软件安全实验六

班级：2022211801　　学号：2022211570　　姓名：项枫

一、目标

1、%x 查看栈内容

2、%s 查看指定地址内容

3、复习缓冲区溢出覆写返回地址跳转 shellcode 的操作流程

4、了解 sprintf()函数缺陷所在

5、分析 shellcode 的构造原理

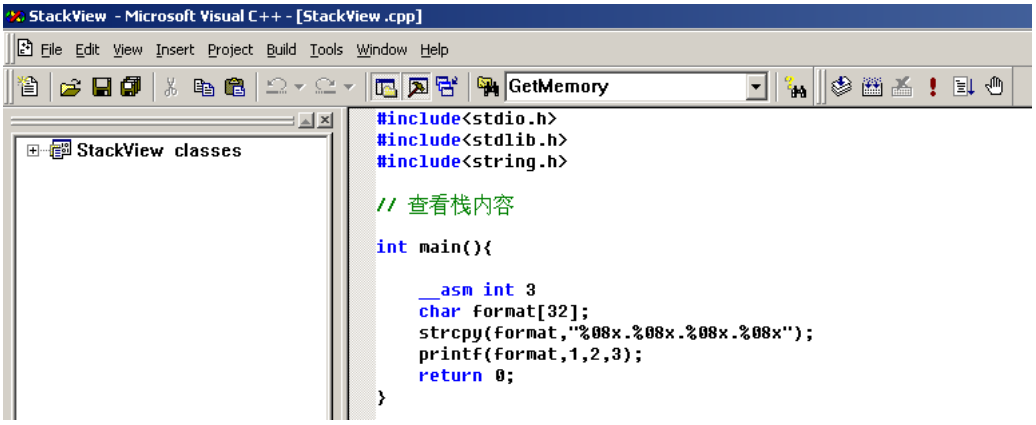
6、实际操作对格式化输出函数漏洞进行利用

二、测试步骤与结果

1、通过%x 查看栈内容

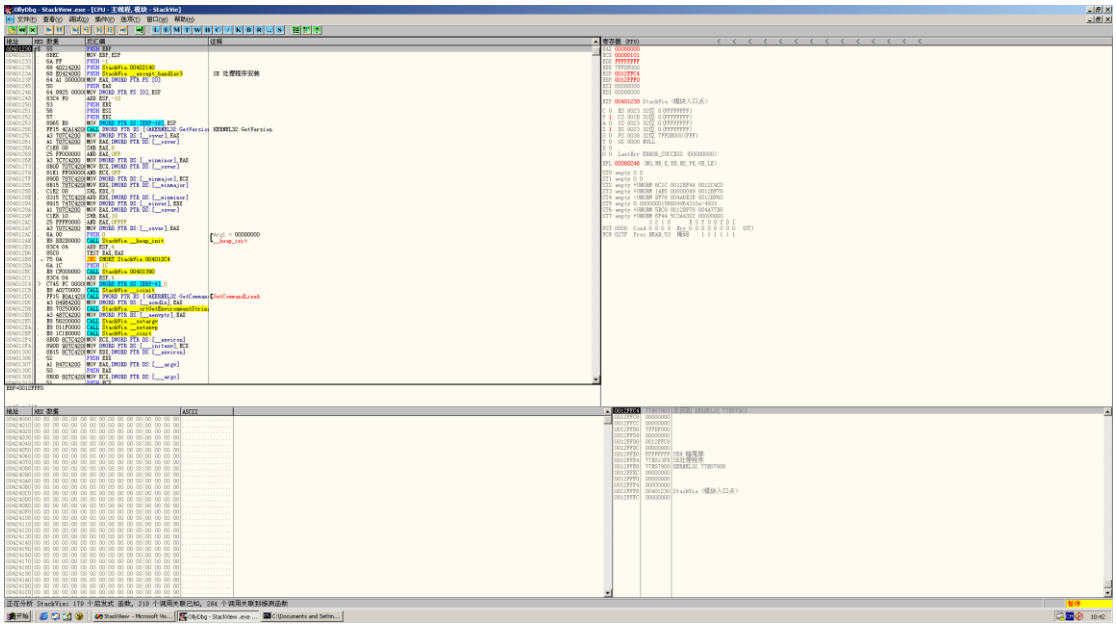
（1）查看代码

代码如下所示：

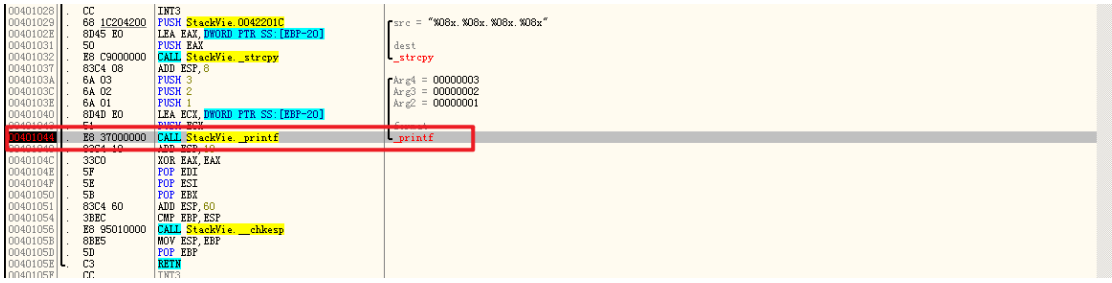


如上所示，执行 printf 时第四个%x 没有提供对应的参数，因此会显示本应该是参数所在位置的栈内容。

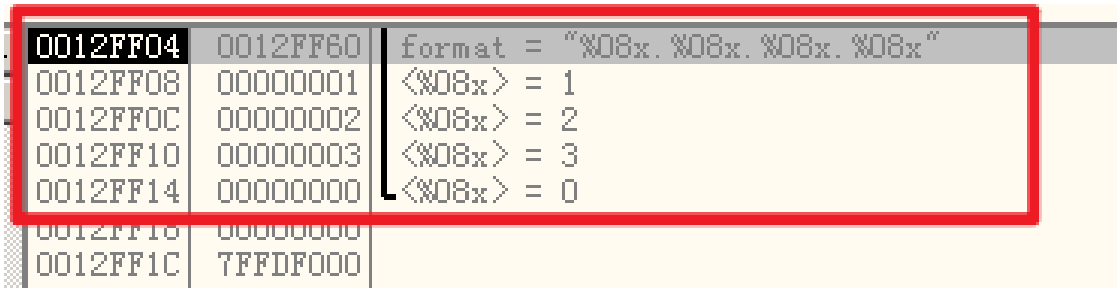
