

**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**实验名称： 缓冲区溢出攻击**

**院 系 ：计算机科学与技术**

**专业班级 ： 计算机 202201**

**学 号 ： U202215643**

**姓 名 ： 王国豪**

**指导教师 ： 张 勇**

**年 月 日**

**一、实验目的与要求**

通过分析一个程序（称为“缓冲区炸弹”）的构成和运行逻辑，加深对理论课中关于程序的机器级表示、函数调用规则、栈结构等方面知识点的理解，增强反汇编、跟踪、分析、调试等能力，加深对缓冲区溢出攻击原理、方法与防范等方面知识的理解和掌握；

实验环境：Ubuntu，GCC，GDB等

**二、实验内容**

**任务** 缓冲区溢出攻击

**程序运行过程中，需要输入特定的字符串，使得程序达到期望的运行效果。**

对一个可执行程序“bufbomb” 实施一系列缓冲区溢出攻击(buffer overflow attacks)，也就是设法通过造成缓冲区溢出来改变该程序的运行内存映像(例如将专门设计的字节序列插入到栈中特定内存位置)和行为，以实现实验预定的目标。bufbomb 目标程序在运行时使用函数 getbuf读入一个字符串。根据不同的任务，学生生成相应的攻击字符串。

实验中需要针对目标可执行程序bufbomb,分别完成多个难度递增的缓冲区溢出攻击(完成的顺序没有固定要求)。按从易到难的顺序，这些难度级分别命名为smoke (level 0)、fizz (level 1)、bang (level 2)、boom (level 3)和kaboom (level 4)。

**1、第0级 smoke**

正常情况下，getbuf函数运行结束，执行最后的ret指令时，将取出保存于栈帧中的返回（断点）地址并跳转至它继续执行（test函数中调用getbuf处）。要求将返回地址的值改为本级别实验的目标smoke函数的首条指令的地址， getbuf函数返回时，跳转到smoke函数执行，即达到了实验的目标。

**2、第1级 fizz**

要求getbuf函数运行结束后，转到 fizz函数处执行。与smoke的差别是，fizz函数有一个参数。 fizz函数中比较了参数val 与 全局变量cookie的值，只有两者相同（要正确打印val）才能达到目标。

**3、第2级 bang**

要求getbuf函数运行结束后，转到 bang 函数执行，并且让全局变量global\_value 与 cookie相同（要正确打印global\_value）。

**4、第3级 boom**

无感攻击，执行攻击代码后，程序仍然返回到原来的调用函数继续执行，使得调用函数（或者程序用户）感觉不到攻击行为。

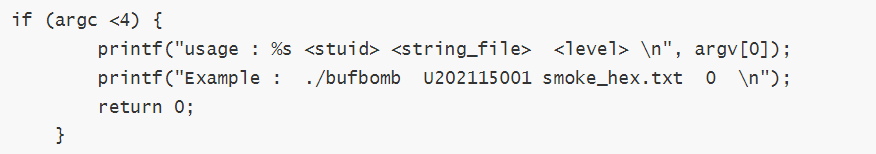
构造攻击字符串，让函数 getbuf将cookie值返回给 test函数，而不是返回值 1 。还原被破坏的栈帧状态，将正确的返回地址压入栈中，并且执行 ret 指令，从而返回到 test函数。

