

**课 程 实 验 报 告**

**课程名称： 计算机系统基础实验**

**专业班级： CS1705**

**学 号： U201714726**

**姓 名： 王明明**

**指导教师： 胡侃**

**报告日期： 2019.6.3**

**计算机科学与技术学院**

目 录

[实验三 缓冲区溢出攻击 3](#_Toc10487882)

[3.1 实验概述 3](#_Toc10487883)

[3.2 实验过程 3](#_Toc10487884)

[2.2.1 阶段一 字符串比较 3](#_Toc10487885)

[2.2.2 阶段2 循环 5](#_Toc10487886)

[2.2.3 阶段三 条件/分支 7](#_Toc10487887)

[2.2.4 阶段四 递归调用和栈 9](#_Toc10487888)

[2.2.5 阶段五 指针 12](#_Toc10487889)

[2.2.6 阶段六 链表 13](#_Toc10487890)

[2.2.7 阶段七 隐藏阶段 16](#_Toc10487891)

[2.3 总结 18](#_Toc10487892)

# 实验三 缓冲区溢出攻击

# 3.1 实验概述

本实验的实验目的是加深对IA-32函数调用规则和栈结构的具体理解。本实验的实验内容是对一个可执行程序“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 冲区溢出来改变该可执行程序的运行内存映像，继而执行一些原来程序中没有的行为，例如将给定的字节插入到其本不应该出现的内存位置等。

本实验需要你构造5个攻击字符串，对目标程序bufbomb分别实施5次缓冲区溢出攻击。5个难度等级分别命名为Smoke（level 0）、 Fizz（level 1）、Bang（level 2）、Boom（level 3）和 Nitro（level 4），其中Smoke级最简单而Nitro级最困难。

实验环境： C语言；linux 

实践技能： 熟练运用gdb、objdump、gcc等工具。

# 3.2 实验过程

设法通过造成缓冲区溢出来改变可执行程序的运行内存映像，继而执行一些原来程序中没有的行为。

## 2.2.1 Level0 smoke

1. 任务描述： 构造一个攻击字符串作为bufbomb的输入，而在 getbuf()中造成缓冲区溢出，使得getbuf()返回时不是返回到 test函数继续执行，而是转向执行smoke。

2. 实验设计： 首先根据反汇编代码求得buf离返回地址的字节距离，确定要填充几个字节，然后找到smoke函数的起始地址，把这个地址填入buf末尾，即可实现这个功能。

3. 实验过程： 首先通过objdump –xDs bufbomb > asm.txt指令对bufbomb文件进行反汇编，查看getbuf的汇编代码如图3-1所示：

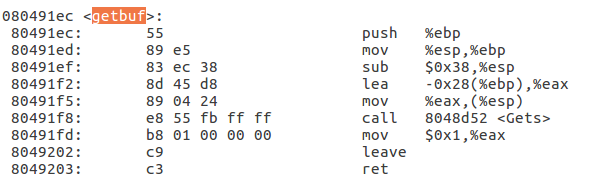


图3-1 getbuf反汇编代码

可见lea把buf的指针地址-0x28(%ebp)传给了Gets()，所以buf离返回地址有0x28+4==44个字节的距离，因此只需要buf处开始填充44个字节的非’\n’数据，接下来填入四个字节的要返回的地址，因为要返回至smoke，查看smoke对应的地址如图3-2所示：

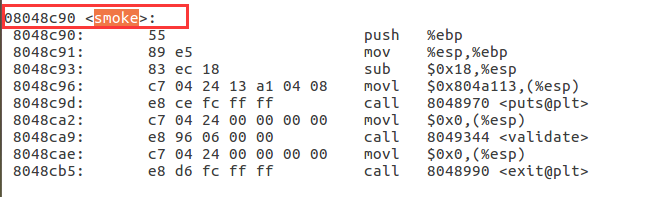


图3-2 smoke对应的地址截图

可见smoke函数的起始地址为0x08048c90，所以最后四个字节应填入90 8c 04 08，所以最后的48个字节的攻击字符串如图3-3所示：

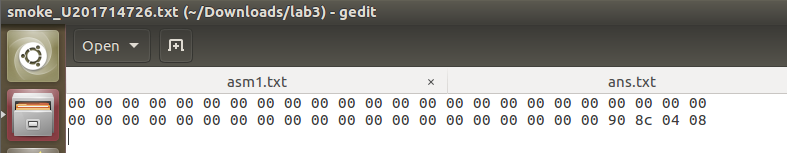


图3-3 Level0攻击字符串数据

将上述字符数据填入smoke\_U201714726.txt文档中，输入指令执cat smoke\_ U201714726.txt |./ hex2raw | ./ bufbomb –u U201714726执行 bufbomb，观察结果，如图3-4所示：

