函数执行区：

```
00401494 | . 8B0D 88C42001 MOV ECX,DWORD PTR DS:[428CA8]
00401496 | . 890D AC8C4201 MOV DWORD PTR DS:[428CAC],ECX
00401498 | . 8B15 88C42001 MOV EDI,DWORD PTR DS:[428CA8]
0040149E | . 52          PUSH EDI
004014A7 | . A1 88C42001 MOV EAX,DWORD PTR DS:[428CA0]
004014AC | . 59          PUSH EAX
004014AD | . 8B0D 9C8C4201 MOV ECX,DWORD PTR DS:[428C9C]
004014B3 | . 51          PUSH ECX
004014B4 | . E8 40FBFFFF CALL 00401005
004014B5 | . 83C4 0C     ADD ESP,0C
004014BC | . 8945 E4     MOV DWORD PTR SS:[EBP-1C],EAX
004014BF | . 8B55 E4     MOV EDX,DWORD PTR SS:[EBP-1C]
004014C2 | . 52          PUSH EDI
004014C3 | . E8 583F0000 CALL 00405420
004014C8 | . 8B45 EC     MOV EAX,DWORD PTR SS:[EBP-14]
004014CB | . 8B08        MOV ECX,DWORD PTR DS:[EAX]
004014CD | . 8B11        MOV EDI,DWORD PTR DS:[ECX]
004014CF | . 8955 E0     MOV DWORD PTR SS:[EBP-20],EDI
004014D2 | . 8B45 EC     MOV EAX,DWORD PTR SS:[EBP-14]
004014D5 | . 50          PUSH EAX
004014D6 | . 8B4D FA     MOV ECX,DWORD PTR SS:[EBP-2A]
004014D7 | . 8B4D FA     MOV ECX,DWORD PTR SS:[EBP-2A]
```

Arg3 => [428CA8] = 0
Arg2 => [428CA0] = 0
Arg1 => [428C9C] = 0
str.00401010

跳转到执行区：观察到操作为：抬高栈帧，留出大量区域：

3 个 push 操作在区域顶部保存了调用函数前函数中的一些值：

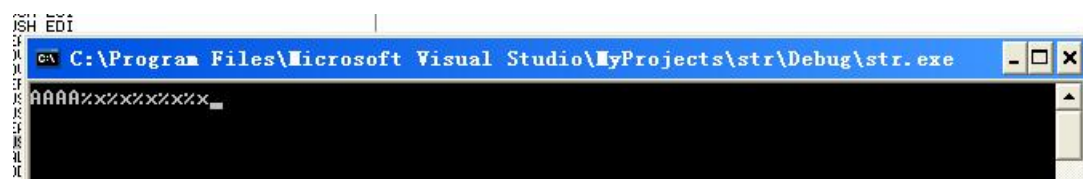
再初始化大小为 108 的空间为

CCCCCCCC；

```
0040100F | . CC          INT3
00401010 | . CC          INT3
00401011 | . 8BEC        MOV EBP,ESP
00401013 | . 81EC 00010001 SUB ESP,108
00401019 | . 53          PUSH EBX
0040101A | . 56          PUSH ESI
0040101B | . 57          PUSH EDI
0040101C | . 8B0D F8FEFF LEA EDI,[LOCAL.66]
00401022 | . 89 42000000 MOV ECX,42
00401027 | . B8 CCCCCCCC MOV EAX,CCCCCCCC
0040102C | . F3AB        REP STOS DWORD PTR ES:[EDI]
0040102E | . 68 305A4200 PUSH OFFSET 00425A30
00401033 | . 68 C0000000 PUSH 0C8
00401038 | . 8D85 38FFFF LEA EAX,[LOCAL.50]
0040103E | . 50          PUSH EAX
0040103F | . E8 CC000000 CALL 00401110
00401044 | . 83C4 0C     ADD ESP,0C
00401047 | . 8D8D 38FFFF LEA ECX,[LOCAL.50]
0040104D | . 51          PUSH ECX
0040104E | . E8 3D000000 CALL 00401090
00401053 | . 83C4 04     ADD ESP,4
00401056 | . 32C0        XOR EAX,EAX
00401059 | . 5F          POP EDI
0040105A | . 5E          POP ESI
0040105B | . 5B          POP EBX
0040105D | . 81C4 00010001 ADD ESP,108
00401061 | . 8BEC        MOV EBP,ESP
00401063 | . E8 20030000 CALL 00401390
00401068 | . 8BE5        MOV ESP,EBP
0040106A | . 5D          POP EBP
0040106B | . C3          RETN
0040106C | . CC          INT3
```