**三、实验记录及问题回答**

**（1）实验任务的实验记录**

**第0级 smoke：**

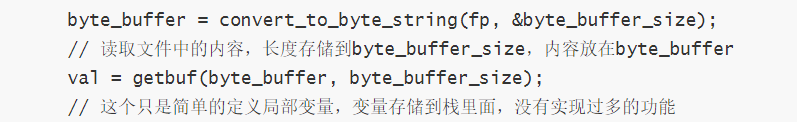
一些前置的知识要注意，当我们使用命令行传参进入函数以后，会对我们传参进入检查。



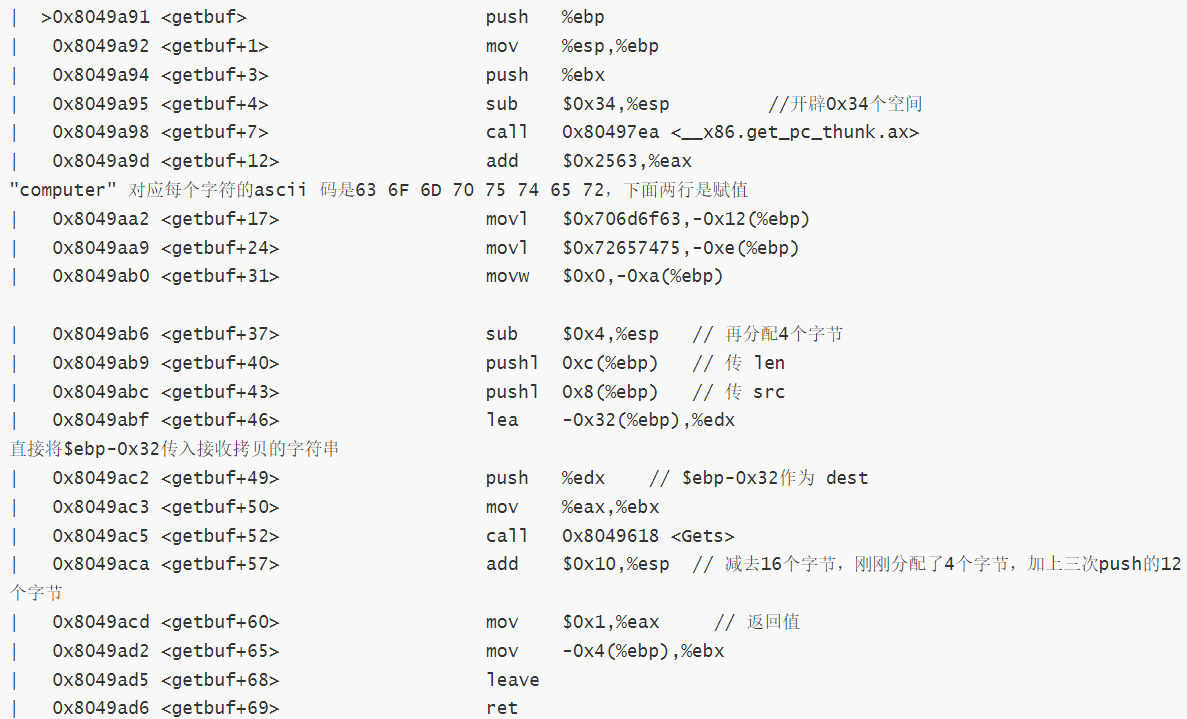
我们输入的参数的数量要大于4才符合，不然直接报错。

接着会把输入的学号，文件名，level进行检查，看是不是符合规范。

接着会进入initialize\_bomb 初始化，接着进入test(struct env\_info \*p)进入测试，这个函数里我们需要注意一下两个函数调用。

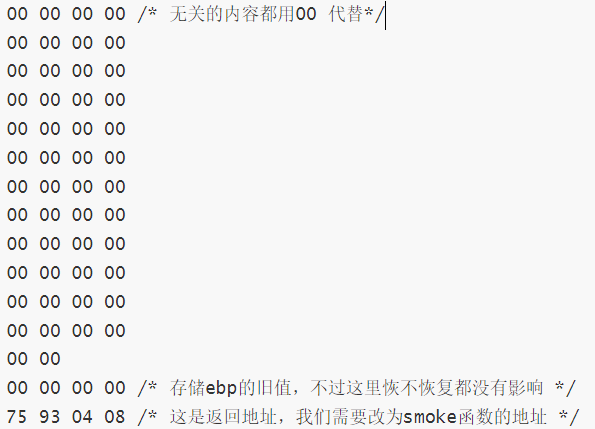


我们需要对getbuf函数额外关注。

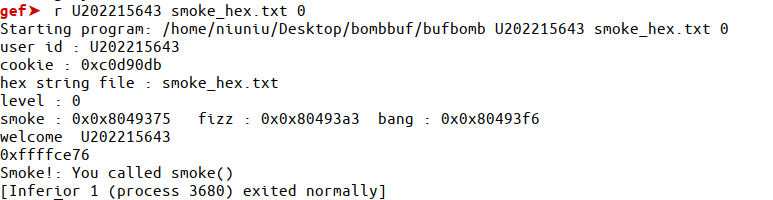


分析可知我们的smoke\_hex.txt里面需要从传入的dest开始的$ebp-0x32开始覆盖，需要把之前传入getbuf之前保存的返回地址给覆盖，这样退出getbuf的时候才能跳转到smoke函数。我们得知smoke函数的地址是0x08049375，注意我们是小端存储，需要把地址低字节放在低地址，并且我们需要注意我们还要把进入getbuf的时候push %ebp的值也要写在文件里面，以免ebp出错，虽然说实际上这个并不影响结果。