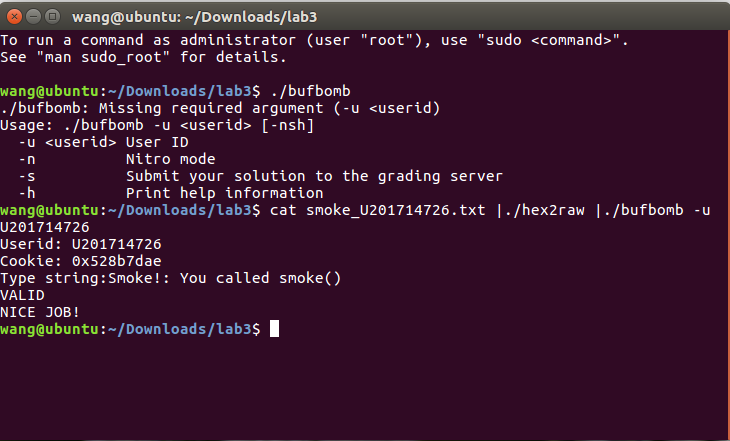


图3-4 结果截图

可见操作成功。

## 2.2.2 Level1 fizz

1. 任务描述：

构造一个攻击字符串作为bufbomb的输入，在getbuf()中造成缓冲区溢出，使得本次getbuf()返回时不是返回到test函数继续执行，而是转向执行fizz()。

与Smoke阶段不同和且较难的地方在于：fizz函数需要一个输入参数，因此你要设法将cookie值作为参数传递给fizz 函数，以便于fizz中val与cookie的比较能够成功。所以,你需要仔细考虑将cookie放置在栈中什么位置。

2. 实验设计： 跟Level0类似，但是多了一个比较cookie的环节，所以应该把cookie填入正确的地址。

3. 实验过程： 得到cookie，并把cookie填入正确的地址，与Level0类似，只是将要执行的函数从smoke换成fizz。

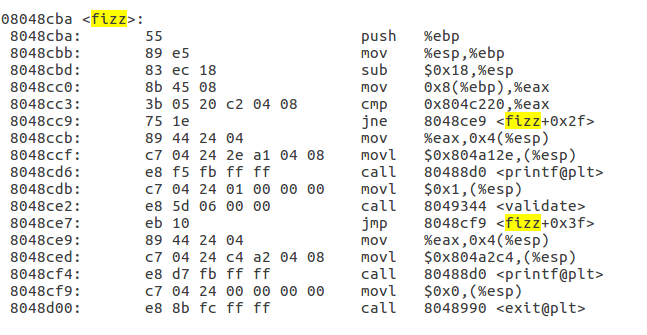


图2-5 fizz反汇编代码段截图

可见val变量存储地址为fizz函数中的0x8(%ebp)，而fizz函数开始的地址为0x08048cba，所以输入的前44个字节为非’\n’的任意数据，第45-48个字节存放fizz函数的起始地址，即ba 8c 04 08，接下来的4个字节也是非’\n’数据，最后为cookie的值，通过makecookie制造自己的cookie的值，如图3-6所示：

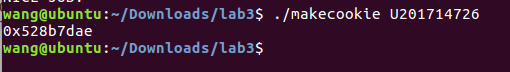


图3-6 生成的cookie值截图

于是攻击字符串为：

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ba 8c 04 08 00 00 00 00 ae 7d 8b 52

将上述攻击字符串存储在fizz\_U201714726.txt文件中，输入指令cat fizz\_ U201714726.txt |./ hex2raw | ./ bufbomb –u U201714726执行程序进行测试，测试结果如图3-7 所示：

