**江苏科技大学**

**实 验 报 告**

课 程： 入侵检测与入侵防御

课 题： 网络入侵

学 院： 计算机学院

学 号： 182210710119

姓 名： 陈四贵

班 级： 1822107101

指导老师： 张笑非

# 实验目的

1、了解常见的攻击原理；

2、掌握至少一种攻击，并实现。

# 实验原理

缓冲区溢出的含义是为缓冲区提供了多于其存储容量的数据，就像往杯子里倒入了过量的水一样。通常情况下，缓冲区溢出的数据只会破坏程序数据，造成意外终止。但是如果有人精心构造溢出数据的内容，那么就有可能获得系统的控制权！如果说用户（也可能是黑客）提供了水——缓冲区溢出攻击的数据，那么系统提供了溢出的容器——缓冲区。

缓冲区在系统中的表现形式是多样的，高级语言定义的变量、数组、结构体等在运行时可以说都是保存在缓冲区内的，因此所谓缓冲区可以更抽象地理解为一段可读写的内存区域，缓冲区攻击的最终目的就是希望系统能执行这块可读写内存中已经被蓄意设定好的恶意代码。按照冯·诺依曼存储程序原理，程序代码是作为二进制数据存储在内存的，同样程序的数据也在内存中，因此直接从内存的二进制形式上是无法区分哪些是数据哪些是代码的，这也为缓冲区溢出攻击提供了可能。

由于栈是低地址方向增长的，因此局部数组buffer的指针在缓冲区的下方。当把data的数据拷贝到buffer内时，超过缓冲区区域的高地址部分数据会“淹没”原本的其他栈帧数据，根据淹没数据的内容不同，可能会有产生以下情况：

1、淹没了其他的局部变量。如果被淹没的局部变量是条件变量，那么可能会改变函数原本的执行流程。这种方式可以用于破解简单的软件验证。

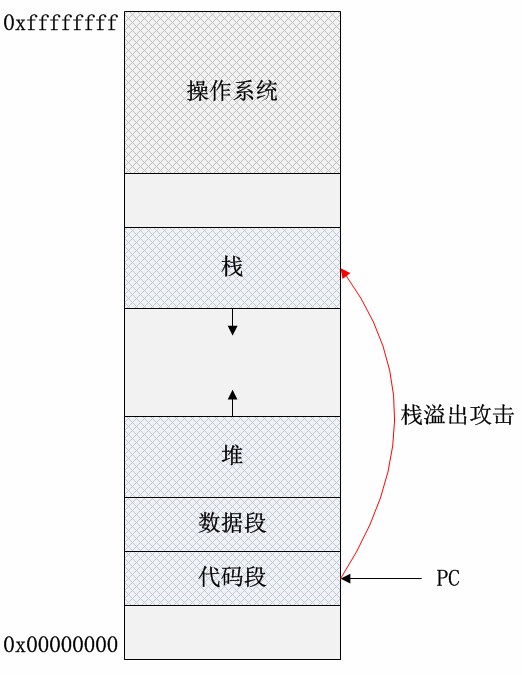
2、淹没了ebp的值。修改了函数执行结束后要恢复的栈指针，将会导致栈帧失去平衡。

3、淹没了返回地址。这是栈溢出原理的核心所在，通过淹没的方式修改函数的返回地址，使程序代码执行“意外”的流程！

4、淹没参数变量。修改函数的参数变量也可能改变当前函数的执行结果和流程。

5、淹没上级函数的栈帧，情况与上述4点类似，只不过影响的是上级函数的执行。当然这里的前提是保证函数能正常返回，即函数地址不能被随意修改（这可能很麻烦！）。

如果在data本身的数据内就保存了一系列的指令的二进制代码，一旦栈溢出修改了函数的返回地址，并将该地址指向这段二进制代码的其实位置，那么就完成了基本的溢出攻击行为。



# 实验环境

Windows PC机、visual studio 2017

# 实验步骤

运行下列代码，并分析缓冲区溢出原理

1. #include <iostream>
2. #include <string.h>
4. **void** foo01() {
5. **char** buff[16];
6. **char** Lbuffer[] = "01234567890123456789========ABCD";
7. strcpy(buff, Lbuffer);
8. }
10. **int** main()
11. {
12. foo01();
13. **return** 0x98;
14. }

# 实验结果

上述foo01()函数调用了strcpy()函数，系统会盲目地将data的全部数据拷贝到buffer指向的内存区域。buffer的长度是有限的，一旦data的数据长度超过BUF\_LEN，便会产生缓冲区溢出。在实际编译运行中，由于visual studio 2017自身编译环境的完善，我们必须将不安全的strcpy()函数换位strcpy\_s()函数，才能正常编译运行，由此，系统检测出了缓冲区长度不匹配的问题。但在某些不完善的编译器中，可以正常使用不安全的strcpy()函数，这不一定能被系统识别，很多时候会出现莫名的错误。我们可以利用这点，写一段大的缓冲区溢出函数，经过一定的伪装，将程序送入目标计算机系统执行，以此达到溢出的目的。