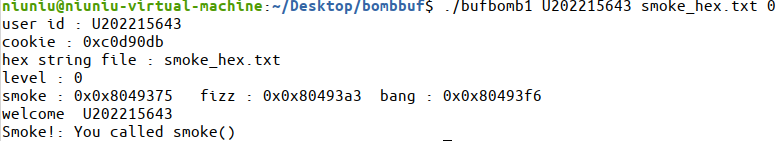
值得注意的是我们文件中数据的大小，我们传入$ebp-0x32的址为起始copy地址，这里起始地址和ebp距离为0x32，这里有50个字节，我们需要修改为ebp下面八个字节，所以我们文件里面需要表示58个字节。



图表 1 smoke\_hex.txt截图



图表 2 gdb内部运行截图



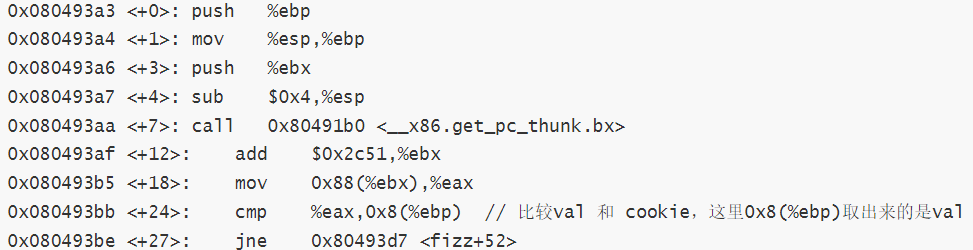
图表 3 命令行直接运行截图

**第1级 fizz：**

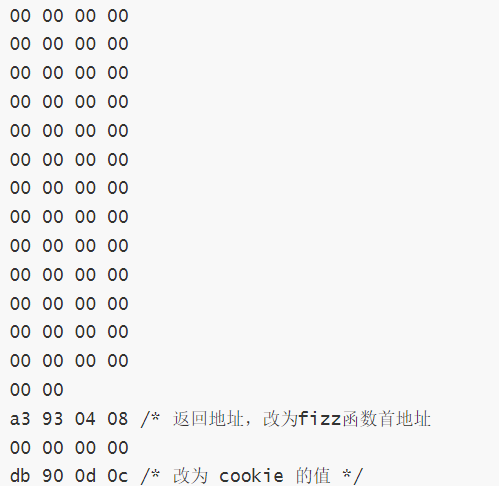
刚刚进入getbuf的时候ebp为0xffffcf98,mov %esp,%ebp后ebp变为0xffffcea8,注意我们传入Gets函数的被拷贝地址为$ebp-0x32,也就是0xffffce76。

刚刚进入Gets的时候ebp为0xffffcee8, mov %esp,%ebp后变为0xffffcea8,也就是说此时此时esp也是等于0xffffcea8，最后leave的时候esp的值会被还原为0xffffcea8

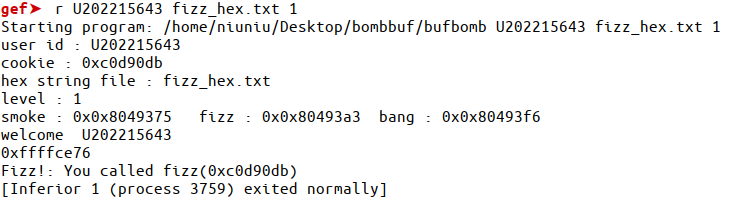
查看fizz中的汇编代码，分析可知，这里和ebp之前的值无关，我们并不需要在读取我们文件的时候恢复ebp值，因为我们的参数存储在刚刚进入的函数的$esp+0x4中，也就是存储在我们mov %esp, %ebp完后的$ebp+0x8中，我们只需要将$ebp+0x8即0xffffceb4的位置赋值为cookie的值就行了。