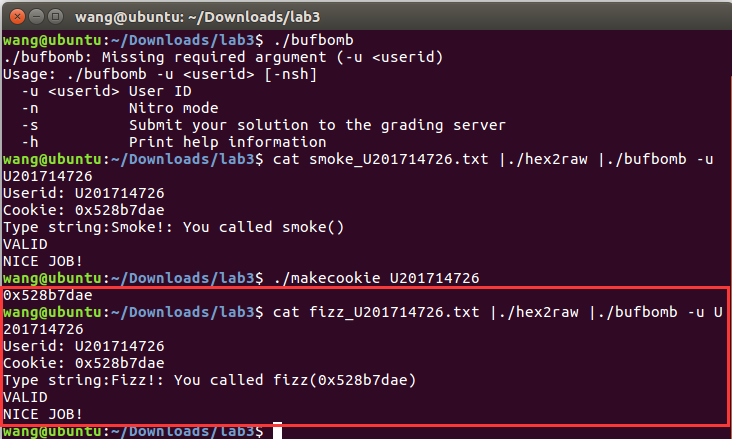


图3-7 Level1测试截图

可知Level1测试成功。

## 2.2.3 Level2 bang

1. 任务描述： 本阶段的任务是设计包含攻击代码的攻击字符串，所含攻击代码首先将全局变量global\_value设置为你的cookie值，然后转向执行bang()。

2. 实验设计：

本实验中，包含机器指令的攻击字符串在覆盖缓冲区时写入函数的栈帧，当被调用函数返回时，将转向执行这段攻击代码。

攻击代码要实现：

1）将全局变量global\_value设置 为自己的cookie值；

2）将bang函数的地址压入栈中；

3）附一条ret 指令。还要做的是设法将这段攻击代码置入栈中且将返回地址指向这段代码。

3. 实验过程：

bang ()函数的功能大体和fizz()类似，但val没有被使用，而是一个全局变量global\_value与cookie进行比较，这里global\_value 的值应等于对应自己的cookie才算成功，所以要想办法将全局变量 global\_value设置为自己的cookie值。

要想构造攻击代码，可以手工编写这些指令的二进制字节编码，或者可以首先编写一个汇编代码文件asm.s，然后使用gcc将该文件编译成机器代码，gcc命令格式： gcc –m32 –c asm.s 然后再使用“objdump –d asm.o”命令将其反汇编，从中可获得需要的二进制机器指令字节序列。

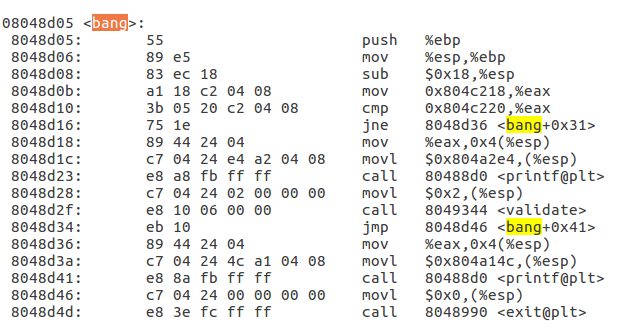


图3-8 bang反汇编代码段截图

Bang函数的首地址是0x08048d05，在bang 函数中，会将全局变量global\_value和cookie的值进行比较，global\_value的地址是0x804c218，cookie的地址是0x804c220，需要将global\_value的值修改为cookie 的值。因此可以编写汇编代码如图3-9所示，并将其保存在example.s文件中，使用指令：gcc -m32 -c example.s与objdump -d example.o进行汇编和反汇编生成example.o文件，如图3-10所示：

