**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： 罗秋明**

**报告人： 刘俊楠 学号： 2017303010 班级： 01班**

**实 验 时 间： 2021/06/02**

**实验报告提交时间： 2021/06/02**

**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. Linux 64位操作系统
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**

**因为本次实验用到的可执行文件是32位，而实验环境是64位的，需要先安装一个32位的库，在root权限下安装如下所示（图1-1）：**

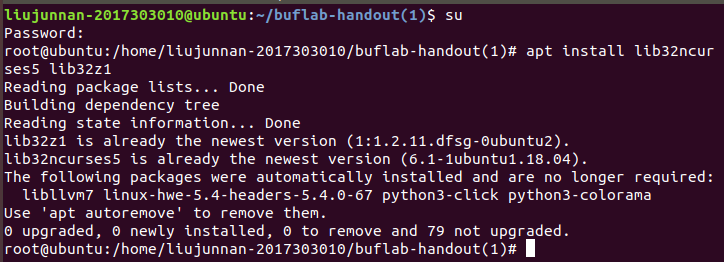


图1-1

**还需要安装sendmail（图1-2）**

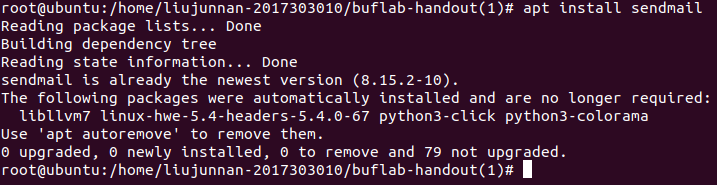


图1-2

首先利用反汇编命令查看getbuf函数的汇编代码，以便分析getbuf在调用<Gets>时的栈帧结构，汇编代码如下（图1-3）：

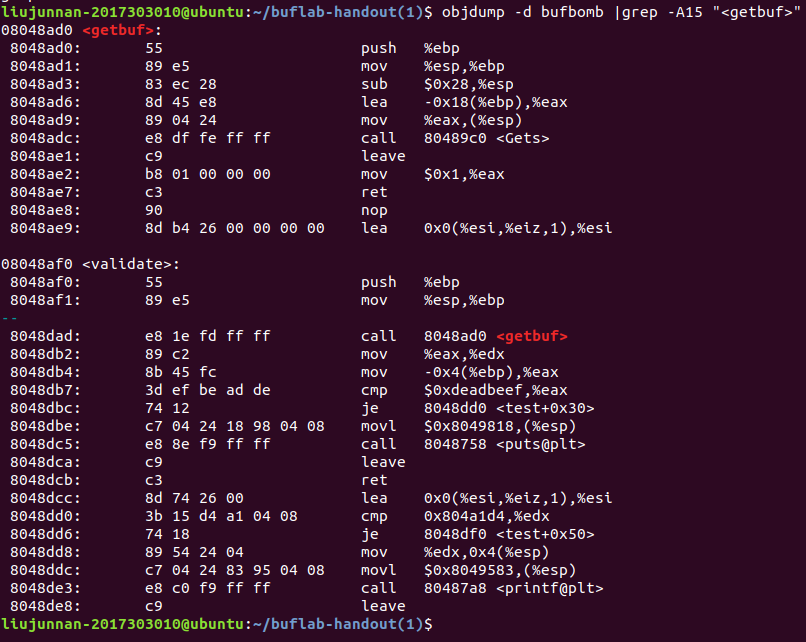


图1-3

**步骤1 返回到smoke()**

* 1. **解题思路**

本实验中，bufbomb中的test()函数将会调用getbuf()函数，getbuf()函数再调用gets()从标准输入设备读入字符串。

系统函数gets()未进行缓冲区溢出保护。其代码如下：

***int getbuf()***

***{***

***char buf[12];***

***Gets(buf);***

***return 1;***

***}***

我们的目标是使getbuf()返回时，不返回到test()，而是直接返回到指定的smoke()函数。

为此，我们可以通过构造并输入大于getbuf()中给出的数据缓冲区的字符串而破坏getbuf()的栈帧，替换其返回地址，将返回地址改成smoke()函数的地址。

* 1. **解题过程**

通过分析getbuf函数反汇编（图1-2-1）可知，getbuf函数为使用gets函数预留了24byte空间给我们输入string，但是其不检测读取string的长度，从而导致我们的输入可以破坏该方程的返回地址，从而达到破坏的目的。