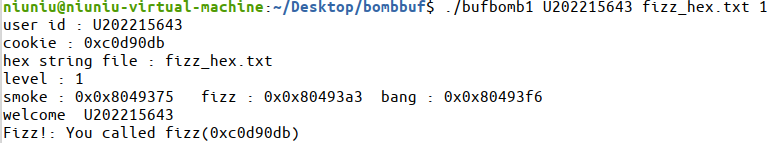
查看我们的cookie的值为0x0c0d90db。此时ebp+0x8为 0xffffceac+0x8 = 0xffffceb4，我们从文件中拷贝的字符会拷贝到起始地址0xffffce76的地方，用0x ffffceb4-0xffffce76为62，所以在我们文件第62到65个数值要写成cookie的值就行啦。



图表 4 fizz\_hex.txt 截图



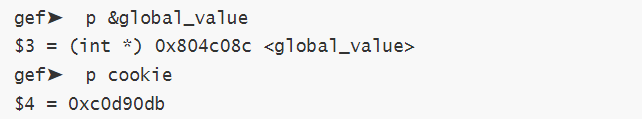
图表 5 gdb模式下运行截图



图表 6 命令行直接运行截图

**第2级 bang：**

我们首先要找到global\_value的地址和cookie的址。



我们需要修改文件中返回地址让我们能够跳转到bang函数，bang函数的地址是0x080493f6。

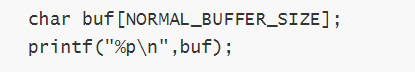
那么我们的bang.s要怎么写呢？

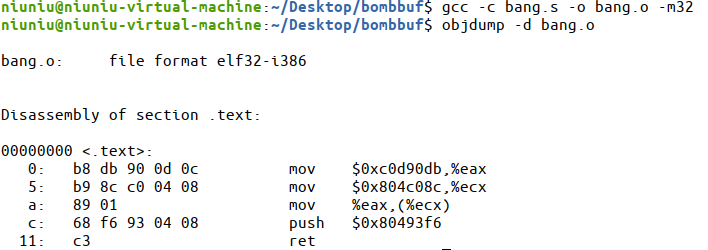
第一步就是实现把global\_value的值赋值为cookie，接着我们需要处理ret的问题。

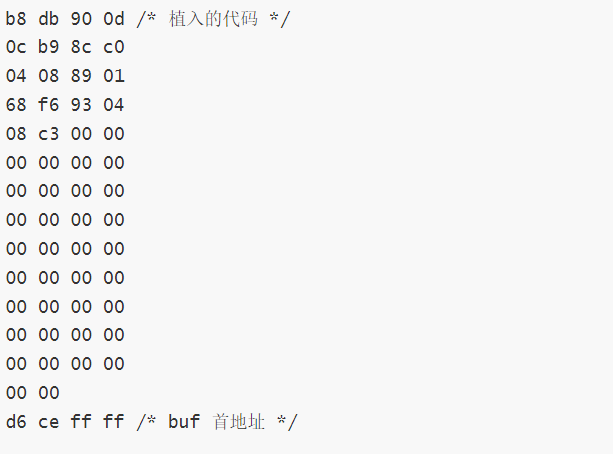
一种思路是找到要返回的esp指向地址，为0xffffceb0,和0xffffce76相减是58，所以在文件第58到61个数字写上这个地址。

另外一种是我们可以在修改global\_value的值后手动push返回地址，这样ret就可以跳转到bang。

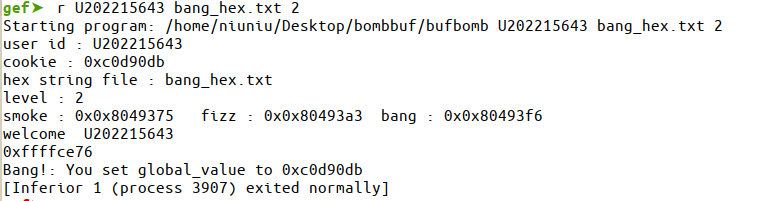
这里有问题，我们这里buf的首地址有一个坑。Gdb中调试的buf的首地址也就是$ebp-0x32，和没有gdb模式直接运行下的buf首地址是不一样的，关闭地址随机化也没有用。我的解决方案就是在源代码中加入一段输出，打印buf的地址，我们先获取buf的地址，再去修改我的bang.txt。这样我们就可以正常运行了。