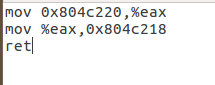


图3-9 example.s文件内容

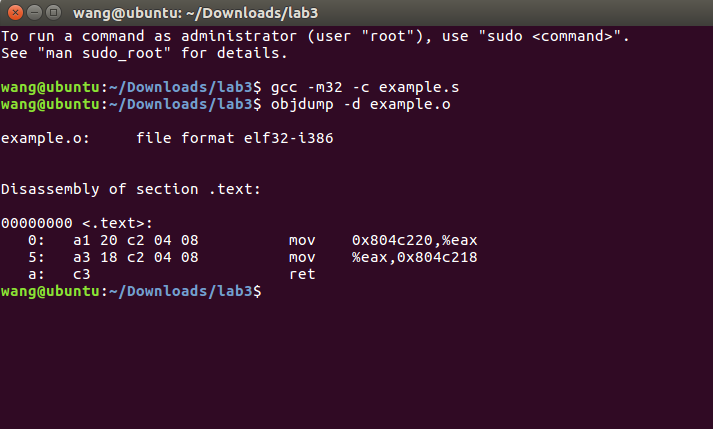


图3-10 example.o文件内容

得到指令序列：a1 20 c2 04 08 a3 18 c2 04 08 c3

此时设置间断点如图3-11所示：

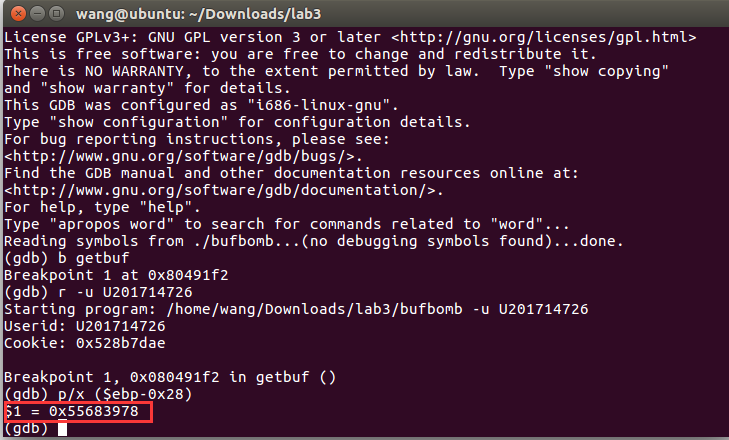


图3-11 buf首地址

由图3-11可知，buf的首地址为0x55683978，结合之前获得的指令序列，45-48字节应当存放buf首址，49—52字节存放bang函数首址，从而得到攻击字符串为：

a1 20 c2 04 08 a3 18 c2 04 08 c3 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 78 39 68 55 05 8d 04 08

将攻击字符串保存到bang\_U201714726.txt文件中，执行指令cat bang\_ U201714726.txt |./ hex2raw | ./ bufbomb –u U201714726，测试结果如图3-12 所示：

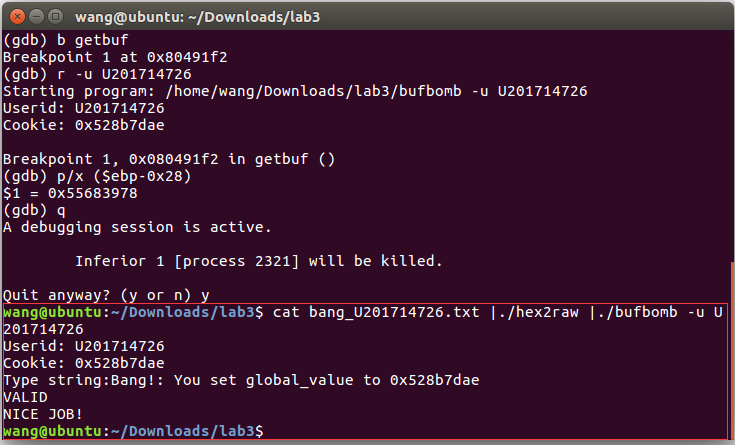


图3-12Level2 bang测试截图

可见结果正确。

## 2.2.4 Level3 boom

1. 任务描述：

本阶段的实验任务就是构造这样一个攻击字符串，使得 getbuf函数不管获得什么输入，都能将正确的cookie值返回给test函数，而不是返回值1。除此之外，攻击代码应还原任何被破坏的状态，将正确返回地址压入栈中，并执行ret 指令从而真正返回到test函数。