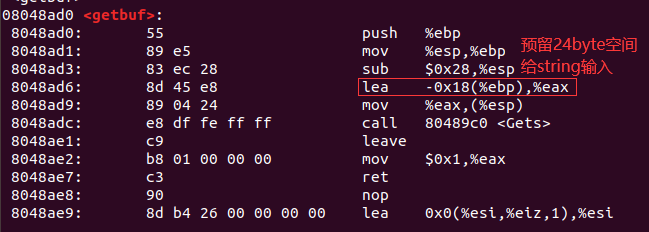


图1-2-1

根据图1-2-1的预留空间指令的地址0x8048ad6，我们在gdb时设置断点并写入自己的姓名作为特定cookie，然后进入汇编调试查看预留缓冲区首地址，由图1-2-2所示，可知预留缓冲区首地址为0xffffbe80.

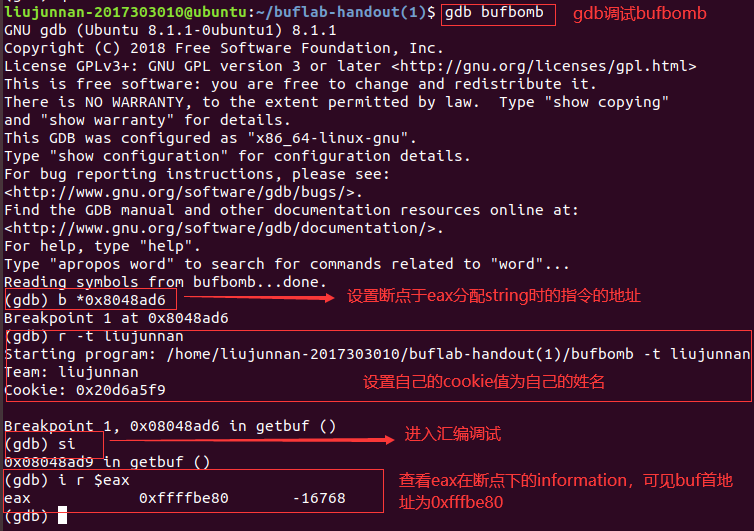


图1-2-2

在32位机器中，一般来讲函数调用过程的栈帧结构如图1-2-3所示，我们只用观察ebp距离函数调用后eax返回的相对位置，即可确定填写地址的位置，通过上述分析我们知道，string只有24byte，再加上图1-3中0x8048db4地址，即可确定ebp与eax只相差4byte，分析可得只需填写28byte即到达return address的起始地址，并且在28-32byte为return address的具体值。