Arg3 = str.425A30
Arg2 = 0C8
Arg1 => OFFSET LOCAL.50
str.00401110

输入字符串：查看寄存器模块可见字符串存储在 0x0012FB8 中



当前模式下，每发生 3 个 push 就对应一个 add 操作，即调用 add 后会返回栈帧状态：

程序最终执行结果为：

输入：AAAA%x%x%x%x

输出：AAAA12da0612adb47ffd8000cccccccc

```
AAAA%x%x%x%x
AAAA12da0612adb47ffd8000cccccccc
```

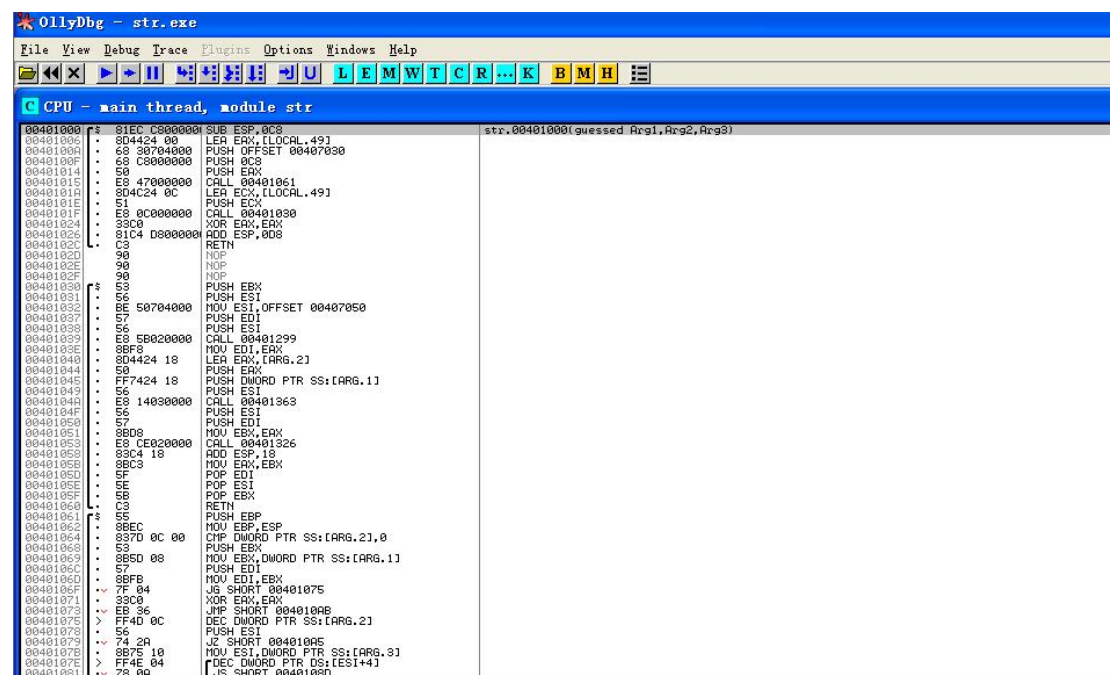
可见此时发生了内存泄漏；

可知在 debug 模式下，想读取 str 的地址需要很多的格式化字符；

3、 release 模式：

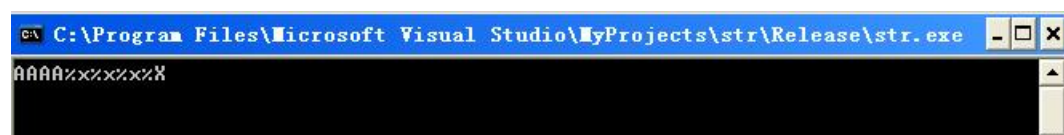
该模式中调试信息比 debug 模式少；

主函数中并没有发现 ebp 入栈，而是发现了一个 sub 抬高，此外，也没有 push 和寄存器的值，比较简洁，效率也更高



没有 ebp 入栈，只有栈顶抬高；

输入字符串：查看寄存器模块可见字符串存储在 0x0012FB8 中



最终结果：

输入：AAAA%x%x%x%x

输出：AAAAAAAA18FE84BB40603041414141

在输出结果中 0x41 为 A 的 ASCII 值



心得体会：

通过本次实验，掌握了 debug 与 release 模式的差异：在 Debug 模式下，开辟了足够大的栈帧并初始化，使得 `char str[200]` 从靠近 EBP 的地址分配空间，要读到 `str` 的地址就需要很多的格式化字符；在 Release 模式下，没有严格按照制式的栈帧分配，执行到 `printf(str)` 的时候，栈区自顶到底部有分存内容，更加简洁。