（2）在 OllyDbg 中打开该程序



运行至 CALL StackView._printf 处：



此时观察右下角：



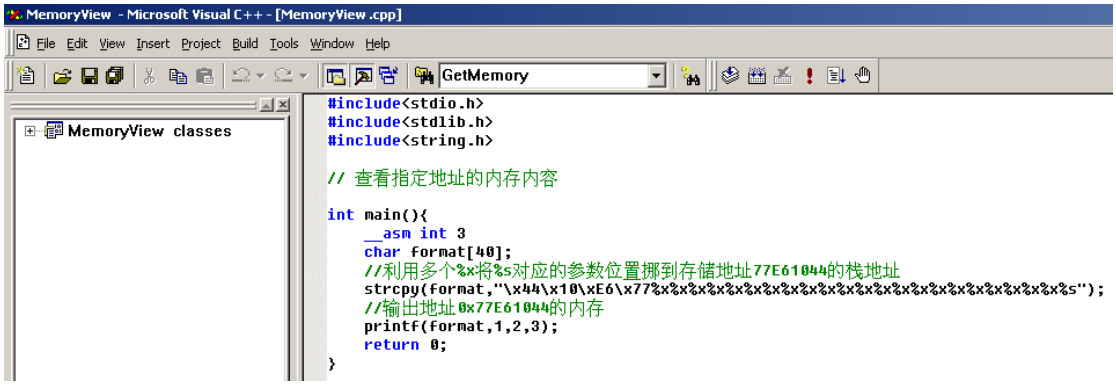
可以看到显示了本应该是参数所在位置的栈内容。

通过更多的%x 甚至可以重建大部分的栈内存：

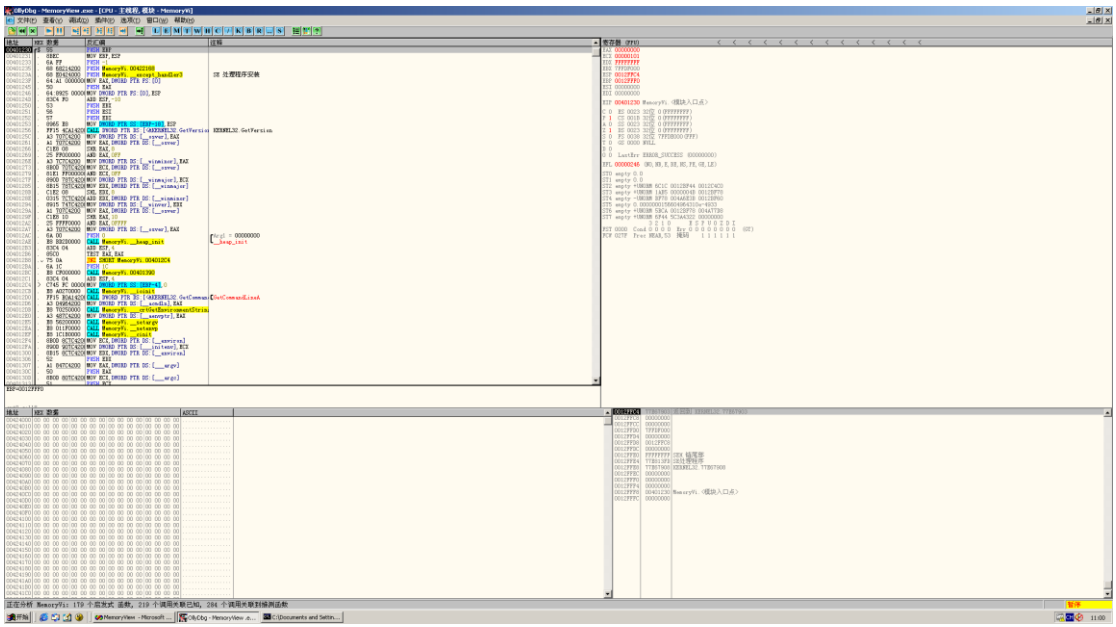
2、%s 查看指定地址内容

（1）查看代码

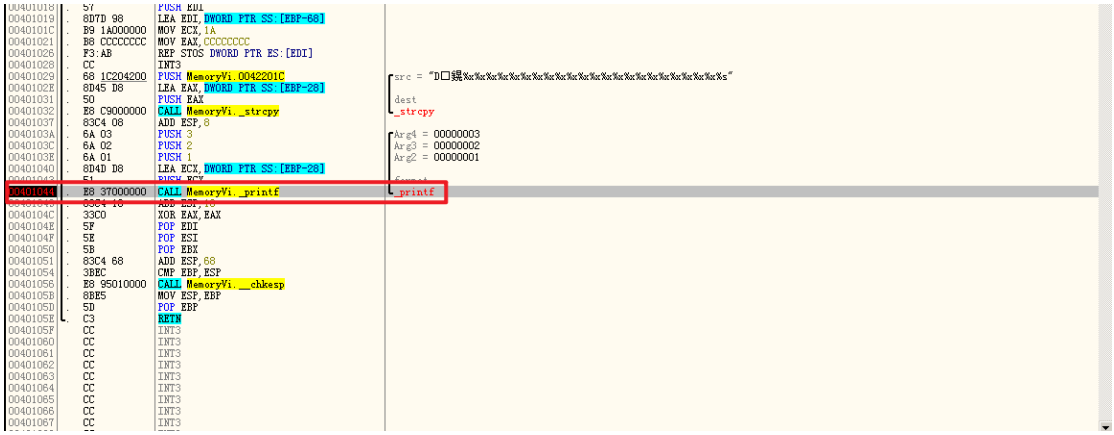
代码如下所示：



（2）在 OllyDbg 中打开该程序



执行 printf 函数：



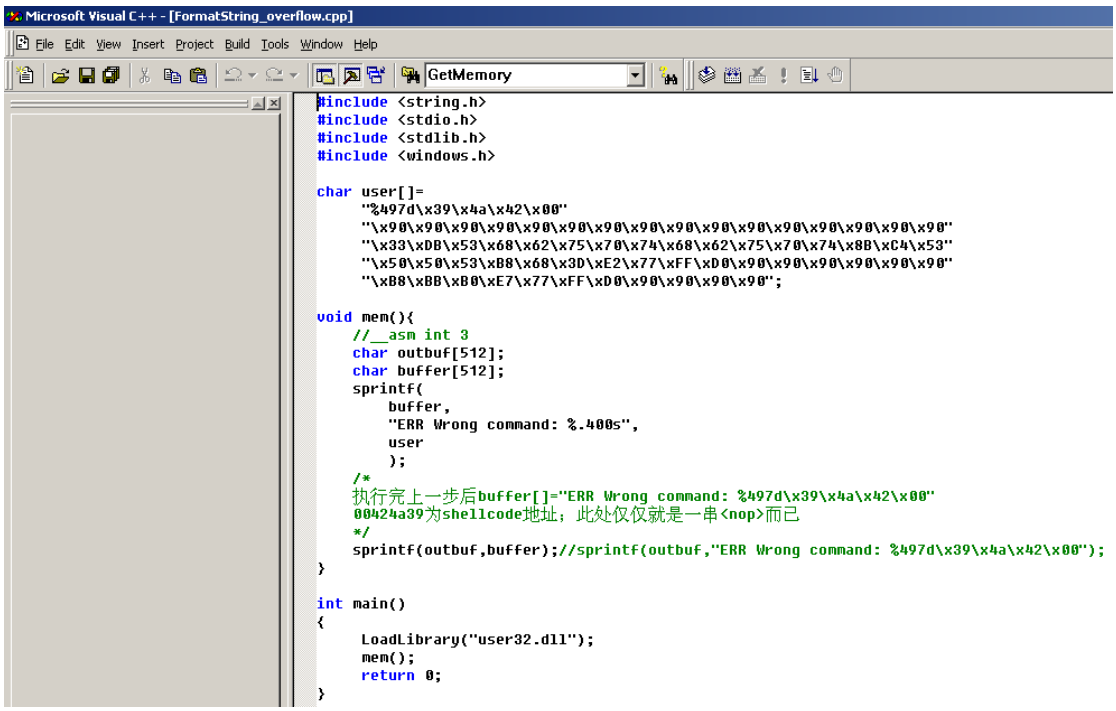
观察右下角：