2. 实验设计： 返回test函数时的eax要赋值为cookie的值，还要恢复被覆盖的ebp的值。

3. 实验过程：

需要完成以下步骤：

（1）将攻击机器代码置入栈中

（2）设置return指针指向该代码的起始地址

（3）还原对栈状态的任何破坏。

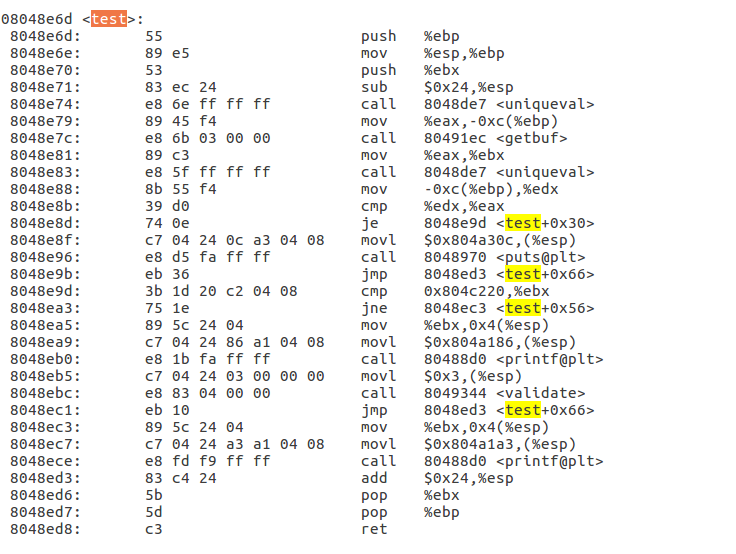


图3-13 test反汇编代码

要返回至test，应该返回到调用getbuf后的一步，地址为0x8048e81，返回的值应为cookie的值，可写出汇编代码并将其保存到cookie.s文件中，如图3-14所示：

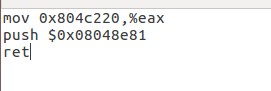


图3-14 cookie.s文件内容

使用指令gcc -m32 -c cookie.s和objdump -d cookie.o进行汇编和反汇编，得到序列如图3-15所示：

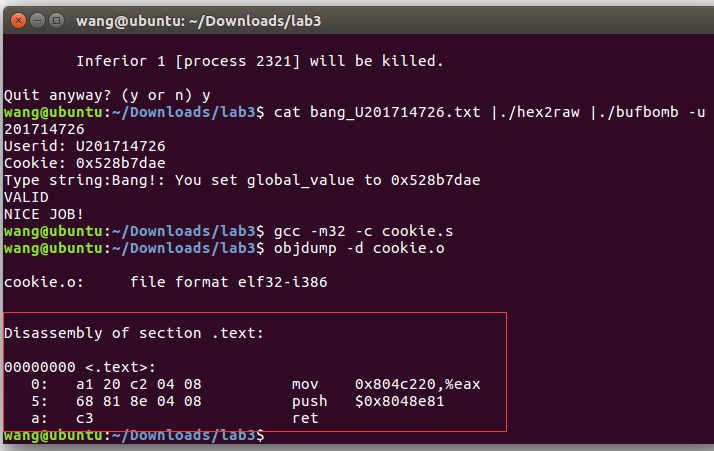


图3-15指令序列截图

得到的指令序列为a1 20 c2 04 08 68 81 8e 04 08 c3

覆盖getbuf返回地址的时候会覆盖保存的寄存器ebp的值，所以通过设置间断点来查看ebp的地址，如图3-16所示：

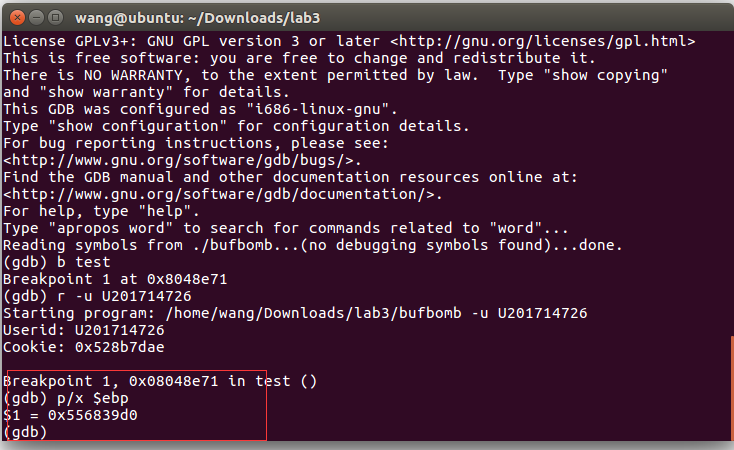


图3-16 ebp地址截图

所以得到的地址为0x556839d0

得到的指令序列放在前11个字节，41—44字节存储ebp地址，最后4个字节存放buf首址，得到最终的攻击字符串并将其保存在bomb\_U201714726.txt文件中：

a1 20 c2 04 08 68 81 8e 04 08 c3 00

00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 d0 39 68 55 78 39 68 55

使用指令cat boom\_ U201714726.txt |./ hex2raw | ./ bufbomb –u U201714726，测试结果如图3-17所示：