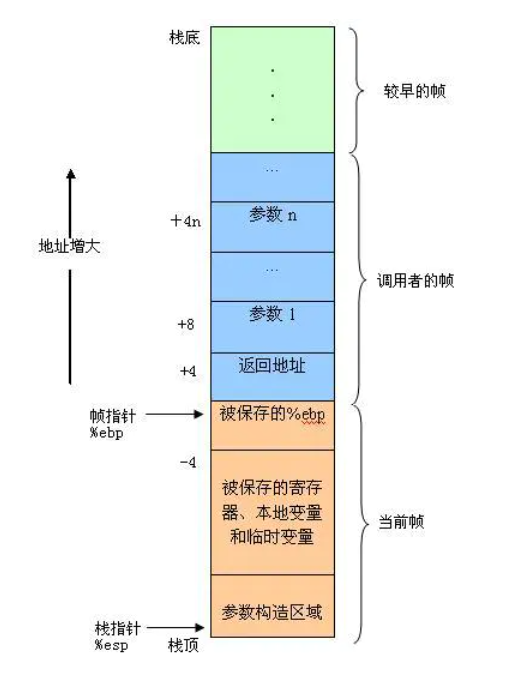


图1-2-3

因此，我们查看smoke函数地址（图1-2-4），并将其与28byte数字合并成为如下（小端表示）：

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 b0 8e 04 08

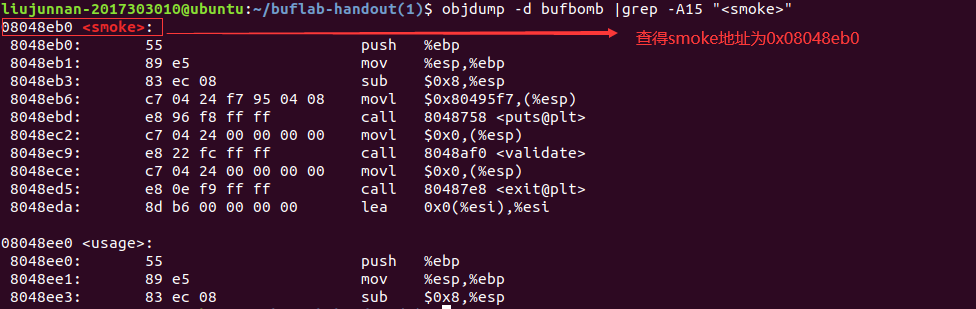


图1-2-4

如图1-2-5，在0.txt中加入上述数字串，后用sendstring将其转换成string 再运行bufbomb 名字为liujunnan cookie为0x20d6a5f9。结果显示成功调用smoke函数，成功。

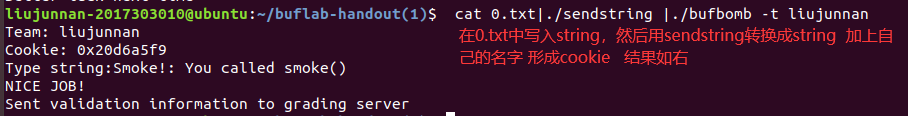


图1-2-5

**1.3 最终结果截图**

结果如图1-3-1所示。

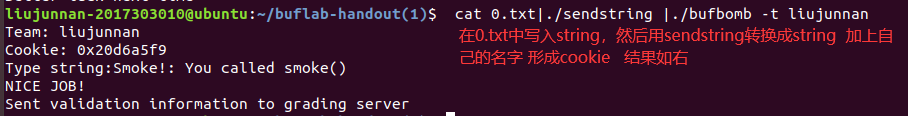


图1-3-1

**步骤2 返回到fizz()并准备相应参数**

**2.1解题思路**

本实验中，bufbomb中的test()函数将会调用getbuf()函数，getbuf()函数再调用gets()从标准输入设备读入字符串。

系统函数gets()未进行缓冲区溢出保护。其代码如下：

***int getbuf()***

***{***

***char buf[12];***

***Gets(buf);***

***return 1;***

***}***

我们的目标是使getbuf()返回时，不返回到test()，而是直接返回到指定的smoke()函数。

为此，我们可以通过构造并输入大于getbuf()中给出的数据缓冲区的字符串而破坏getbuf()的栈帧，替换其返回地址与参数，将返回地址改成fizz()函数的地址与一个黑客cookie值的参数。

**2.2解题过程**

通过分析jizz函数反汇编（图2-2-1）可知，fizz函数初始地址为0x08048e60，由第一题得在输入string时28-32byte填写地址，32-40byte填写参数。