1,2,3 是提供 3 个参数。利用%x 步进，将%s 的参数对应到 77E61044，因此可以输出 77E61044 开始的字符串直到遇到截断符。0x0012FF58 为 format 字符串起始地址，前四个字节即我们想要查看的内存地址 77E61044。

3、缓冲区溢出

（1）查看代码

代码如下所示：



（2）sprintf()函数

```
int sprintf(char *buffer, const char *format, [argument] … );
```

buffer 指针指向将要写入字符串的缓冲区；

format 格式化字符串；

argument 为可选参数。

漏洞：作为向字符数组中写入数据的格式化输出函数， sprintf()会假定存在任意长度的缓冲区

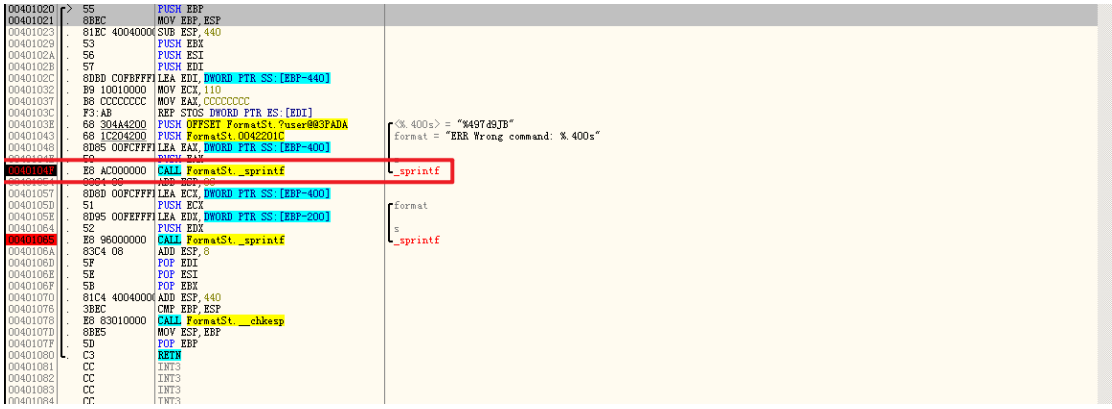
（3）shellcode 构造

作为向字符数组中写入数据的格式化输出函数，sprintf 会假定存在任意长度的缓冲区。此处将字符数组 user 作为由用户构造的输入，其中出现了非常规字符%497d，是此次实验成功的关键。\x39\x4a\x42\x00 是 shellcode 的起始地址，用来覆盖返回地址。\x90…\x33…\xD0…\x90 为此次的弹框的 shellcode。

```
char user[]=
      "%497d\x39\x4a\x42\x00"
      "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
      "\x33\xDB\x53\x68\x62\x75\x70\x74\x68\x62\x75\x70\x74\x8B\xC4\x53"
      "\x50\x50\x53\xB8\x68\x3D\xE2\x77\xFF\xD0\x90\x90\x90\x90\x90\x90"
      "\xB8\xBB\xB0\xE7\x77\xFF\xD0\x90\x90\x90\x90";
```