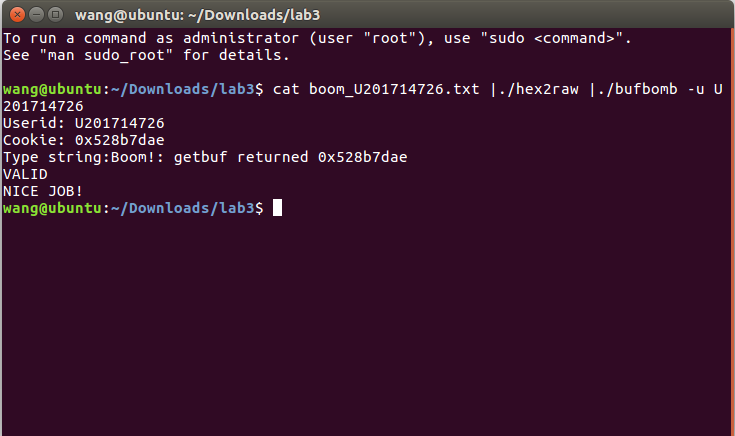


图3-17 Level3 boom测试截图

## 2.2.5 Level4 nitro

1. 任务描述：

与阶段四类似，构造一攻击字符串使得getbufn函数（注，在Nitro阶段，bufbomb将调用testn函数和getbufn函数,见bufbomb.c）返回cookie值至testn函数，而不是返回值1。 此时,需要将cookie值设为函数返回值，复原/清除所有被破坏的状态，并将正确的返回位置压入栈中，然后执行ret指令以正确地返回到testn函数。

2. 实验设计： 本次实验要求你要能较熟练地使用gdb、objdump、gcc，另外需要使用本实验提供的hex2raw、makecookie等工具。

3. 实验过程： 本阶段你需要使用“-n”命令行开关运行 bufbomb，以便开启Nitro模式，进行本阶段实验。与boom不同的是，本阶段的每次执行栈（ebp） 均不同，所以要想办法保证每次都能够正确复原栈被破坏的状态，以使得程序每次都能够正确返回到test。

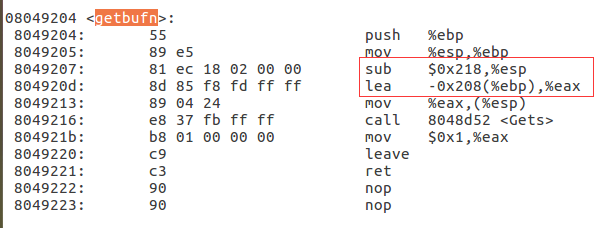


图3-18 getbufn截图

分析：字符串区域变大了很多，而且这次实验最大的特点是栈的位置不确定，仔细思考发现这个区域应该小于0x208，所以可以在攻击字符串中插入适量的nop指令来达到顺利执行攻击程序的效果。在bufbomb的反汇编代码中找到testn函数，记下它调用getbufn的下一条语句地址（0x8048e15），如图3-19所示：