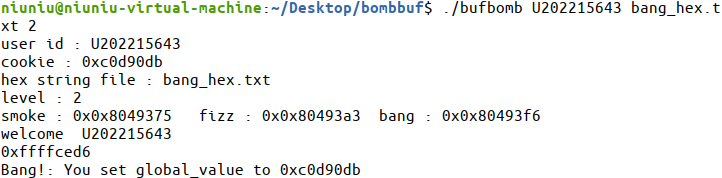




图表 7 bang\_hex.txt 截图



图表 8 gdb模式下运行截图



图表 9 命令行直接运行截图

**第3级 boom：**

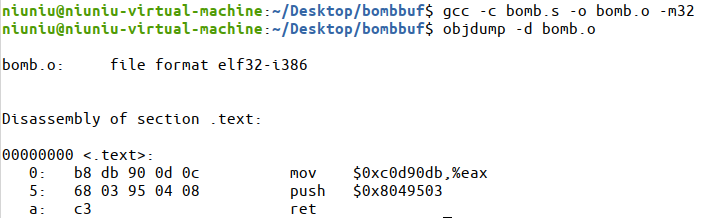
我们要完成指导cookie的值，把这个值给eax（因为eax存储返回值），我们还要恢复ebp的值，这个可以通过mov指令直接赋值，另外保证eip能够正常的话需要ret指令。

记录刚刚进入getbuf的ebp的值为0xffffcee8，push完ebp后esp的值为0xffffcea8,这个我们需要复原的。此时记录返回地址也就是$esp+0x4的值，为0x08049503，这个是需要ret的。还记录下cookie的值为0xc0d90db。

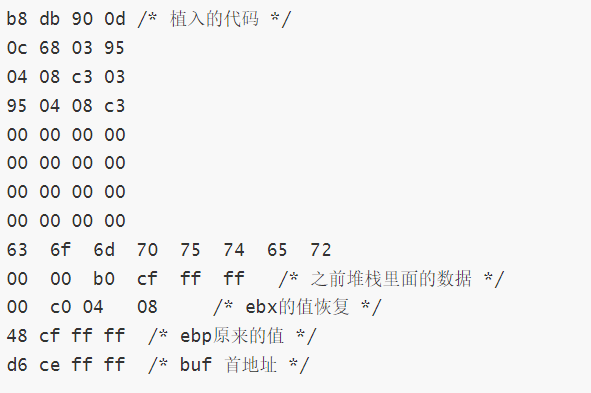
和上一关类似，我们写一个汇编代码放在buf开头，且结束Gets函数的时候跳转到buf的开头，开始执行我们预想的代码，buf的开头地址为0xffffce76。



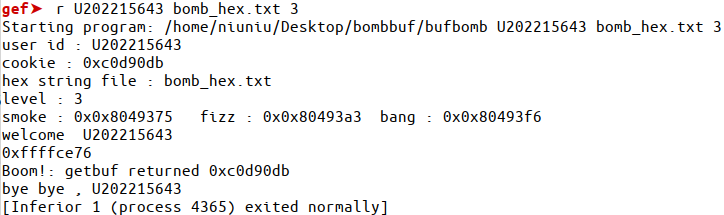
我们还需要注意一个问题，我们的文件里面还要还原堆栈，因为我们后续需要用到ebx的值，但是我们一开始在txt中将一些空间中的值置为0了，所以我们需要把堆栈中一些原来的数据恢复。下面我们来看一下我们的bomb.s文件怎么写。



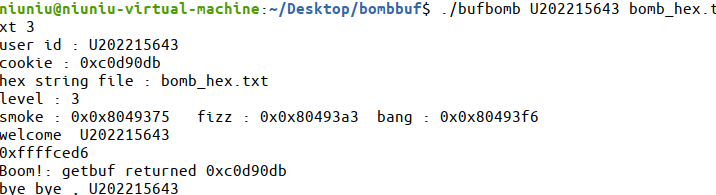
我们只需要在这里设定好返回值以及跳转的地址即可，至于ebp的值恢复以及ebx的设置我们直接放在我们的bomb\_hex.txt 中设置。