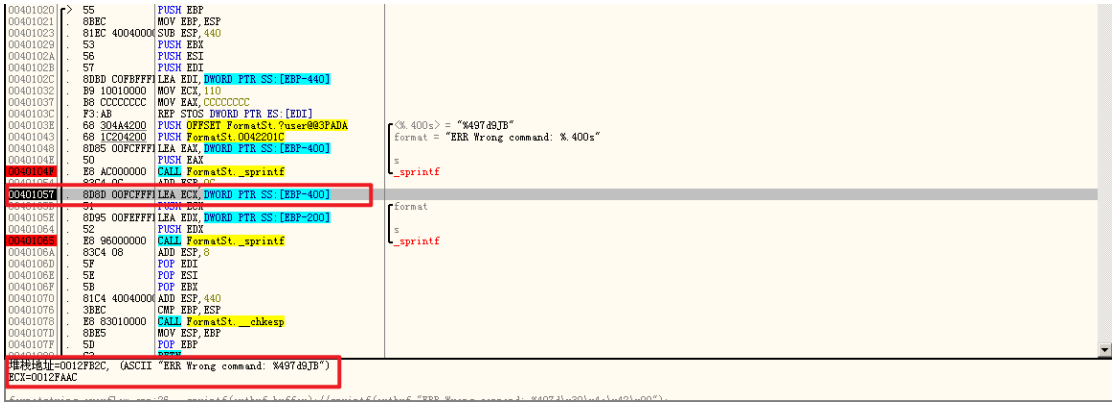
（4）第一次调用 sprintf()前

第一次调用 sprintf(buffer,"ERR Wrong command:%.400s",user);时写入数据的目的地址为 0x0012FB2C，格式化字符串为"ERR Wrong command:%.400s"，%.400s 对应的参数为起始地址为 0x00424A30 的字符串，即用户输入的字符数组 user。对地址 0x00424A30 数据窗口跟随后可以看见该字符数组的内容。



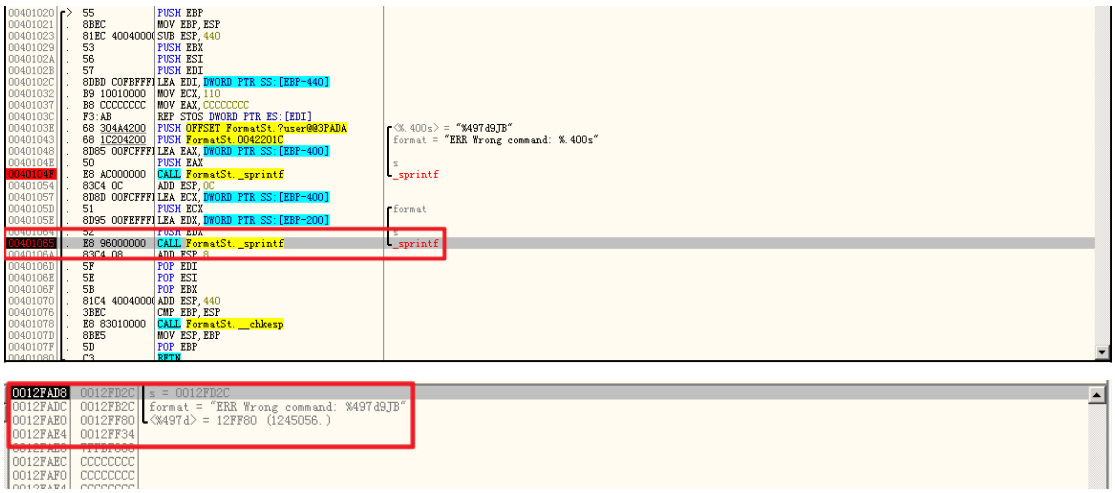
（5）第一次调用 sprintf()后

对地址 0x0012FB2C 查看 ASCII 数据，可见此时 buffer 中的字符串为"ERR Wrong command:%497d\x39\x4a\x42\x00"，其后的数据由于 0x00 而被截断。



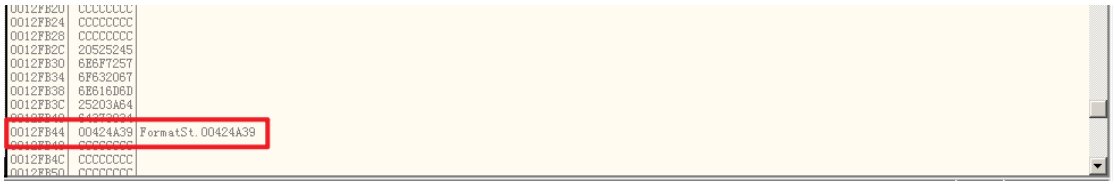
（6）第二次调用 sprintf()前

执行 sprintf(outbuf, buffer);时 buffer 中格式化字符串为"ERR Wrong command:%497d\x39\x4a\x42\x00"，根据格式化字符串，sprintf 会读取一个参数以%497d 的格式写入 outbuf，由于未提供该参数，会自动将栈地址 0x0012FAE0 中的值视为该参数，即 0x12FF80(十进制 1245056)。需要写入 outbuf 的总字符串长度为 19＋497＝516，而 outbuf 长度为 512，因此会导致栈溢出，使得函数返回执行 sprintf 后 outbuf 中的内容。