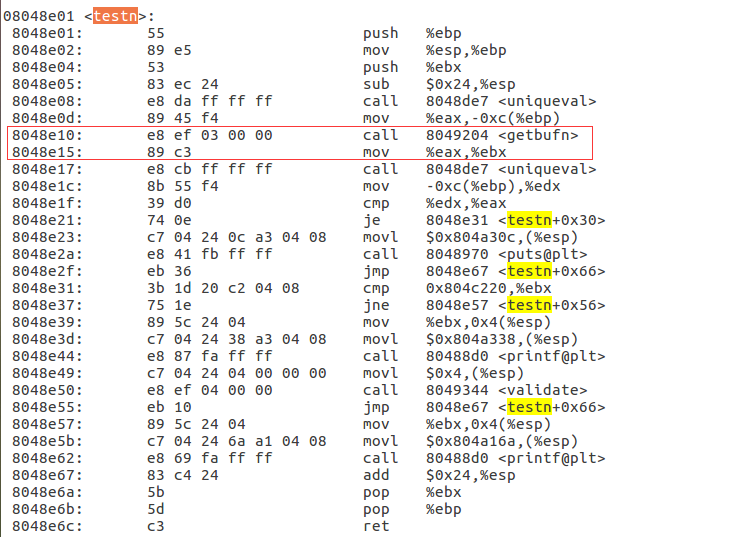


图3-19 testn函数截图

本次实验和level3相差不大，不同的是每次的ebp的值可能是不同的，所以还原ebp只能相对地址。而且每次旧的ebp减去新的ebp的值为常数，而且因为攻击代码运行时新的ebp的值已经被覆盖，所以要通过间接的方法得到新的ebp。将攻击代码保存到nitro.s文件中，如图3-20所示：