图表 10 bomb\_hex.txt 截图



图表 11 gdb模式下运行截图



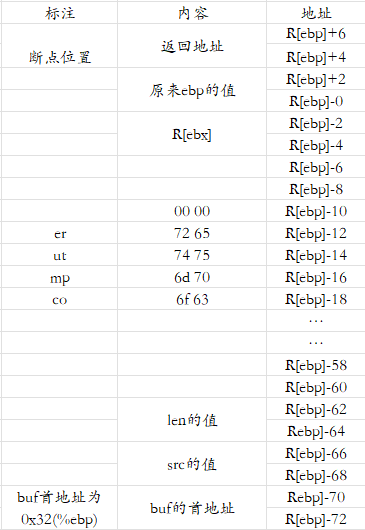
图表 12 命令行直接运行截图

**（2）缓冲区溢出攻击中字符串产生的方法描述**

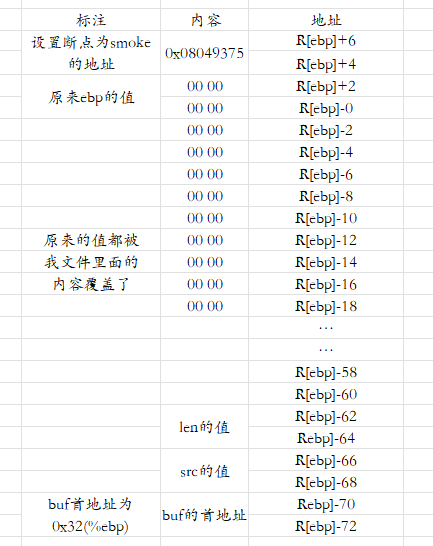
**要求：一定要画出栈帧结构 （包括断点的存放位置，保存ebp的位置，局部变量的位置等等）**

**第0级 smoke：**

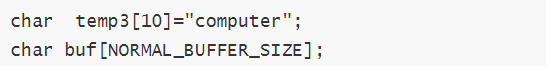
对我们程序起作用的就是getbuf函数的堆栈，所以我们只需要画出调用Gets函数前后的堆栈情况即可，这里不以具体的地址表示，用刚刚进入getbuf后新的ebp作为标准。



图表 13 未调用Gets前的堆栈



图表 14 调用Gets 后堆栈的情况



简单分析一下，堆栈中有temp3和buf两个局部变量，temp3的首地址是R[ebp]-18,buf的首地址是R[ebp]-50。我们还需要注意断点在R[ebp]+4的位置，ebp的存储位置在R[ebp]。

**第1级 fizz：**

这一关的堆栈情况和上一关基本一致，区别在于在R[ebp]+12的位置填入cookie的值，这个存放的位置由计算得来。

**第2级 bang：**

这一关相较与第一关，区别在于buf的存储空间，也就是R[ebp]-50开始的位置，填入了我们预先限号的机器码，剩下的空间才接着用00填充。

**第3级 boom：**

这一关中，堆栈的内容会有较大的改变，因为我们还需要还原堆栈，从R[ebp]-50开始的32个字节（因为buf开辟了32个字节）填写我们植入的机器码，接着从R[ebp]-18开始我们会恢复堆栈，因为R[ebp]-4存储的是ebx之前的值，接着断点位置，ebp原来的值的位置都没有改变。

**四、体会**

在进行缓冲区溢出实验的过程中，我获得了宝贵的经验和深刻的体会，以下是我的总结：

首先，我深刻认识到了安全漏洞的严重性。缓冲区溢出漏洞对计算机系统安全构成了巨大威胁，可以导致执行恶意代码、获取敏感信息甚至控制系统。通过实验，我们仅仅利用了一个数组的输入溢出问题就导致了各种函数之间的跳转以及常量的被修改，这让我深刻意识到了这种漏洞的危害性，加强了我对这方面知识的学习和认识。

其次，我学习了缓冲区溢出的原理和技术，包括栈溢出、堆溢出等常见类型。深入了解攻击者利用溢出漏洞实施攻击的方法和手段，实践实现了一些常见的攻击方式，如smoke、fizz、bang、boom等。了解到这些攻击方式背后的原理和实现机制，有助于我将来面对攻击时能够更好地找到防御的方向，例如利用栈溢出来覆盖返回地址，实现程序流程控制。

我还掌握了调试和分析工具的使用方法，如GDB调试器、objdump反汇编工具等。这些工具帮助我深入理解程序的执行过程，分析攻击手段，并且使我能够成功自己编写一些简单的缓冲区溢出攻击代码。例如，通过GDB调试器，我可以观察程序栈的变化情况，从而找到溢出点，并利用此信息进行攻击代码的编写。

通过实验，我不仅提升了自己的安全意识和技术能力，还加强了对计算机系统安全的认识和理解。加强了团队合作和沟通，与同学们共同合作、相互交流学习，共同解决问题。这些经验对我未来的学习和工作产生了积极的影响，使我能够更好地应对安全挑战和保护系统安全。