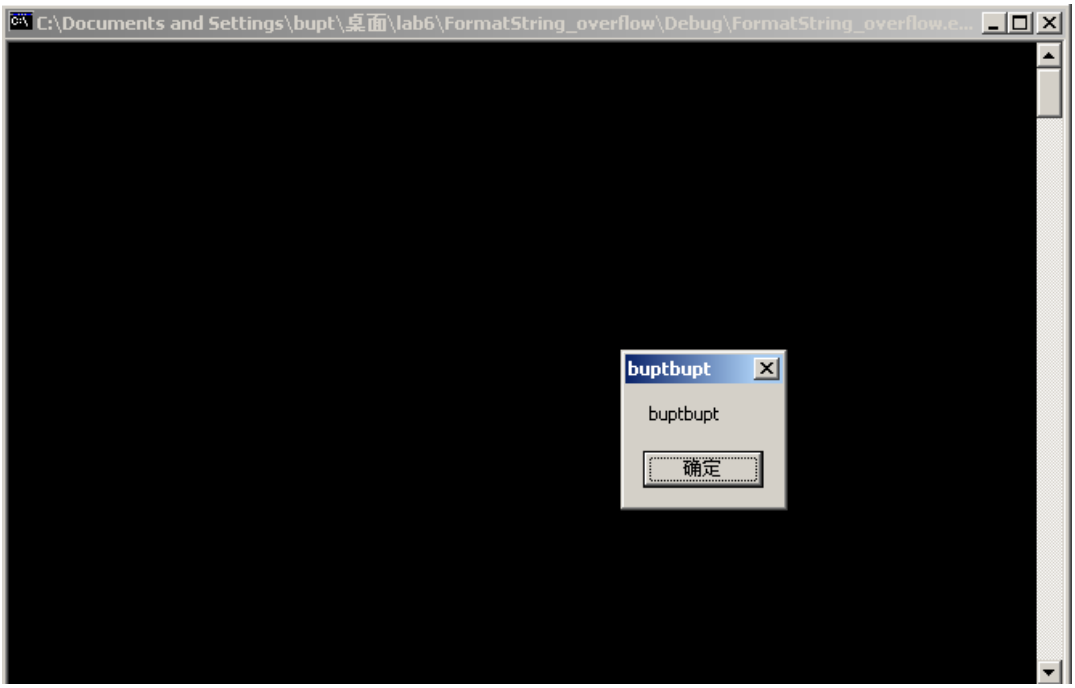
（7）第二次调用 sprintf()后

outbuf 起始地址为 0x0012FD2C，19 字节的字符串"ERR Wrong command:"后为 497 字节的整型数字 1245056，因此从 0x0012FF30 开始为\x39\x4a\x42\x00。下图可见成功将返回地址覆写为 shellcode 起始地址 0x00424A39。



（8）执行结果

继续执行，可以看到弹出对话框。



三、测试结论

格式化字符串漏洞是由于程序员在使用诸如 printf 这类的格式化输出函数时未能正确验证输入，导致攻击者可以通过构造特定的输入来控制格式化字符串的行为。如果格式化函数如 sprintf 或 printf 使用了用户控制的输入作为格式化参数，攻击者可能会插入恶意格式化指令，这些指令可以读取或写入内存，导致缓冲区溢出或其他安全问题，例如执行任意代码。这种类型的漏洞在软件安全中是严重且常见的，因为它允许攻击者绕过正常的权限检查来危害系统。