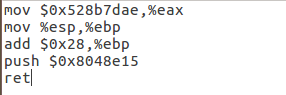


图3-20 nitro.s文件内容

执行指令gcc -m32 -c nitro.s和objdump -d nitro.o进行汇编和反汇编，得到序列为：

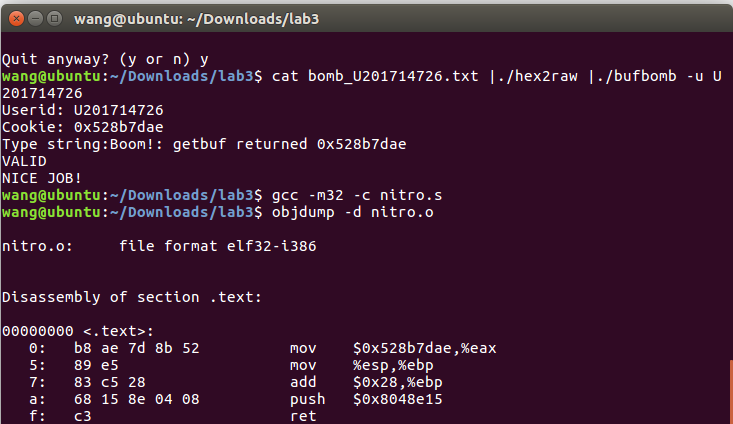


图3-21 所得到的序列

接下来寻找一个合适的返回地址，打开-n开关，执行5次testn后，观察攻击字符串的开始地址，如图3-22、3-23、3-24所示

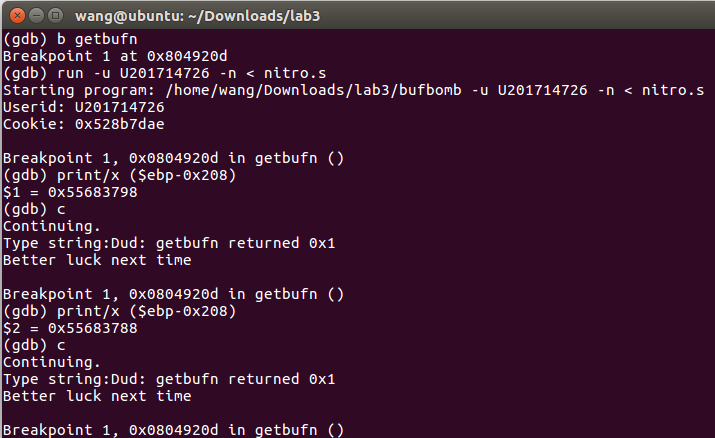


图3-22 所得到的地址

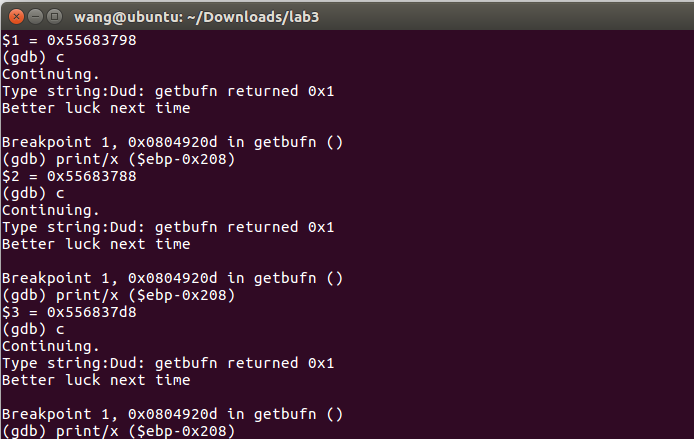


图3-23 所得到的地址

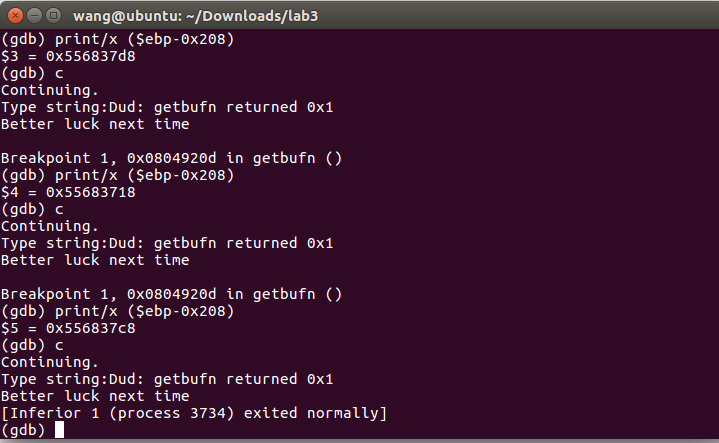


图3-24 所得到的地址

可知地址以此为：0x55683798、0x55683788、0x556837d8、0x55683718、0x556837c8

适当的修改地址，选择0x556837e0时，符合要求，因此总的攻击字符串如图3-25所示：

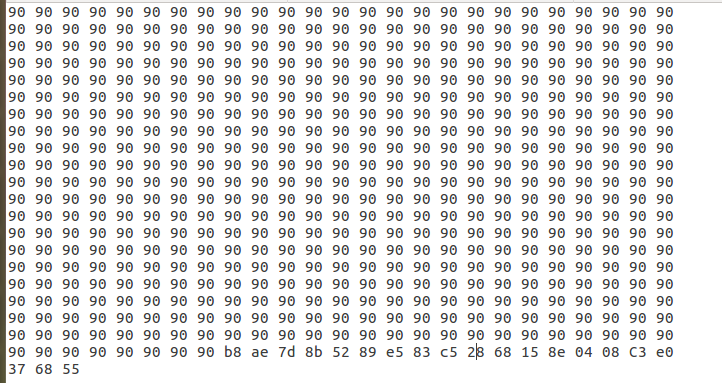


图3-25 总的攻击字符串

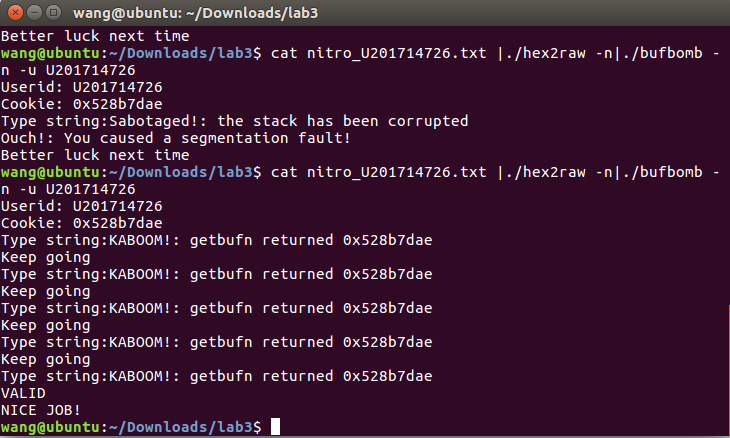
将上述字符串保存到nitro\_U201714726.txt文件中，执行指令：cat nitro\_ U201714726.txt |./ hex2raw -n| ./ bufbomb -n –u U201714726，测试结果如图3-26所示：

图3-26 Level4 nitro测试截图

可见结果正确。

# 3.3 总结