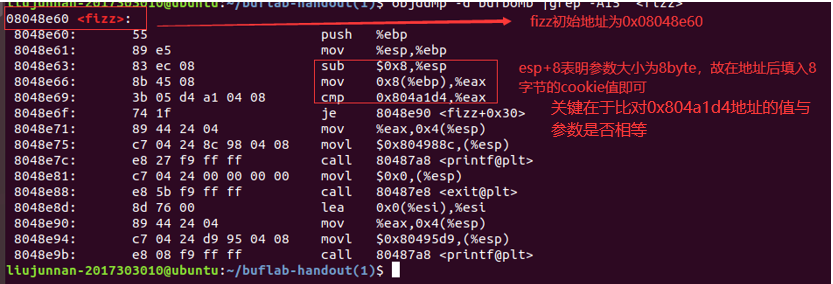


图2-2-1

根据图2-2-2的查找名字cookie的值，我们可以确定第一个参数(图2-2-3)，再进入0.txt编写时与第一问类似，前面0-28字节为任意值28-32为jizz地址值，32-40为cookie值也就是参数值，具体如下：

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 f9 a5 d6 20



图2-2-2

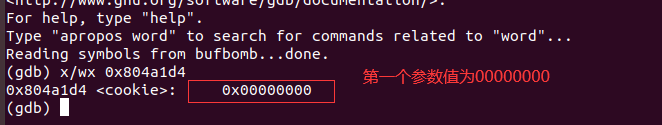


图2-2-3

分析可知，在比较返回地址后四字节是否为00000000，然后才能传入我们的cookie值，这是本题的关键，将上述数字串填入0.txt后，执行命令返回正确结果（图1-2-4）。

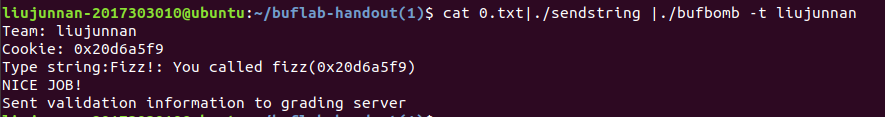


图2-2-4

**2.3 最终结果截图**

结果如图2-3-1所示。

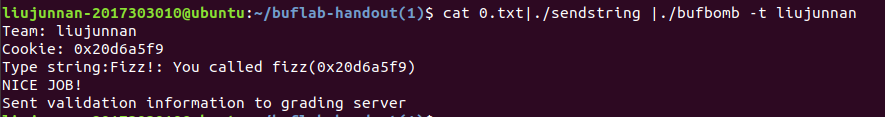


图2-3-1

**步骤3 返回到bang()且修改global\_value**

**3.1解题思路**

本实验中，bufbomb中的test()函数将会调用getbuf()函数，getbuf()函数再调用gets()从标准输入设备读入字符串。

系统函数gets()未进行缓冲区溢出保护。其代码如下：

***int getbuf()***

***{***

***char buf[12];***

***Gets(buf);***

***return 1;***

***}***

我们的目标是使getbuf()返回时，不返回到test()，而是直接返回到指定的smoke()函数。

为此，我们可以通过构造并输入大于getbuf()中给出的数据缓冲区的字符串而破坏getbuf()的栈帧，替换其返回地址与参数，将返回地址改成bang()函数的地址并且修改全局变量global\_value为我的黑客cookie值。

**3.2解题过程**

通过分析bang函数反汇编（图3-2-1）可知，bang函数初始地址为0x08048e10,分析代码可知要实现成功调用bang，必须实现global\_value与cookie相等，否则失败。