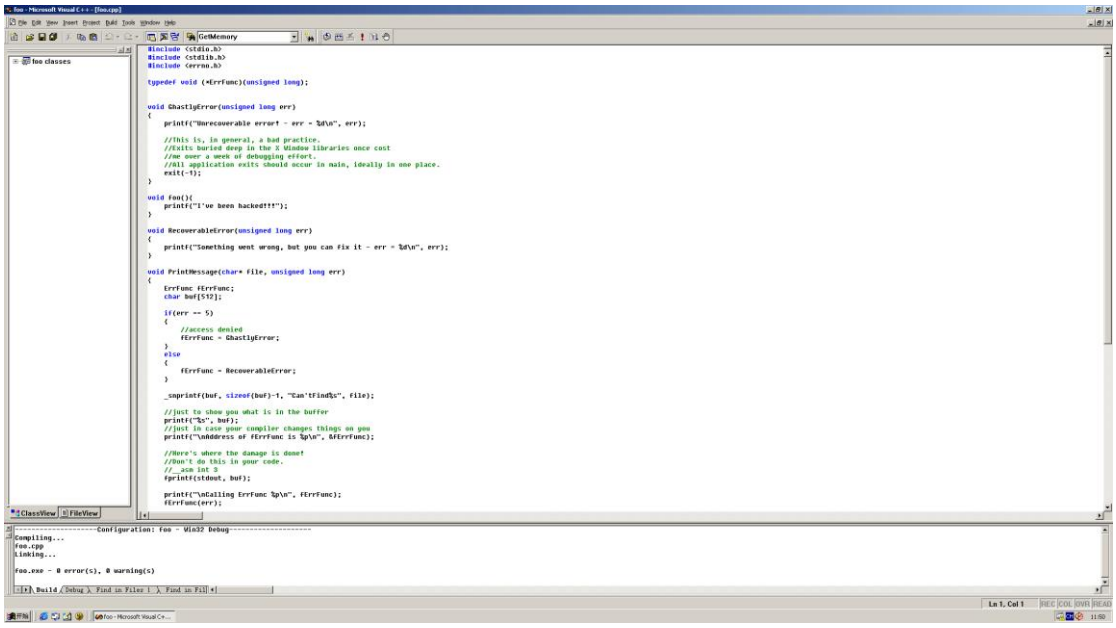
本次实验中了解到了格式化字符串造成缓冲区溢出的具体原理，并对其危害和效果进行了尝试。我意识到软件面临着极大的安全隐患，需要我们去保护和守卫。

四、思考题

破解 foo.exe 程序，在不改变源代码的情况下，要求通过命令行输入，利用格式化字符串漏洞 n%的方式，调用隐藏的 foo 函数，并尝试调用一个 shellcode。

1、分析代码

（1）查看代码



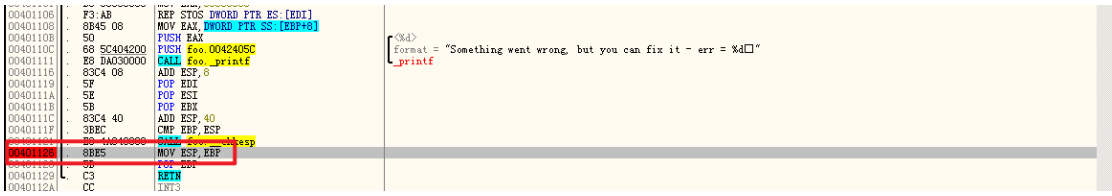
（2）分析代码

main 函数中根据命令行提供的参数打开对应的文件。如果这个文件不存在，那么就调用 PrintMessage 函数打印相应的错误信息。在 PrintMessage 函数中把错误分为 GhastlyError 和 RecoverableError 两类。要想调用 foo 函数，可以通过%n 把 fErrFunc 函数的地址修改为 foo 函数的地址。命令行参数为：%x%x…%x%x%n+fErrFunc 函数指针的地址。snprintf 之后 buf 为 Can'tFind%x%x…%x%x%n+fErrFunc 函数指针的地址，接下来由于 fprintf(stdout, buf)中缺少了 argument 参数，所以已打出的字符总数通过%n 被写入 fErrFunc 函数指针的地址。通过控制%x 调整已打出的字符总数就能达到我们的目的。