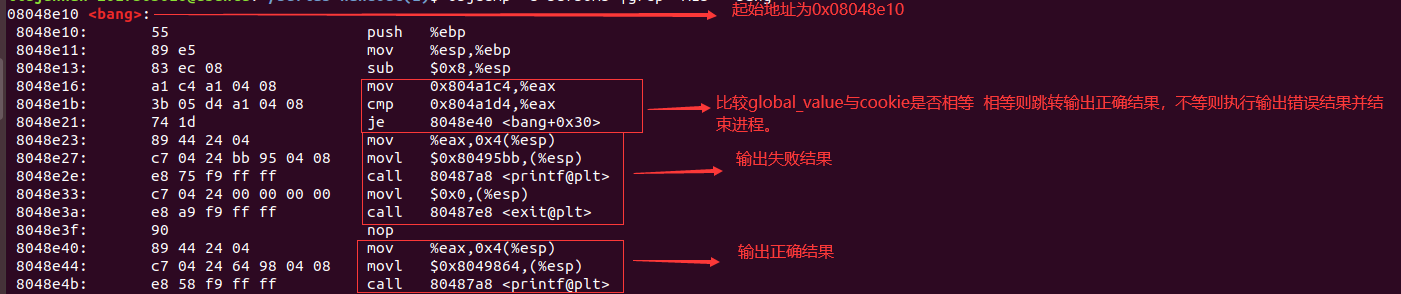


图3-2-1

根据图3-2-2的查找名字cookie的值，我们可以确定要修改的值为0x20d6a5f9,然后使用objdump -D bufbomb | less指令，找到global\_value与cookie的初始值与地址（图3-2-3），要更改global\_value就要 cookie。



图3-2-2

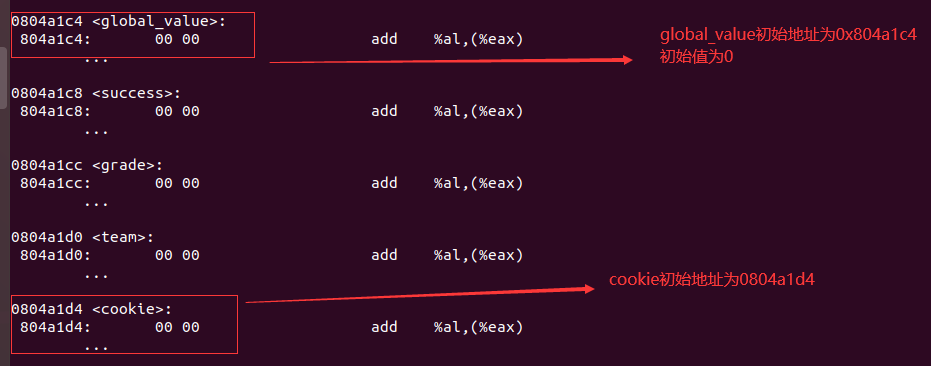


图3-2-3

分析可知，我们需要构造自定义攻击指令，在创建bang.s后填入以下指令：

Movl $0x20d6a5f9, 0x 0804a1c4 //另global\_value赋值为cookie值

Push $0x08048e10 //将esp指向bang函数,使得ret能返回bang函数

Ret //返回bang函数

使用gcc -m32 -c bang.s 将上述代码生成为机器码bang.o，再使用objdump -d bang.o 观察机器码如下（图3-2-4）：



图3-2-4

由图1-2-2可知，buf缓冲区首地址为0xffffbe80，且大小为0x18即24字节，再加上4字节ebp，即28-32为填充buf缓冲区首地址（图3-2-5）,拼接如下：

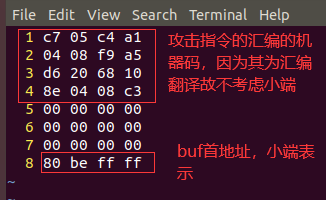
c705c4a10408f9a5d62068108e0408c300000000000000000000000080beffff

图3-2-5

**3.3 最终结果截图**

结果如图3-3-1所示。

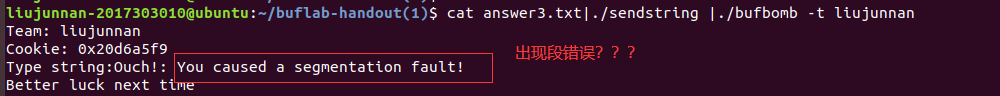


图3-3-1

在严格的检查以及在不同电脑上实验之后，我确定不是我的步骤出了问题，最后，经过上网的咨询与询问，我明白了出现段错误是因为**Linux系统默认开启了栈保护机制，用于阻止缓冲区溢出攻击**！！