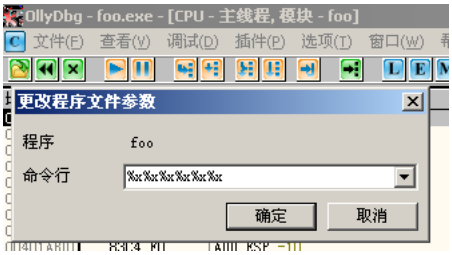
2、调试

在 OllyDbg 中打开程序。

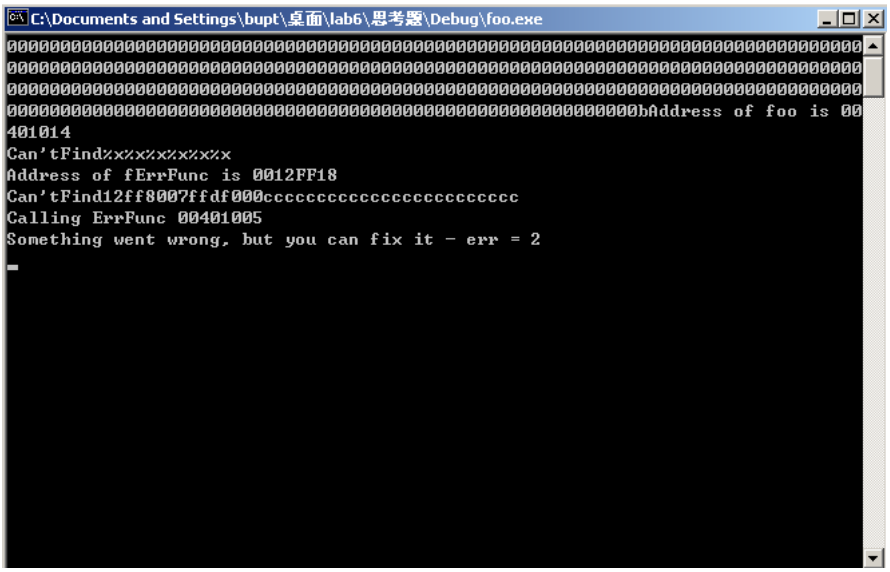
（1）在输出函数下断点。



（2）点击调试->参数，向程序传递命令行参数。首先尝试一串%x：%x%x%x%x%x%x%x



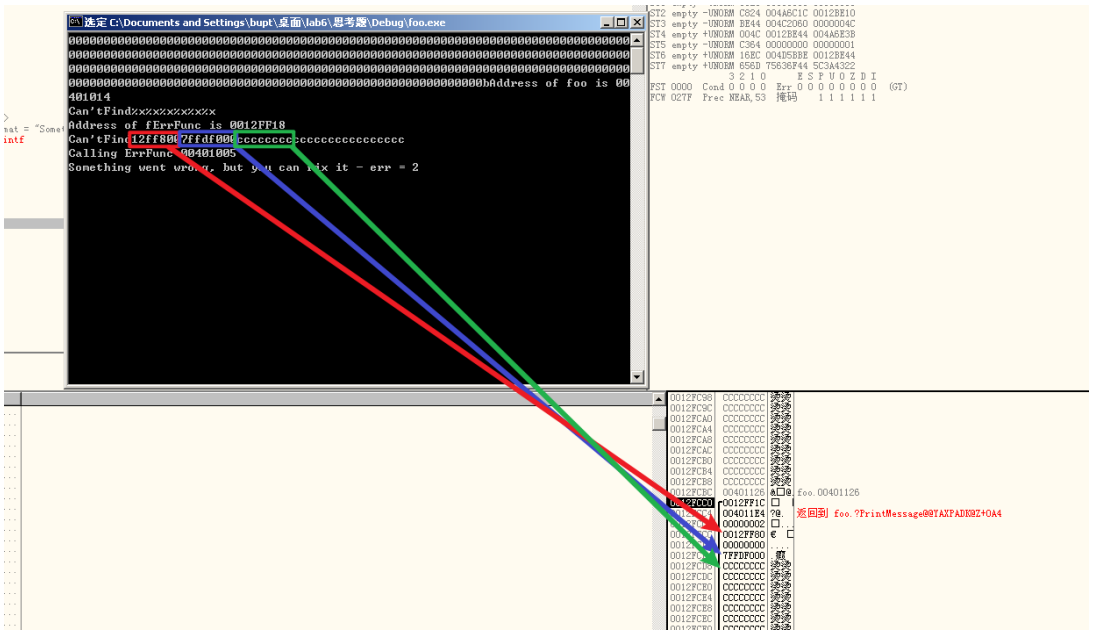
（3）输出结果



3、查看栈空间

可见存储从 0012FD18 开始，由于设定 char buf [512] 因此会一直存储到 0012FF18。但是_snprintf (buf, sizeof (buf)-1, “Can’tFind% s”, file); 所以无法实现对 0012FF18 的直接写入。

4、查看 % x 的指针调用位置



可见，指针是从 0012FCCC 开始调用的。

5、分析

如果想调用 foo 函数，需要在返回地址，即 0012FF18 写入 foo 函数地址 00401014，而现有转跳是 00401005。

因此可以使用 %hn 替换其低四位内容，即 %hn 输出前字符串输出 1014。并且使指针通过 %x 遍历，最后停止在 0012FF18 的地址上。

参数格式应为 %x%x%x%x%x……%x%x%x%x%hn \x00 \x12 \xFF \x18。

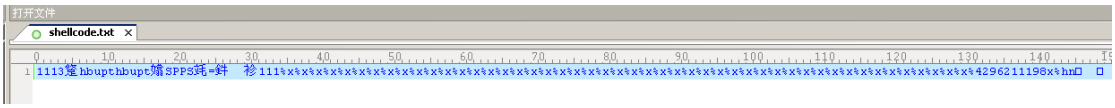
6、计算 %x 个数

1014 转换为 10 进制为 4116，由此构造输出位数，结果如下：

打开文件
1.txt
1111111111111%x%x%x%x%x%x

8、构造 shellcode

由于 shellcode 构造在参数部分，即 0012FD20。因此在转跳时需要转跳此处，需要把 0012FF18 处内容 00401005 全部替换。构造如下：


打开文件
shellcode.txt
1113%x%x%x