于是，我尝试安装execstack，执行sudo apt-get install execstack（图3-3-2），然后执行execstack -s bufbomb修改堆栈的可执行属性。失败。

最后，我明白了是地址随机化的问题，于是遵循老师ppt的指令（echo “0” > /proc/sys/kernel/randomize\_va\_space）但也并未成功（图3-3-3），最后我查阅网上资料，运行sysctl -w kernel.randomize\_va\_space=0 ，最后成功（图3-3-4）。

关闭地址随机化之后，执行第三关指令，最后成功！（图3-3-5）

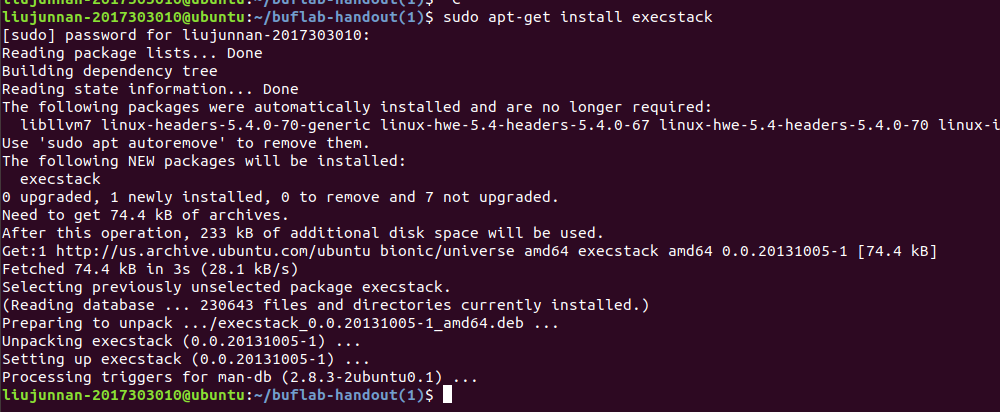


图3-3-2



图3-3-3

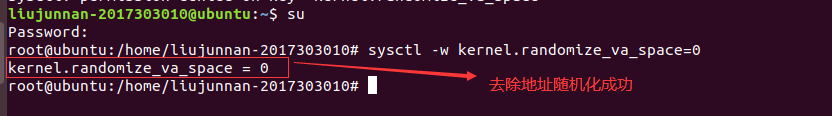


图3-3-4

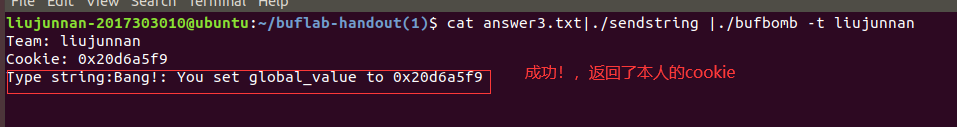


图3-3-5

**五、实验总结与体会**

**实验总结：**

实验中通过三个关卡考察了栈帧结构的基本原理，缓冲区溢出安全在汇编语言层面的知识点，主要运用了gdb里面的si i与r指令，通过定位相应函数位置并阅读其反汇编代码，应用gets的漏洞，进行相应的操作，最后实现缓冲区溢出攻击的操作。

**体会：**

在这个实验中我学习到了许多：

1. 在栈帧结构中，最低端为ebp，上面才是返回地址，在上面是参数表。
2. linux的随机地址分配对简单的缓冲区溢出攻击做了很好的防范，使得linux安全性更高
3. 通过这个实验我熟练了gdb的调试，以及学会使用grep查找文件里符合条件的字符串，受益匪浅。

|  |
| --- |
| **指导教师批阅意见：**  **成绩评定：**  指导教师签字： 刘刚    2018年 6月1日 |
| 备注： |